Resilience-Building Technologies: State of Knowledge – ReSIST NoE Deliverable D12

Lorenzo Strigini, Nuno Neves,
Michel Raynal, Michael Harrison,
Mohamed Kaaniche, Friedrich von Henke

DI–FCUL TR–07–26

November 2007

Departamento de Informática
Faculdade de Ciências da Universidade de Lisboa
Campo Grande, 1749–016 Lisboa
Portugal

Technical reports are available at `http://www.di.fc.ul.pt/tech-reports`. The files are stored in PDF, with the report number as filename. Alternatively, reports are available by post from the above address.

ReSIST: Resilience for Survivability in IST

A European Network of Excellence

Contract Number: 026764

Deliverable D12

Resilience-Building Technologies: State of Knowledge

Report Preparation Date: September 2006
Classification: Public Circulation

Contract Start Date: 1st January 2006
Contract Duration: 36 months
Project Co-ordinator: LAAS-CNRS
Partners: Budapest University of Technology and Economics
City University, London
Technische Universität Darmstadt
Deep Blue Srl
Institut Eurécom
France Telecom Recherche et Développement
IBM Research GmbH
Université de Rennes 1 – IRISA
Université de Toulouse III – IRIT
Vytautas Magnus University, Kaunas
Universidade de Lisboa
University of Newcastle upon Tyne
Università di Pisa
QinetiQ Limited
Università degli studi di Roma "La Sapienza"
Universität Ulm
University of Southampton

CONTENTS:

D12: "Resilience-building Technologies: State of Knowledge"

APPENDIX CONTENTS:

Part ARCH

[Arief et al. 2006] B. Arief, A. Iliasov, and A. Romanovsky, "On Using the CAMA Framework for Developing Open Mobile Fault Tolerant Agent Systems", Workshop on Software Engineering for Large-Scale Multi-Agent Systems, pp. 29-35, May 2006.

[Avizienis 2006] A. Avizienis. "An Immune System Paradigm for the Assurance of Dependability of Collaborative Self-Organizing Systems", Proceedings of the IFIP 19th World Computer Congress, 1st IFIP International Conference on Biologically Inspired Computing, pp. 1-6., 2006.

[Becker et al. 2006] S. Becker, A. Brogi, I. Gorton, S. Overhage, A. Romanovsky and M. Tivoli, "Towards an Engineering Approach to Component Adaptation", R. H. Reussner, J. A. Stafford and C. A. Szyperski, editors, Architecting Systems with Trustworthy Components, Vol. 3938 of LNCS, pp. 193-215, 2006.

[Damasceno et al. 2006] K. Damasceno, N. Cacho, A. Garcia, A. Romanovsky, and C. Lucena, "Context-Aware Exception Handling in Mobile Agent Systems: The MoCA Case", Workshop on Software Engineering for Large-Scale Multi-Agent Systems, May 2006.

[Gashi and Popov 2006] I. Gashi and P. Popov. "Rephrasing Rules for Off-The-Shelf SQL Database Servers", Proceedings of the 6th European Dependable Computing Conference, October 2006.

[Gashi et al. 2006b] I. Gashi, P. Popov and L. Strigini "Fault Tolerance via Diversity for Off-the-shelf Products: A Study with SQL Database Servers", manuscript, 2006.

[Gonczy and Varro 2006] L. Gonczy and D. Varro "Modeling of Reliable Messaging in Service Oriented Architectures", Andrea Polini, editor, Proceedings of the International Workshop on Web Services Modeling and Testing, pp. 35-49, 2006.

[Karjoth et al. 2006] G. Karjoth, B. Pfitzmann, M. Schunter, and M. Waidner "Service-oriented Assurance-Comprehensive Security by Explicit Assurances", Proceedings of the 1st Workshop on Quality of Protection, LNCS, to appear in 2006.

[Martin-Guillerez et al. 2006] D. Martin-Guillerez, M. Banâtre and P. Couderc, “A Survey on Communication Paradigms for Wireless Mobile Appliances”, INRIA Report, May 2006.

[Mello et al. 2006] E. Ribeiro de Mello, S. Parastatidis, P. Reinecke, C. Smith, A. van Moorsel, and J. Webber "Secure and Provable Service Support for Human-Intensive Real-Estate Processes", Proceedings of 2006 IEEE International Conference on Services Computing, Chicago, Illinois, September 2006, p495-502. *[This work won FIRST PRIZE in the IEEE International Services Computing Contest, September 2006]*.

[Mian et al. 2006] A. Mian, R. Beraldi, and R. Baldoni, "Survey of Service Discovery Protocols in Mobile Ad Hoc Networks", Technical Report - Midlab 7/06, Dip. Informatica e Sistemistica "Antonio Ruberti", Università di Roma "La Sapienza", 2006.

[Salatge and Fabre 2006] N. Salatge and J.-C. Fabre, "A Fault Tolerance Support Infrastructure for Web Services based Applications", LAAS Research Report No. 06365, May 2006.

[Stankovic and Popov 2006] V. Stankovic and P. Popov, "Improving DBMS Performance through Diverse Redundancy", Proceedings of the 25th International Symposium on Reliable Distributed Systems, October 2006.

[Veríssimo et al. 2006] P. Verissimo, N. Neves, C. Cachin, J. Poritz, D. Powell, Y. Deswarte, R. Stroud, and I. Welch, "Intrusion-Tolerant Middleware: The Road to Automatic Security", IEEE Security & Privacy, Vol. 4, No. 4, pp. 54-62, July/August 2006.

Part ALGO

[Baldoni et al. 2006] R. Baldoni, S. Bonomi, L. Querzoni, A. Rippa, S. Tucci Piergiovanni, A. Virgillito, "Fighting Erosion in Dynamic Large-Scale Overlay Networks", Technical Report - Midlab 9/06 , Dip. Informatica e Sistemistica "Antonio Ruberti", Universit di Roma "La Sapienza", 2006.

[Baldoni et al. 2006-07] R. Baldoni, S. Bonomi, L. Querzoni, A. Rippa, S. Tucci Piergiovanni and A. Virgillito, "Evaluation of Unstructured Overlay Maintenance Protocols under Churn", IWDDS 2006 co-located with ICDCS2006.

[Baldoni et al. 2006-10] R. Baldoni, M. Malek, A. Milani, S. Tucci Piergiovanni, "Weakly- Persistent Causal Objects In Dynamic Distributed Systems", To appear in proc. of SRDS 2006, october 2006, Leeds (UK).

[Baldoni et al. 2006-11] R. Baldoni, R. Guerraoui, R. Levy, V. Quema, S. Tucci Piergiovanni, "Unconscious Eventual Consistency with Gossips", To appear in Proc. of SSS 2006, November 2006, Dallas (USA).

[Correia et al., 2006a] Correia, M., Bessani, A. N., Neves, N. F., Lung, L. C., and Ver´ıssimo, P. (2006a). Improving byzantine protocols with secure computational components. In the report.

[Courtés et al., 2006] Courtés, L., Killijian, M.-O., and Powell, D. (2006). Storage tradeoffs in a collaborative backup service for mobile devices. In Proceedings of the 6th European Dependable Computing Conference (EDCC-6), number LAAS Report #05673, pages 129–138, Coimbra, Portugal.

[Mostefaoui et al. 2006] Mostéfaoui A., Raynal M. and Travers C., Exploring Gafni's reduction aand: from Omega-k to wait-free (2p-p/k)-renaming via set agreement. Proc. 20th Symposium on Distributed Computing (DISC'06), Springer Verlag LNCS #4167, pp. 1-15, Stockholm (Sweden), 2006.

[Raynal and Travers 2006] Raynal M. and Travers C., In search of the holy grail: Looking for the weakest failure detector for wait-free set agreement. Invited paper. Proc. 12th Int'l Conference on Principles of Distributed Systems, (OPODIS'06), To appear in Springer Verlag LNCS, 2006.

[Ryan and Schneider 2006] P.Y.A. Ryan and S. A. Schneider, Prêt à Voter with Re-encryption Mixes, School of Computing Science Technical Report CS-TR: 956, Newcastle University, 2006.

Part SOCIO

[Alberdi et al. 2006] Alberdi, E, Ayton, P, Povyakalo, A. A, and Strigini, L. "Automation Bias in Computer Aided Decision Making in Cancer Detection: Implications for System Design". Technical Report, CSR, City University, 2006. 2006.

[Barboni et al. 2006a] Barboni, E, Conversy, S, Navarre, D, and Palanque, P. "Model-Based Engineering of Widgets, User Applications and Servers Compliant with ARINC 661 Specification". Proceedings of the 13th

conference on Design Specification and Verification of Interactive Systems (DSVIS 2006). 2006. Lecture Notes in Computer Science, Springer Verlag.

[Barboni et al. 2006b] Barboni, E, Navarre, D, Palanque, P, and Basnyat, S. "Exploitation of Formal Specification Techniques for ARINC 661 Interactive Cockpit Applications". Proceedings of HCI aero conference, (HCI Aero 2006). 2006.

[Basnyat and Palanque 2006] Basnyat, S and Palanque, P. "A Barrier-based Approach for the Design of Safety Critical Interactive Application". ESREL 2006 Safety and Reliability for Managing Risk. Safety and Reliability Conference. 2006. Balkema (Taylor & Francis).

[Basnyat et al. Submitted] Basnyat, S, Schupp, B, Palanque, P, and Wright, P. "Formal Socio-Technical Barrier Modelling for Safety-Critical Interactive Systems Design". Special Issue of Safety Science Journal. Submitted.

[Bryans et al. 2006] Bryans, J. W, Ryan, P. Y. A, Littlewood, B, and Strigini, L. "E-voting: dependability requirements and design for dependability". First International Conference on Availability, eliability and Security (ARES'06). 988-995. 2006.

[Harrison and Loer 2006] Harrison, M. D and Loer, K. "Time as a dimension in the design and analysis of interactive systems". (in preparation).

[Harrison et al. 2006] Harrison, M. D, Campos, J. C, Dohery, G, and Loer, K. "Connecting rigorous system analysis to experience centred design". Workshop on Software Engineering Challenges for Ubiquitous Computing. 2006.

[Palanque et al. 2006] Palanque, P, Bernhaupt, R, Navarre.D, Ould, M, and Winckler, M. "Supporting Usability Evaluation of Multimodal Man-Machine Interfaces for Space Ground Segment Applications Using Petri net Based Formal Specification". Ninth International Conference on Space Operations, Rome, Italy, June 18-22, 2006.

[Schupp et al. 2006] Schupp, B, S.Basnyat, S, Palanque, P, and Wright, P. A Barrier-Approach to Inform Model-Based Design of Safety-Critical Interactive Systems. 9th International Symposium of the ISSA Research Section Design process and human factors integration: Optimising company performances. 2006.

[Sujan and Harrison 2006] Sujan, M and Harrison, M. D. "Investigation of structural properties of hazard mitigation arguments". Analysis of the structure of mitigation arguments and the role of barriers or defences with particular reference to the EUROCONTROL Reduced Vertical Separation Minima Functional Hazard Analysis. 2006.

[Sujan et al. 2006a] Sujan, M, Harrison, M. D, Steven, A, Pearson, P. H, and Vernon, S. J. "Demonstration of Safety in Healthcare Organisations". Proceedings SAFECOMP. Springer LNCS. 2006.

Part EVAL

[Alata et al. 2006] E. Alata, V. Nicomette, M. Kaaniche and M. Dacier, "Lessons learned from the deployment of a high-interaction honeypot", LAAS_Report 06-331, April 2006. To appear in Proc. Sixth European Dependable Computing Conference (EDCC-6), Coimbra, Portugal, October 18-20, 2006.

[Albinet et al. 2007] A. Albinet, J. Arlat and J.-C. Fabre, "Robustness of the Device Driver-Kernel Interface: Application to the Linux Kernel", LAAS_Report 06-351, May 2006. To appear in *Dependability Benchmarking of Computer Systems*, (K. Kanoun and L. Spainhower, Eds.), IEEE CS Press, 2007.

[Gönczy et al. 2006] L. Gönczy, S. Chiaradonna, F. Di Giandomenico, A. Pataricza, A. Bondavalli, and T. Bartha, "Dependability evaluation of web service-based processes". In Proc. of European Performance Engineering Workshop (EPEW 2006), LNCS Vol. 4054, pp. 166-180, Springer, 2006.

[Kaâniche et al. 2006] M. Kaâniche, E. Alata, V. Nicomette; Y.Deswarte, M. Dacier, "Empirical analysis and statistical modeling of attack processes based on honeypots" WEEDS 2006 - workshop on empirical evaluation of dependability and security (in conjunction with the international conference on dependable systems and networks, (DSN2006), Philadelphia (USA), June 25 - 28, 2006, pp. 119-124.

[Kanoun and Crouzet 2006] K. Kanoun and Y. Crouzet, "Dependability Benchmarks for operating Systems", International Journal of Performability Engineering, Vol. 2, No. 3, July 2006, 275-287.

[Kanoun et al. 2007] K. Kanoun, Y. Crouzet, A. Kalakech and A.-E. Rugina, "Windows and Linux Robustness Benchmarks With Respect to Application Erroneous Behavior", LAAS report, May 2006. To appear in "Dependability Benchmarking of Computer Systems", (K. Kanoun and L. Spainhower, Eds.), IEEE CS Press, 2007.

[Lamprecht et al. 2006] C. Lamprecht, A. van Moorsel, P. Tomlinson and N. Thomas, "Investigating the Efficiency of Cryptographic Algorithms in Online Transactions," International Journal of Simulation: Systems, Science and Technology, UK Simulation Society, Vol. 7, Issue 2, pp. 63—75, 2006.

[Littlewood & Wright 2006] B. Littlewood and D. Wright, "The use of multi-legged arguments to increase confidence in safety claims for software-based systems: a study based on a BBN of an idealised example", 2006.

[Lollini et. al. 2006] P. Lollini, A. Bondavalli, and F. Di Giandomenico, "A general modeling approach and its application to a UMTS network with soft-handover mechanism", Technical Report RCL060501, University of Firenze, Dip. Sistemi e Informatica, May 2006.

[Rugina et al. 2006] A.-E. Rugina, K. Kanoun and M. Kaâniche, "A System Dependability Modeling Framework using AADL and GSPNs", LAAS-CNRS Report N° 05666, April 2006.

[Salako & Strigini 2006] K. Salako and L. Strigini, "Diversity for fault tolerance: effects of "dependence" and common factors in software development", Centre for Software reliability, City University, DISPO project technical report KS DISPO5 01, Sept 2006.

Part VERIF

[Backes et al. 2006b] M. Backes, B. Pfitzmann, and M. Waidner, "Non-Determinism in Multi-Party Computation"; Workshop on Models for Cryptographic Protocols (MCP 2006), Aarhus, July-August 2006; abstracts as report of ECRYPT (European Network of Excellence in Cryptology, IST-2002-507932).

[Backes et al. 2006c] M. Backes, B. Pfitzmann, and M. Waidner, "Soundness Limits of Dolev-Yao Models"; Workshop on Formal and Computational Cryptography (FCC 2006), Venice, July 2006 (no formal proceedings).

[Micskei and Majzik 2006] Z. Micskei and I. Majzik, "Model-based Automatic Test Generation for Event-Driven Embedded Systems using Model Checkers," in Proc. of Dependability of Computer Systems (DepCoS '06), Szklarska Poręba, Poland, pp.192-198, IEEE CS Press, 2006.

[Micskei et al. 2006] Z. Micskei, I. Majzik and F. Tam, "Robustness Testing Techniques For High Availability Middleware Solutions," in Proc. Int. Workshop on Engineering of Fault Tolerant Systems (EFTS 2006), Luxembourg, Luxembourg, 12 - 14 June, 2006.

[Pfeifer and von Henke 2006] H. Pfeifer and F. von Henke, "Modular Formal Analysis of the Central Guardian in the Time-Triggered Architecture", Reliability Engineering & System Safety, Special Issue on Safety, Reliability and Security of Industrial Computer Systems, Elsevier Ltd., 2006, to appear.

[Serafini et al. 2006] M. Serafini, P. Bokor and N. Suri, "On Exploiting Symmetry to Verify Distributed Protocols", Fast Abstract, International Conference on Dependable Systems and Networks (DSN) 2006.

[Waeselynck et al. 2006] H. Waeselynck, P. Thévenod-Fosse and O. Abdellatif-Kaddour, "Simulated annealing applied to test generation: landscape characterization and stopping criteria", to appear in *Empirical Software Engineering*, 2006.

Introduction to ReSIST deliverable D12

L. Strigini

City University

This document is the first product of work package WP2, "Resilience-building and -scaling technologies", in the programme of jointly executed research (JER) of the ReSIST Network of Excellence.

The problem that ReSIST addresses[1] is achieving sufficient resilience in the immense systems of ever-evolving networks of computers and mobile devices, tightly integrated with human organisations and other technology, that are increasingly becoming a critical part of the critical infrastructure of society. These systems – large, networked, evolving systems constituting complex information infrastructures, perhaps involving everything from supercomputers and huge server "farms" to myriads of small mobile computers and tiny embedded devices – referred to in ReSIST as "ubiquitous computing systems", are essential to support and provide Ambient Intelligence (AmI) in the developing Information Society. Features of these ubiquitous systems are already present in embryo in existing infrastructure, and the risk of a "dependability and security gap" undermining the potential societal benefit from these systems is evident. A resilience approach to building and operating these systems is essential: they must be capable of surviving and delivering sufficiently dependable and secure service despite the inevitable residual development and physical faults, interaction mistakes, or malicious attacks and disruptions that their very scale, complexity and openness make more likely.

ReSIST identifies the research challenge here in the need for *scalable* resilience of policies, algorithms and mechanisms. Quoting from ReSIST's Description of Work: "The current state-of-knowledge and state-of-the-art reasonably enable the construction and operation of critical systems, be they safety-critical (e.g., avionics, railway signalling, nuclear control) or availability-critical (e.g., back-end servers for transaction processing). The situation drastically worsens when considering large, networked, evolving, systems either fixed or mobile, with demanding requirements driven by their domain of application". Therefore,

> "The initial programme of jointly executed research (JER) on resilience-building technologies will focus on surveying and developing the expertise available within the network with a view to identifying gaps in the collective knowledge portfolio that need to be filled for these technologies to meet the scaling challenges of ubiquitous systems.

[1] This introduction only gives a very brief summary of ReSIST's motivations and plans, which are described in full in ReSIST's Description of Work document (Annex I to the contract).

> To this end, months 1 to 9 will be devoted to an exchange of know-how between the partners, catalysed by the production of a deliverable at month 9, summarising the current state of knowledge and ongoing research by the partners on methods and techniques for building resilient systems."

The present document is that first deliverable, D12. This work is a basis for the second phase, which will last until month 18 in the life of the network and will focus on

> "the scaling challenges identified by ReSIST, identifying a roadmap of integrated research for employing the current resilience-*building* technologies to develop the required resilience-*scaling* technologies identified in Section 6.1, i.e.: evolvability, assessability, usability and diversity."

The work in this first phase has been divided among five working groups (each with active participation from the ReSIST partners that work in the corresponding research area) dealing with different aspects of resilience building and the corresponding subdisciplinary areas:

- Resilience architecting and implementation paradigms (WG Arch)
- Resilience algorithms and mechanisms (WG Algo)
- Resilient socio-technical systems (WG Socio),
- Resilience evaluation (WG Eval)
- Resilience verification (WG Verif)

This document is therefore made up of five parts, each produced by one of the working groups. Each working group's activity started with a session during the first Plenary Network Meeting, at which the participants presented their backgrounds and viewpoints. This was followed by further interaction, via E-mail, shared "wikis" and meetings, depending on the working group. Each working group aimed to cover the relevant areas of knowledge and recent contributions of ReSIST partners, and to situate them with respect to the state of the art and related research in the field. Each part of the deliverable was then reviewed with an emphasis on the viewpoint of scientists who are not specialists of the subdisciplines covered, so that the document can serve as an introduction to the problems relevant for ReSIST in the area covered, besides documenting the advanced results produced by ReSIST members, for those colleagues who already perform research in closely related areas.

This document is made up of three layers dealing with the subject matter in increasing levels of detail: this brief overview (first layer) is followed by five survey-style parts, one for each WG, which perform the role of introduction for non-specialists (second layer); last, a set of appendices contain selected technical papers produced (some published, some being prepared for publication) since the start of ReSIST (third layer). These "third layer" papers, together with the other references in the second layer, allow readers to find more in-depth reading on specific topics of interest, and can be found through references listed in boldface in the "second layer" parts.

A brief outline of the five "second layer" parts follows.

Part Arch – Resilience Architecting and Implementation Paradigms

Part Arch surveys four lines of research, historically originating from different technical communities or application areas, that must converge in order to address the topic of scalable resilience in ubiquitous computing systems.

Two of these research areas, “service oriented architectures” and “mobile services and their infrastructures”, have as their goal to exploit the existence of large-scale networks to provide enhanced flexibility of network-based services, respectively from the viewpoint of interoperability (across differences in software technology and across organisational boundaries) and of mobility. In this sense, this research is itself motivated by some of the scalability goals that ReSIST addresses.

The other two research areas surveyed, “Building resilient architectures with off-the-shelf components” and “Intrusion tolerant architectures”, are about pressing problems that arise from the state of commercial IT technology even for “non-ubiquitous” systems; but these problems are of vital importance for ReSIST because this same technology is the basis on which ubiquitous systems are built. The market situation that gives us low-cost off-the-shelf components also makes them insufficiently trustworthy, requiring fault tolerance as a safeguard. This is also part of the motivation for “intrusion tolerance”, the application of fault tolerance techniques to security. Researchers in these two lines of research were originally motivated by concern about different types of faults (design faults vs malicious attacks), but acknowledge that system architectures must address both concerns. An interesting question is how their approaches scale to integrate with the approaches developed in the other two chapters.

Part Algo – Resilience Algorithms and Mechanisms

Part Algo discusses some of the essential categories of algorithms and protocols that underlie fault tolerance in distributed systems. As such, all of these, seen as research topics, have issues of scalability as part of their basic formulation, where scalability is defined in terms of number of nodes connected and number of faults to deal with. The ReSIST resilience-scaling challenge has other facets, however, like the integration of systems built on separate lower-level infrastructures or applying different basic protocols, that may pose new problems in the area of algorithms.

This part starts with reviewing the general problems underlying fault-tolerant distributed computing: the problem of achieving consistent views between communicating entities despite arbitrary, possibly subtle faults, a problem that has been heavily studied over the last three decades because the limits to its solutions affect a broad variety of fault-tolerant algorithms; and the even more general problem of recognising faulty (or permanently faulty) components so as to properly manage reconfiguration and preserve resilience. Next, chapters about “cooperative backup mechanisms”, “cooperation incentive schemes”, “overlay network connectivity” deal with “peer-to-peer” and similar approaches aiming at fully scalable, fully decentralised resilient services. The chapter titled “Wait-free asynchronous computing” deals with an intriguing category of distributed object designs with provable dependability and timeliness properties. Last, we have a summary of development driven by a specific, challenging application, E-voting (also a topic of Part-Socio), where algorithms have been developed that guarantee the traditional requirements of secrecy of vote and protection from tampering, only with much greater confidence (as far as the algorithmic part is concerned) than the traditional manual voting procedures.

Part Socio – Resilient Socio-Technical Systems

Part Socio deals with one of the greatest challenges in ReSIST, i.e., integrating the analysis and design of the technical and human-organisational subsets of ubiquitous systems. ReSIST acknowledges that as “ubiquitous” information and communication systems are deeply interwoven with human organisations and society, it is the “socio-technical system” that they form together, not only its software-based components. that must be the focus of analysis for its resilience characteristics.

This part thus examines the process of reasoning about complex socio-technical systems, in order from the tasks of elucidating the intended and actual behaviour of people in the system at a descriptive level, to those of verifying and assessing the proper functioning of the system, first logically (and possibly formally, in the computer science sense of this word), to support for instance the finding and elimination of potential design faults, and then probabilistically, to evaluate and compare systems.

An important topic throughout Part Socio is the tension between the need for designers and assessors to reason about both the human and automated parts of a system in combination, and the differences that make the application of similar techniques to both difficult. Socio-technical systems will often, while behaving satisfactorily, actually work in ways that differ from those assumed by their designers and managers. These discrepancies are a frequent cause of failure in the process of automation of organisations, often through loss of the resilience previously provided by the human operators: hence the emphasis on techniques for understanding, and languages for modelling, the structure and organisation of socio-technical systems. Another area where the difficulties of applying common techniques to humans and to the automated parts of a system are underscored is that of probabilistic assessment extended to human action. Much expertise is available within ReSIST on these various topics, but it is generally acknowledged that hard problems remain open even for relatively simple systems, and thus there are important challenges ahead in porting this knowledge to ubiquitous systems.

Part Eval – Methods and Tools for Resilience Evaluation

Part Eval deals with evaluation techniques, with an emphasis on quantitative evaluation, in its dual role of guiding design decisions and assessing systems as built, and in its components of mathematical modelling and empirical measurement. This part starts from considering the stochastic modelling techniques that have long been the basis of dependability assessment, and the basic scalability challenge created by systems with increasing numbers of components or states. It then proceeds to address other challenges in extending the domain of quantitative evaluation.

The first chapter, on “compositional modelling for large and evolving systems” describes progress in addressing the scalability problems both in the construction and in the solution of models of stochastic processes used for performance and dependability assessment, and then summarises encouraging applications to evaluation of systems that have some degree of “ubiquitousness”, e.g., Web-based services or mobile telephone networks.

The next issue addressed is the extension of quantitative techniques to security assessment. A chapter on “evaluation with respect to malicious threats” briefly introduces the common security evaluation criteria, the contributions given by ReSIST partners to defining the topic of quantitative assessment of security and to developing empirical measurement methods to apply to attacks and threat environments. This is still a young research area, and its potential has yet to be explored; some immediate targets for its development are suggested.

Another current challenge is “dependability benchmarking”, the development of repeatable measurement practices for comparing and selecting systems and components. Here, standard “loads” of workload and injected faults are used to evaluate resilience. This area is still being developed, mainly with reference to small and self-contained systems.

The next chapter deals with assessment of “diversity”, shorthand for assessing the probability of common-mode failure between redundant components, and the effects of the precautions (various ways of diversifying the components or their environments) meant to reduce it. ReSIST partners have made substantial

contributions in this difficult area, but most of the work has concerned simple embedded systems: the models created are general enough to address some aspects of diversity in ubiquitous systems, but pose a substantial challenge for extending them to deal with large numbers of diverse components. Last, a chapter on "dependability cases" deals with the difficulties in the proper organisation of the complex evidence pertinent to the evaluation of any real system, including modelling, statistical evidence and many forms of informal but relevant evidence.

Part Verif – Methods and Tools for Verifying Resilience

Part Verif deals with two sets of important techniques – formal methods, based on mathematical logic and related to proving properties of systems, and method related to testing, for the empirical demonstration of properties of system operation – all applied to resilience-related properties and the mechanisms intended to support them.

Logic-based techniques – formal methods – are covered in the first three chapters, "Deductive Theorem Proving", "Model Checking", "Symbolic Execution and Abstract Interpretation", which document multiple examples of application of advances by ReSIST partners to verifying fault tolerance and security properties of systems and protocols. All these areas are concerned with, and making advances in, scalability in the size of models to be treated, although application to ubiquitous systems may bring new problems not just of size of models but for instance of heterogeneity of system components. A chapter on "verification of systems containing cryptography" demonstrates results in an area that is problematic for "standard" formal methods because the properties of cryptographic subsystems are naturally defined in probabilistic, rather than deterministic, terms and based on computational complexity.

The empirical techniques are covered in the chapter on "robustness testing", dealing with techniques, like fault injection, for exposing design or physical faults in systems that are meant to be resilient and robust. There is wide experience within ReSIST, which has led to an armoury of multiple techniques, which vary in the methods for choosing stimuli, with much research on systematic ways for selecting stimuli with high "power" in revealing faults, as well as in how the stimuli are applied to the system. The transition to ubiquitous systems poses multiple challenges, including the great variety of threats against which they must be robust, and the difficulty in building experiments (injecting stimuli and observing their effects) in systems with mobile components and/or spread over large geographical areas.

Summary

This "state of knowledge" document is to serve as a basis for the development of ReSIST's research roadmap and a stepping stone in the process of integration within the ReSIST network. In pursuing this internal goal, the five working groups have also produced substantial surveys that will be useful for all researchers.

As can be seen, in all areas ReSIST has a substantial body of knowledge that can be applied to the challenge of ubiquitous systems. The obstacles to be overcome in this application vary greatly between these areas. Some research areas have successfully addressed problems in small systems but may have yet to deal with scaling up to large size systems; other research areas have a focus on scalability problems as part of their very definition, and the difficulties may come mainly from the need to deal with combinations of heterogeneous systems, or the difficulty of incorporating considerations of human behaviour into a computer-oriented discipline; others still face substantial unsolved questions for the kinds of systems to which they were originally applied, and a start in applying them to ubiquitous systems may bring progress to

the whole area. Last, all face the need for more integration between different techniques (models, architectural principles, protocols), between consideration of accidental and of malicious faults, and between automated and human subsystems.

Part Arch – Resilience Architecting and Implementation Paradigms

Co-ordinator: N. Neves

Contributors: A. Avizienis[9], J. Arlat[5], R. Beraldi[8], C. Cachin[4], M. Correia[6], Y. Deswarte[5], P. Ezhilchelvan[7], J.-C. Fabre[5], L. Gonczy[1], Q. Inayat[7], I. Kocsis[1], C. Lac[3], N. Neves[6], A. Pataricza[1], P. Popov[2], D. Powell[5], H. V. Ramasamy[4], A. Romanovsky[7], N. Salatge[5], L. Strigini[2], R. Stroud[7], P. Veríssimo[6]

[1]Budapest University, [2]City University, [3]France Telecom, [4]IBM Zurich, [5]LAAS-CNRS, [6]University of Lisboa, [7]University of Newcastle, [8]University of Roma, [9]Vytautas Magnus University

Chapter co-ordinators:

1 - Service Oriented Architectures: J-C. Fabre;

2 – Mobile Services and their Infrastructures: R. Beraldi and C. Lac;

3 – Building Resilient Architectures with Off-the-shelf Components: P. Popov;

4 – Intrusion Tolerant Architectures: N. Neves

Introduction

Future ubiquitous computing systems will create important new challenges to system architects, who will have to devise solutions that will ensure the necessary levels of resilience and survivability in these pervasive information infrastructures. These challenges result from changes in the way these systems will have to be implemented, which can be functional, technological or environmental. A few examples of such changes are: the growth of the complexity and size of these systems; the heterogeneity of the networks and devices (from super-computers to tiny embedded devices); the flexible and evolving nature of the infrastructures (e.g., ad-hoc or spontaneous networks of mobile nodes); and newer failure modes related to malicious behaviours of users and administrators.

The background of the ReSIST partners on architectural and implementation paradigms covers a significant number of topics, which were addressed in several successful European research projects. Several of the ReSIST partners have been working on the resilient aspects of computing systems for the past 20 years, and more recently they have started to address the challenges created by larger scale systems (see for example the projects MAFTIA, GUARDS, and DSOS). This part of the ReSIST D12 deliverable provides a summary of the ReSIST partners' current state of knowledge and ongoing investigations on architectural solutions, with a special focus on the areas that are relevant to the project.

This part includes four main chapters, each one corresponding to a key research area that has to be considered and integrated in ReSIST, in order to support scalable resilience in ubiquitous systems. The first two chapters, "Service Oriented Architectures" and "Mobile Services and their Infrastructures", address issues related to the large-scale, heterogeneous, and evolving nature of pervasive information infrastructures. The other two chapters tackle a set of problems that have (and will) contributed to the rise in concerns on the dependability and security aspects of current (and future) computing systems. These chapters are about "Building Resilient Architectures with Off-the-shelf Components" and "Intrusion Tolerant Architectures".

Service Oriented Architectures

Service oriented architectures (SOA) were introduced as a new approach to organize and access distributed services spread over large-scale heterogeneous infrastructures. This type of architecture provides a uniform way for clients to discover, access, and interact with remote servers, potentially owned and operated by different organizations. One of the main characteristics of SOA solutions is flexibility because services can be discovered and reached dynamically, without any prior knowledge of their underlying implementations. Moreover, SOA promotes service reuse which can help organizations to adapt more rapidly to changing environmental conditions, and facilitate the interconnection of existing IT systems. The "Service Oriented Architecture" Chapter introduces the fundamental components of the architecture and their relations, and then presents several examples, research projects and current standardization efforts related to resilience issues. These topics are addressed in a set of sections which encompass a number of aspects of the architecture, including the resilience of the executive support, the reliability of the SOA protocols, security and authentication concerns, and composition and orchestration of services.

Mobile Services and their Infrastructures

Future ubiquitous computing systems will be complex information infrastructures involving many kinds of distinct devices (e.g., large server farms and tiny mobile computers), some of which will be interconnected by a collection of wireless transmission technologies. Currently, even though the wireless networks arena is still dominated by voice-oriented cellular infrastructures, the increasing availability of small computer devices is leading to newer architectures. These computers have mobility as their main attribute, and they support various types of networks, ranging from personal- to wide-area networks (e.g., Bluetooth, WiFi, UMTS). The kind of services that will accessible in these architectures will be influenced by numerous factors, such as the lifetime of the network and the transmission range between devices. The "Mobile Services and their Infrastructures" Chapter analyses the resilience aspects of some mobile architectures. In particular, it considers cellular-based infrastructures and mobile ad hoc networks (MANET).

Building Resilient Architectures with Off-the-shelf Components

Cost pressure is one of the main drivers for the creation and widening of the dependability and security gap in current and future large evolving information systems. The use of off-the-self (OTS) components has been common in the business and government sectors for some time, but in recent years economics has created a trend where these components are also utilized in industries that deal with critical computer applications (e.g., telecommunication and electrical sectors). Many low-priced OTS components, however, have records of poor dependability or lack evidence of sufficient dependability for such applications. Even though vendors are continuously improving their development practices, in many cases, the resulting components are still far from being fully satisfactory from a dependability perspective. From the resilience view point there is however an upside on using OTS components – diversity in redundancy can be exploited as an effective defense against common-mode accidental failures and malicious attacks. The chapter "Building Resilient Architectures with Off-the-shelf Components" analyses several research lines on resilience with OTS components. Namely, it describes efforts to identify flaws in OTS software, the use of diversity both for fault-tolerance and security, and computer systems that are capable of being adaptive.

Intrusion Tolerant Architectures

Complexity will grow in ubiquitous information infrastructures, where support for large numbers of evolving networks of computers and mobile devices is needed. Due to their pervasiveness, these infrastructures will be the ideal target for both amateur hackers and professional criminals. Therefore, in order to guarantee the necessary resilience and survivability of these infrastructures, capabilities have to be provided to deal with malicious attacks and disruptions, in addition to accidental failures, for instance due to residual development mistakes. Intrusion tolerance is a new approach that has emerged during the past decade that merges ideas from the security and dependability communities. Its objective is to cope with a wide set of faults, ranging from random defects to successful intrusions. The focus of intrusion tolerance is on ensuring that systems remain in operation (possibly in a degrade mode) and continue to provide services despite the various kinds of faults. The chapter "Intrusion Tolerant Architectures" discusses the main mechanisms for building intrusion tolerant systems, and reports on recent advances on architecting this kind of systems. For instance, the following mechanisms are addressed in light of intrusion tolerance: auditing, error processing and intrusion detection.

1 – Service Oriented Architectures

Introduction

The term *Service Oriented Architecture* (SOA) refers to a style of information systems architecture in which distributed applications are constructed by combining loosely coupled and interoperable *Services* that inter-operate according to a *Service Contract* that is independent of the underlying platform and programming language used to implement the service [Wikipedia 2006]. Because the service contract hides the implementation of the service, SOA-compliant systems can be independent of any particular development technology or execution platform. For example, a service implemented in Java running on the J2EE platform could interact with a service implemented in C# running on the .NET platform.

In principle, SOAs can be realised using a variety of different middleware protocols (for example, CORBA or Jini), but in practice, the term SOA is often used to refer to an SOA implemented using the web services protocol stack (see Figure 1.1). A *Web Service* is essentially just a software component with a well-defined interface that can be accessed programmatically over a network using standard protocols. In this sense, web services are no different from conventional client-server applications built using middleware technologies such as CORBA. However, the distinguishing characteristic of web services is the use of XML-based protocols and languages to describe the interface to the web service and the messages that it understands and generates. In particular, the interface to a web service is described using WSDL (*Web Service Description Language*), and other systems interact with the web service using SOAP messages (*Simple Object Access Protocol*), which are typically conveyed over transport protocols such as HTTP using an XML serialization. Thus, SOAP is a protocol for delivering XML messages to applications, whereas WSDL is a language for describing the format of the messages that an application can understand and generate.

Web services are typically implemented on top of a multilayer platform that hosts the web service within an application server, running on top of a web service protocol stack, and a conventional operating system. In order to interact with a web service, it is necessary for an application to obtain a copy of the web service's interface description, which might be stored at a known location associated with the web service endpoint, typically described by a URI (*Uniform Resource Identifier*), or else discovered dynamically using a protocol such as UDDI (*Universal Description, Discovery, and Integration*), which is a large and complex specification of a distributed registry for publishing and discovering web services.

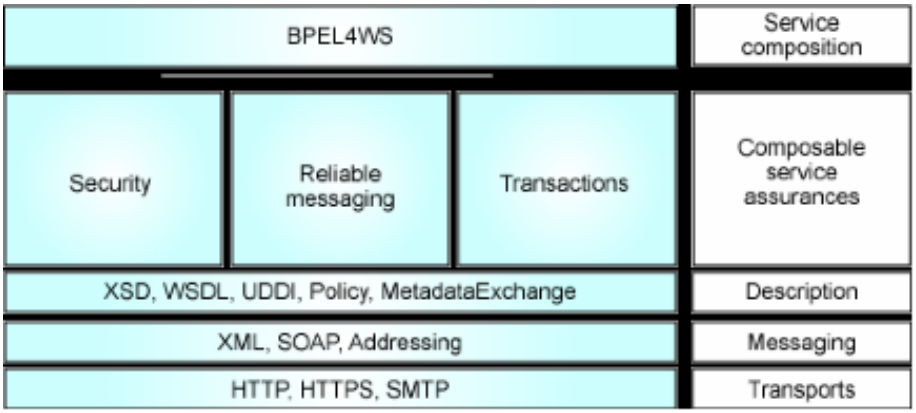

Figure 1.1 – Web services protocol architecture (from [IBM and Microsoft 2003]).

Interactions with web services take place via SOAP messages, but because SOAP is independent of the underlying transport protocol, it is necessary to use additional layers of web service protocols to provide end-to-end guarantees about the secure and reliable delivery of messages. Such guarantees can be viewed as composable service assurances that can be added to the basic SOAP messages [IBM and Microsoft 2003]. Furthermore, since some web services may require particular service attributes to be provided, and potential web service users may wish to know which attributes are supported by a given web service, it is necessary to augment the WSDL description of a web service interface with additional information about the web service's requirements and support for these attributes, for example, whether a transactional context or security token is required. Thus, it is necessary to define additional languages for describing policies for interacting with a web service. Finally, since new web services can be built from existing web services by composing them together according to a particular application workflow, it is also desirable to define a language for web service composition. Taken together, these various web service standards define an interoperable web services protocol architecture, such as the one depicted in Figure 1.1. Such diagrams are not intended to imply a strict functional layering, but rather indicate the relationship between the different functional areas. Thus, there are specifications for XML messaging, languages for describing web services and additional web service assurances, and languages for service composition.

It is clearly important to agree on a standard set of such protocols in order to prevent application developers from inventing their own solutions, and thus not only wasting effort but also potentially inhibiting interoperability between different web service applications. However, unfortunately, web service standardisation is proving to be a technical and political battleground between open standards and proprietary standards. Although there is broad agreement about core web service standards such as SOAP and WSDL, in general, web service standards are very immature, particularly above the messaging layer, and there are a number of rival or overlapping proposals for standardising particular aspects of web services.

Although the benefits of Service Oriented Architectures are certainly of high interest for applications with important dependability requirements, the lack of mature advances regarding the resilience of such architecture is a major impairment to their use in large critical applications. Thus, the big challenge is how to build reliable/secure distributed applications out of unreliable/insecure Web Services and communication infrastructures.

Regarding resilience and dependability of SOAs, we propose to classify the various works and contributions in six research domains. This separation is not always clear as problems and solutions often overlap several domains. It is however a way to summarize our state of knowledge on SOA and related resilience issues:

- Resilience of the executive support for *Web Services*,
- Resilience assessments and tools,
- Security and authentication issues
- Reliability issues,
- Transaction, composition and orchestration.
- Quality of service requirements

The first dependability issue for SOA applications is to improve the reliability of the runtime support of the Web Services, i.e., the platform on which the web service is executed. Conventional dependability techniques can be used to address this aim, from both an architectural and evaluation viewpoint. For example, relevant techniques include replication at various hardware and software levels (OS, middleware, etc.), failure mode analysis using fault injection at various levels, and failure mode analysis proving inputs to the design of fault tolerance mechanisms. This is why a large portion of current work in this area tackles the problem in this way. In the same way, conventional techniques can be applied to the communication infrastructure and transport protocols.

The second dependability issue is to tackle the problem at the level of actual SOA concepts, i.e., all protocols and software components used to interact with Web Services. The works we are aware of currently address security and reliability issues, transactional problems, flexibility of dependability solutions with respect to the application needs, and orchestration of large-scale applications based on Web Services. Clearly, there are still many open subjects and difficult issues to address in this second dimension. It is worth noting however that although the backbone of an SOA may introduce multiple fault sources [Ingham et al. 2000], the architecture also allows for design diversity in the form of alternative services and communication channels that may be available over the Internet.

1.1 Research on SOA

In the next sections, we report on some significant examples, research projects and current standardization efforts targeting various aspects of resilience for Service Oriented Architectures, following the six research and development topics mentioned above.

1.1.1 Resilient executive support for Web Services

A framework for improving dependability of web services was proposed in DeW (*A Dependable Web Service Framework*) [Alwagait and Ghandeharizadeh 2004], which can be understood as a register containing the address of different copies of a web service. This register guarantees the physical-location-independence and thus a web service-based application is able to continue running as long as a reference of an available copy is reachable through the register. This framework enables non-stop operation in the presence of crash faults affecting the web services or service migration. Active-UDDI [Jeckle and Zengler 2003] is based on a similar research approach.

FT-SOAP [Liang et al. 2003] enables a service provider to replicate Web Services using a passive replication strategy. This approach is based on interceptors at the SOAP layer enabling the client application to redirect request messages towards replicas in case of failure of the primary. At the server location, interceptors are

used to log messages, to detect faulty replicas and to manage available replicas. The important problem of the state transfer is controlled internally, i.e., it is part of the implementation of the service. A similar passive replication approach has been developed in the JULIET project, using .NET [Murty 2004, Dialani et al. 2002]. In both cases, the approach relies on a specific software layer that must be installed at the client and at the provider platform, which can raise interoperability problems due to inconsistency with the notion of SOA.

FTWeb [Santos et al. 2005] is another example of an infrastructure providing active replication strategies for web service-based applications. This project is based on the FT-CORBA standard, WS-Reliability [SUN 2003] (see section on reliability of SOA protocols) and a global ordering service [Defago et al. 2000] of client requests to ensure replica consistency. This project also introduces a specific software layer that may impair interoperability. Beyond that, it is clear that this architecture strongly depends on a particular middleware support, namely CORBA, and thus Web Services must be implemented as CORBA objects.

Thema [Merideth 2005] is a another middleware-based implementation of fault tolerance mechanisms for web services, more precisely aiming at tolerating Byzantine Faults. The communication service relies on former work [Rodrigues et al. 2001], providing reliable multicast and consensus policies. WS-FTM (*A Fault Tolerance Mechanism for Web Services*) [Looker and Munro 2005] is a similar research effort on consensus issues for web services.

1.1.2 Resilience assessments and tools

Most of web services must be considered as black-box components, which means that their development process, their design features, and their robustness in the presence of faults is not known. This kind of situation is not new, and also applies to the use of COTS components in dependable systems for which a lot of work has been carried out, targeting operating systems [Koopman and DeVale 1999], real-time micro-kernels [Arlat et al. 2002] and middleware such as CORBA [Marsden *et al.* 2002, Marsden 2004]. However to our knowledge, the assessment of the resilience of SOA is rather limited today and currently relies on conventional benchmarking approaches.

Although failure mode analysis has not been performed yet for web services, we can however mention the assessment by fault injection (SWIFI) carried out in the OGSA platform based on web services [Looker and Xu 2003, Looker *et al.* 2004, Looker *et al.* 2005]. One of the results of this work is the development of WS-FIT (*Web Services Fault Injection Tool*) [Looker et al. 2004] for the assessment by fault injection of the SOAP protocol. In practice, the SOAP parser Axis 1.1 is instrumented to inject faults on input/output request messages. Recent results presented in [Silva et al. 2006] also reveal that version 1.3 of the Axis software should not be used in business-critical applications because of several problems like memory-leakage, performance degradation and hang-up situations of the server that require manual restart intervention. This type of work relates to the discussion of dependability benchmarking in the chapter on Evaluation of System Resilience elsewhere in this deliverable.

One benefit of such work is to propose a fault model for web service based applications, i.e., the kind of faults affecting a web service in operation [D. Cotroneo et al. 2003] and [Gorbenko et al. 2004]. Beyond physical faults affecting the runtime support, communication faults are one of the most important sources of faults for large-scale applications on the Internet [5, 6]. However, because of the complexity of the web service infrastructure at runtime (protocol analysers, dynamic code generation, application servers, virtual machines / operating systems), software faults must be considered as a first class of problems [Ingham et al. 2000]. For example, the SOAP parser can be subject to development faults, which could lead to an incorrect

analysis of messages and/or a wrong mapping of data types. In short, the development of large-scale critical application in this context must take into account both physical and software faults affecting the executive and communication infrastructure but also evolution faults, e.g., inconsistency between WSDL documents and stubs produced from older versions.

1.1.3 Security and authentication issues

A lot of work has been carried out regarding security in Services Oriented Architectures built out of web services, and this part of the web services protocol stack is relatively mature. There are several security-related web service standards, in particular WS-Security [OASIS 2004], which aims at providing end-to-end security including authentication (using security tokens), confidentiality and integrity of messages. WS-Security, which is based on XML-Signature and XML-Encryption, is implemented at the SOAP request level by using non-functional elements in the SOAP header and by encrypting the message body. XML-Signature [Bartel et al. 2002] is a specification targeting the creation and the processing of electronic signatures of XML documents or part of them. XML-Encryption [Imamura et al. 2002] is a specification of object encryption and formatting of the encrypted result in XML.

WS-Security defines a basic framework for transmitting web service messages securely. Above this layer there is less agreement, but IBM and Microsoft have proposed the following set of layers and software abstractions:

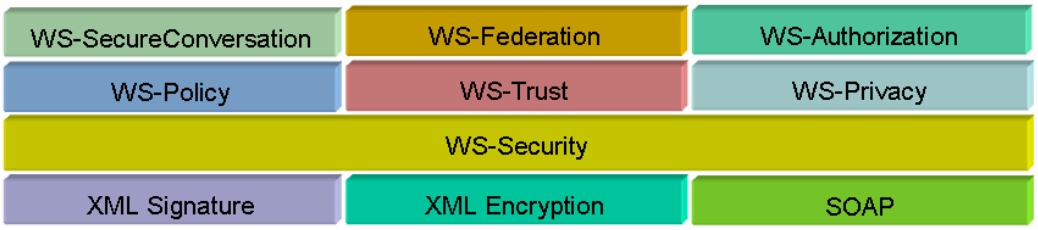

Figure 2.2: Abstractions and Software Layers for Web Services Security (from [IBM and Microsoft 2002])

WS-Policy [Bajaj et al. 2006], WS-Trust [Anderson et al. 2005] and WS-Privacy build directly on WS-Security to add higher level security characteristics. Thus, WS-Policy specifies the security contract, describing how to express the security requirements of the provider and the capabilities of the client, WS-Trust specifies the model of the mechanisms to establish trusted relations, either directly or indirectly via trusted third party services, whilst WS-Privacy (in progress) is a language for the specification of privacy features (on both machine- and human-readable format) that can be interpreted by user agents.

Higher-level protocols build on top of these protocols to solve interoperability issues between heterogeneous security approaches. Thus, *WS-SecureConversation* [Anderson et al. 2004], *WS-Federation* [Bajaj et al. 2003], and *WS-Authorization* (in progress) provide a general framework for authorization mechanisms in web services architectures.

1.1.4 Reliability of SOA protocols

There is clearly a need for web service messages to be delivered reliably as well as securely, and there are many examples of reliable messaging systems that can provide such guarantees. However, web services cannot depend on the semantics of the underlying transport protocols, and thus, reliable messaging must be

implemented at the SOAP level to ensure end-to-end reliability guarantees and interoperability across a range of different transport protocols.

A reliable messaging protocol for web services essentially defines an abstraction of a reliable message-processing layer that hides the details of the underlying mechanism used to ensure reliable delivery, which is typically some form of message-oriented middleware. The basic reliability guarantee provided is "best effort" delivery, perhaps within a specified time limit, or else a failure indication.

Unfortunately, for historical and perhaps political reasons, there are two rival standards for reliable messaging for web services, namely *WS-Reliability* [SUN 2003] and *WS-ReliableMessaging* [Microsoft 2004]. Although the two specifications offer very similar capabilities with respect to reliable messaging, they seem to have a very different approach to specifying the policy associated with a reliable messaging channel. *WS-ReliableMessaging* appears to take a rather static view in which the policy is negotiated as part of setting up the channel, and then the agreed parameters apply to the whole sequence of messages. This means that it is not necessary to specify the parameters as part of each message, but it also means that it is not possible for the policy to be adapted dynamically except by shutting down the channel and starting again. In contrast, *WS-Reliability* seems to allow much more flexibility, but at the cost of some overhead associated with each message. Ultimately, the performance of a reliable messaging system depends on the underlying implementation rather than the protocol, but it would seem that the *WS-Reliability* proposal allows the implementer more flexibility.

1.1.5 Transaction, composition and orchestration

In addition to BTP *(Business Transport Protocol)* [Ceponkus et al. 2002], an older technology, *WS-AtomicTransaction* [Cabrera *et al.* 2005b], *WS-BusinessActivity* [Cabrera *et al.* 2005c] and *WS-Coordination* [Cabrera *et al.* 2005a] are three complementary specifications supported by IBM, Microsoft and BEA that should support the implementation of synchronous short duration and ACID transactions (*WS-AtomicTransaction*) as well as asynchronous long-running business transactions (*WS-BusinessActivity*).

Transaction management provides a basic infrastructure for coordinating the execution of web services. However, a service-oriented application can be composed of several web services, which requires some sort of language for "composition", "orchestration" or "choreography" of services. The web services included in the composition are coordinated by a workflow that constitutes the business process of the composition.

Several initiatives specify such business processes using XML documents. Three similar specifications have been proposed for the workflow oriented applications: XLANG [Thatte 2001] by Microsoft, WSFL *(Web Services Flow Language)* [Leymann 2001] by IBM, and WSCL *(Web Services Language Conversation)* [Banerji et al. 2002] by Hewlett-Packard. BPEL4WS *(Business Process Execution Language for Web Services)* [Andrews et al. 2003], which is a fusion of WSFL and XLang, is the most mature version and an implementation is already provided by IBM and Microsoft. This integrates the notion of web services transactional specifications to manage the composition and to perform compensatory transactions (a kind of undo operation) when necessary. In web services technology, a transactional service must propose compensatory transactions, which are the only practical way to maintain consistency in case of blocking or failure situations.

Recovery aspects at service level have been investigated in WSCAL (*Web Service Composition Action Language*) [Tartanoglu *et al.* 2003a, Tartanoglu *et al.* 2003b], a language for service composition based on XML developed to address fault tolerance of web services, in particular by using forward error recovery

strategies. The principle of forward error recovery is clearly more suitable for web services because it does not impose the handling of state management issue to the web service provider. The WSCAL language creates a coordinator seen as a proxy between the client and all web services in the composite application. The main role of the coordinator is to handle exceptions returned by web services, based on an exception tree for all web services belonging to a composite application. When an exception is raised by a service, the proxy is able to check whether this exception as an impact on other services or not, thanks to the exception tree. When necessary, the proxy triggers the compensating transaction or any other recovery action on target services. The implementation of the language is still in progress.

Finally, BPEL4WS has been used to manage the upgrade of individual component web services into composite web services based applications [Gorbenko et al. 2004]. The idea consists of switching the composite web service from using the old release of the component web service to using its newer release, only when confidence in the new version is high enough, so that the composite service dependability will not deteriorate as a result of the switch.

1.1.6 Quality of service requirements

Quality of service issues in the broader sense have been addressed in [Menascé 2002]. In this respect, except for some security and transaction aspects, there is no formalism today to express properly the expected QoS of a web service, i.e., the core parameters (e.g., delays, resources requirements, etc.) that could be exploited by developers and application designers. In a recent work, IBM proposed a language *WSEL* (*Web Services EndPoint Language*) whose aim is to precisely define certain QoS characteristics of a Web Service Endpoint. Its development is still in progress. In addition, huge efforts are spent to define *Service Level Agreements* (SLAs) corresponding to some QoS agreement between client and provider, including expected QoS objective criteria (e.g., delay of service restart in case of failure) and penalties when the provider does not fulfil them.

WS-Agreement [Andrieux et al. 2004] is an XML language that describes a service-level agreement for Grid Computing and that is supported by the GRAAP working group (*Grid Resource Allocation and Agreement Protocol*). Service-level agreements distinguish between negotiation of QoS requirements and monitoring of the provided QoS in operation. Hence, quality of service and other guarantees that depend on actual resource usage cannot simply be advertised as an invariant property of a service and then bound to by a service consumer. Instead, the service consumer must request state-dependent guarantees to the provider, resulting in an agreement on the service and the associated guarantees. Additionally, the guarantees on service quality must be monitored and failure to meet these guarantees must be notified to consumers.

The objective of the *WS-Agreement* specification is to define a language and a protocol for advertising the capabilities of providers and creating agreements based on creational offers, and for monitoring agreement compliance at runtime. Currently, *WS-Agreement* is just a draft, but this is the most promising and advanced work with respect to other research work like *WSOL* [Tosic *et al.* 2005], *WS-QDL* [Yoon et al. 2004], [Yu-jie et al. 2005], [Hasan and Char 2004], etc.

1.2 Recent research work in RESIST

In this section we summarize the work targeting Service Oriented Architectures done by the partners of the network in the relevant period of RESIST, included as contributions in this deliverable. These contributions relate to the topics proposed in the previous sections.

1.2.1 A Fault Tolerance Support Infrastructure for Web Services based Applications

In this paper, researchers from LAAS propose a support infrastructure that enables both clients and providers to add dependability mechanisms to web services used in large-scale applications **[Salatge and Fabre 2006]**. To this aim, it is introduced the notion of so-called *Specific Fault Tolerance Connectors*. The connectors are software components able to capture web service interactions between clients and providers. They implement filtering and error detection techniques (e.g. runtime assertions) together with recovery mechanisms to improve the robustness of web services. The same web service can be used in several service-oriented applications with different dependability constraints and thus taking advantage of different connectors. To implement recovery strategies, connectors can use the natural redundancy of web services. Similar services can also be found to provide an acceptable service instead the original one, a sort of degraded service. As this approach provides separation of concerns, such dependability mechanisms can easily be adapted to the needs. A central contribution of this work is a dedicated language (a DSL, *Domain Specific Language*) that has been developed to build reliable connectors. A platform has been developed (services and tools), used to implement dependable web services based applications and tested with various web services available on the Internet.

1.2.2 Secure and Provable Service Support for Human-Intensive Real-Estate Processes

In this paper, researchers from Newcastle introduce SOAR, a service-oriented architecture for the real-estate industry that embeds trust and security, allows for formal correctness proofs of service interactions, and systematically addresses human interaction capabilities through web-based user access to services **[Mello et al. 2006]**. The paper demonstrates the features of SOAR through a Deal-Maker service that helps buyers and sellers semi-automate the various steps in a real-estate transaction. This service is a composed service, with message-based interactions specified in SSDL, the SOAP service description language. The implemented embedded trust and security solution deals with the usual privacy and authorization issues, but also establishes trust in ownership and other claims of participants. We also demonstrate how formal techniques can prove correctness of the service interaction protocol specified in SSDL. From an implementation perspective, a main new contribution is a protocol engine for SSDL. A proof-of-concept demonstration is accessible for try-out.

1.2.3 Service-oriented Assurance - Comprehensive Security by Explicit Assurances

Flexibility to adapt to changing business needs is a core requirement of today's enterprises. This is addressed by decomposing business processes into services that can be provided by scalable service-oriented architectures. Service-oriented architectures enable requesters to dynamically discover and use sub-services. Today, service selection does not consider security. In this paper, researchers from IBM introduce the concept of Service-Oriented Assurance (SOAS), in which services articulate their offered security assurances as well as assess the security of their sub-services **[Karjoth et al. 2006]**. Products and services with well-specified and verifiable assurances provide guarantees about their security properties. Consequently, SOAS enables discovery of sub-services with the "right" level of security. Applied to business installations, it enables enterprises to perform a well-founded security/price trade-off for the services used in their business processes.

1.2.4 Modelling of Reliable Messaging in Service-Oriented Architectures

Due to the increasing need of highly dependable services in Service-Oriented Architectures, service-level agreements include more and more frequently such traditional aspects as security, safety, availability, reliability, etc. Whenever a service can no longer be provided with the required QoS, the service requester needs to switch dynamically to a new service having adequate service parameters. In the current paper, researchers from the University of Budapest propose a meta-model to capture such parameters required for reliable messaging in services in a semi-formal way **[Gonczy and Varro 2006]**. Furthermore, they incorporate fault-tolerant algorithms into appropriate reconfiguration mechanisms for modelling reliable message delivery by graph transformation rules.

Currently these researchers are working on the formal verification of the correctness of the proposed reconfiguration mechanisms using existing verification tools for graph transformation systems. As the next step in the future, they plan to implement the automatic generation of runtime implementation in existing middleware and to create test cases for reliable messaging.

2 – Mobile Services and their Infrastructures

Introduction

Mobile phones offer perhaps the most well known example of a mobile service - users are free to receive and make phone calls while moving. The mobile phone service is built around a rather complex infrastructure composed of a huge number of base stations, which extend the fixed Telco physical infrastructure[1]. The complexity of such a cellular-based system remains confined in the managed infrastructure and allows the construction of relatively simple mobile devices.

Nowadays, the availability of mature wireless transmission technologies, such as Bluetooth, WiFi, WiMax, as well as of powerful small-sized computational elements (e.g., PDA, smart phone), allows new forms of mobile networks, whose mobility support does not rely on any infrastructure. Examples of such infrastructure-less network architectures include Mobile Ad Hoc Networks (MANET) and Spontaneous Networks (SN). They enlarge the spectrum of wireless networks, dominated by voice-oriented cellular infrastructure-based network architectures, and provide users with new types of services.

The lack of a physical infrastructure is one key element of this new family of networks, and it is motivated by the much shorter lifetimes (e.g., just a couple of minutes as in SN) or a priori objective of the network. For example, a MANET can be set up for a specific (ad hoc) purpose (e.g., first aid operation) and can last for days [Gerla et al. 2005], while a SN appears spontaneously when two or more users can opportunistically exchange information for a couple of minutes. Figure 2.1 helps positioning some of the ReSIST partner's studies concerning mobile networks.

To offer a mobile service in infrastructure-less networks is particularly challenging for two main reasons: firstly the effects of mobility are in general visible at any layer of the network architecture, from the physical layer to the application one. At the physical/link layers mobility produces wireless communication links with time-variant QoS properties (e.g., delay and bandwidth) so that end-to-end application level channels are in general themselves variable in properties, with the extreme case of many short-term intermittent interactions between end points. The second point is that infrastructure-less networks are by definition open, self-organizing adapting systems.

[1] A similar architectural solution, even if it is extremely smaller in size, is used by wireless LAN, whose connectivity service is offered inside a specific area via wireless access points.

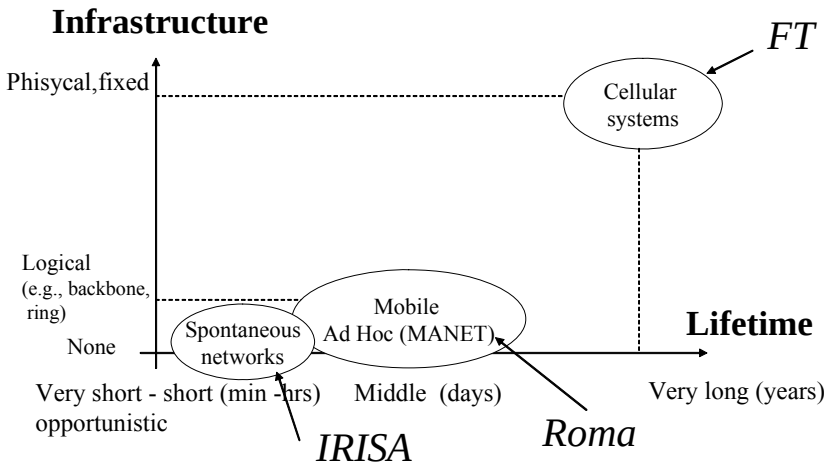

Figure 2.1: Some examples of ReSIST partner studies concerning mobile networks.

2.1 Fundamental research lines

The following sections outline the main lines of researches that ReSIST partners are addressing, and which are related to mobile services and the challenges introduced by ubiquitous systems.

2.1.1 Dependability in cellular systems

Since the early years of mobile networks growth in the late 90's, concerns about the dependability of cellular networks have been raised. Basic studies have adapted to the wireless environment the successful reliability methods used in fixed networks, such as architecture comparison during design phase, thanks to reliability estimates with the use of system decomposition and computation [Varshney and Malloy 2001].

Basic elements, either in the hardware and software, of a mobile infrastructure may fail. The design of reliable networks is thus based on *a priori* computations of their dependability metrics (e.g., availability). Numerical characteristics of network elements (MTBF, MTTR, ...) are then needed: these values are usually extracted from the analysis of *field failure data* [Matz et al. 2002, Lac 2006] which are, ideally, collected automatically, as in a Bluetooth case [Cinque et al. 2005].

2.1.2 Basic communication services for MANETs

Due to the lack of an infrastructure, mobile devices forming a MANET have to cooperate even to provide the basic forms of communications among applications [Martin-Guillerez et al. 2006]. Since the diameter of a MANET is usually much higher than the transmission range of single mobile device, data packet transportation is multi-hop in nature. Moreover, as a consequence of mobility the topology rising at network layer is time-variant [Snow et al. 2000].

Routing protocols represent the main algorithmic tool for implementing a communication service [Beraldi and Baldoni 2002]. Although routing protocols are well understood for fixed data network (e.g. Internet), their applicability to dynamic topologies raised several new issues, e.g. new metrics that take route stability and energy efficiency into account [Jiang et al. 2005, Park et al. 2006]. Things are even more challenging when a Service Level Agreement (SLA) becomes part of the requirements and QoS routing is mandatory [Porcarelli el al. 2003, Zhang and Mouftah 2005, Al-Karaki and Kamal 2004].

As surveyed in [Liu and Kaiser 2005], a large number of routing protocols have been already proposed (e.g., AODV [Perkins el al. 2000], DYMO [Chakeres et al. 2005], and SHARP [Ramasubramanian et al. 2003]). New studies are however still required to full understand the implication of mobility on routing protocols and to investigate new promising directions, like probabilistic algorithms **[**Beraldi el. al. 2006**]** and network coding [Lee at al. 2006].

2.1.3 Middleware service support for mobile applications

This research theme starts from the assumption that some basic forms of communication facility are already available and faces the problem of how to build more powerful abstractions that facilitate the design of mobile applications.

A usual approach is grouping a set of recurrent "low level" services as a *middleware* that application developers can exploit and tune [He 2004, Lac and Ramanathan 2006]. For example, middleware is more powerful to support interactions than simple point-to-point communication, e.g. the publish/subscribe one [Baldoni at al. 2005], or it can provide developers with suitable inter-process coordination primitives [Gadiraju and Kumar 2004], e.g. put/get operations in a generic tuple space **[Arief et al. 2006, Damasceno et al. 2006]**.

2.1.4 Service discovery for MANETs

In order to adapt to different contexts, applications designed for mobile networks need to be flexible and open [Zhu et al. 2005]. In general terms it could be convenient to cast the application context as services that the application could exploit, a central paradigm in Service Oriented Architectures (SOA).

Discovering services available at a given time as well as selecting the most convenient one is a crucial aspect for successfully applying SOA to mobile networks [Cho and Lee 2005]. Mobile devices in fact have inherently few and limited number of resources as compared to fixed ones. It then becomes important to utilize the resources and services available in other devices to accomplish tasks that cannot be done alone. This research line aims at studying how to discover, select and invocate services available at a given time in a network **[Mian et al. 2006]**.

2.1.5 Spontaneous networks

This form of mobile systems is gaining increased attention due to their potential applications. In this case, the interaction between devices is one hop in scope, and limited to the lifetime of the spontaneous communication link, established occasionally between two devices. Spontaneous communications enable to create new services that are synchronized on the meetings of two, or more, physical entities. More generally, these communications enable to synchronize services on spatial conditions, such as the presence, or absence, of an object in an area, or the meeting of several objects.

Applications for this kind of networks often require *atomic* commitment between two mobile devices, i.e., in situations where there are simultaneous state changes on multiple devices. This hard problem, since the available communication time is limited, may be solved by the use of dynamic adaptation protocols [Pauty et al. 2005].

2.2 Recent research work in RESIST

This section summarizes four recent contributions by ReSIST partners, which were included in this deliverable.

2.2.1 On Using the CAMA Framework for Developing Open Mobile Fault Tolerant Agent Systems

Newcastle developed the CAMA (Context-Aware Mobile Agents) framework intended for constructing large-scale mobile applications using the agent paradigm **[Arief et al. 2006]**. CAMA provides a powerful set of abstractions, a supporting middleware and an adaptation layer allowing developers to address the main characteristics of the mobile applications: openness, asynchronous and anonymous communication, fault tolerance, device mobility. It ensures recursive system structuring using location, scope, agent and role abstractions. CAMA supports system fault tolerance through exception handling and structured agent coordination. The applicability of the framework is demonstrated using an ambient lecture scenario - the first part of an ongoing work on a series of ambient campus applications.

2.2.2 Context-Aware Exception Handling in Mobile Agent Systems: The MoCA Case

Handling erroneous conditions in context-aware mobile agent systems is challenging due to their intrinsic characteristics: openness, lack of structuring, mobility, asynchrony and increased unpredictability. Even though several context-aware middleware systems support now the development of mobile agent-based applications, they rarely provide explicit and adequate features for context-aware exception handling.

This paper reports the experience of Newcastle in implementing error handling strategies in some prototype context-aware collaborative applications built with the MoCA (Mobile Collaboration Architecture) system **[Domasceno 2006 et al.]**. MoCA is a publish-subscribe middleware supporting the development of collaborative mobile applications by incorporating explicit services to empower software agents with context-awareness. We propose a novel context-aware exception handling mechanism and discuss some lessons learned during its integration in the MoCA infrastructure.

2.2.3 A Survey on Communication Paradigms for Wireless Mobile Appliances

During the design of wireless services, one must have in mind problems linked to the mobility of the devices. Indeed, when a device moves, it can lose the network, acquire it, or the performances of the network can change. To handle those issues, wireless services must be designed with the knowledge of what can happen in the network. This paper by IRISA surveys existing paradigms to deal with mobility in wireless networks **[Martin-Guillerez et al. 2006]**.

2.2.4 Survey of Service Discovery Protocols in Mobile Ad Hoc Networks

Mobile devices are inherently scarce in resources, having to cooperate among them for performing tasks that cannot be done alone. This cooperation is in the form of services that are offered by other devices in the

network. To get the benefits from the services offered by other devices, these services have to be discovered. Service Discovery Protocols (SDPs) used for this purpose constitute an important area of research in mobile computing and ubiquitous computing.

In this report produced by Roma, twelve SDPs for multihop mobile ad hoc networks are analyzed with respect to six criteria: service discovery architectures, management of service information, search methods, service selection, methods for supporting mobility and service description techniques **[Mian et al. 2006]**. Among these, the most important aspect is the service discovery architecture as it affects other aspects of the service discovery.

We have categorized the service discovery architectures in four groups namely directory-based with overlay support architecture, directory-based without overlay support architecture, directory-less with overlay support architecture and directory-less without overlay support architecture. The management of service information and search methods mainly depends on the type of service discovery architecture.

It was found that mobility support and service selection methods, as well as service descriptions, are independent of the SDP architecture. Mostly the services are described using XML or extensions. Open issues are discussed at the end of the report.

3 – Building Resilient Architectures with Off-the-shelf Components

Introduction

The societal impact of the [un]dependability of off-the-shelf (OTS) information and telecommunication components can hardly be overstated. As the "information society" takes shape, people increasingly depend on the proper functioning of a loose, open computing/communication infrastructure whose building blocks (e.g., proprietary or open-source operating systems and web servers) have either established records of poor dependability, or little evidence of good development practices and acceptable dependability. There has now been for several years a trend towards increasing reliance on OTS components: both from developing custom-built components for each new system towards using existing components, and from using components developed for niche markets with high dependability requirements to buying alternatives that offer lower costs thanks to a larger market. These trends have been accompanied by increasing concerns, perhaps mostly about complex OTS software, with design faults leading to more frequent failures and security problems, but also about OTS hardware for the mass market, containing design faults and also becoming increasingly susceptible to transient faults. In embedded computing, increased reliance on OTS components has already created serious challenges for designers. Apart from headline-making events like the disabling of a U.S. Navy warship by a Windows NT crash [Slabodkin 1998], industries dealing with hazardous processes face the inevitability of replacing, for instance, safety-qualified hardware sensors, now unavailable, with ubiquitous "smart" sensors containing software, that offer many advantages except comparable evidence of dependability.

Using OTS components is commonly believed to reduce at least the initial cost of deploying complex IT systems. But their actual advantages in terms of Total Cost of Ownership (TCO) are uncertain, and inadequate dependability and security contribute heavily to this cost. For instance, a recent analysis [Patterson et al. 2002] suggests that out of TCO values for OTS systems that vary between 3.6 and 18.5 times the purchase cost of systems, "a third to half of TCO is recovering from or preparing against failures". To such visible costs, one must add the cost of failures that are never detected (e.g., data corruption that causes sub-optimal business decisions and other waste); and of catastrophic failures that are too rare to be part of such surveys.

In many of the applications that depend on commercial OTS (COTS) components, the risk from design faults has not yet been addressed adequately. While awareness of these costs and risks slowly grows among users, large vendors of OTS components are slow in improving, due to contrasting market pressures and the sheer difficulty of improving the huge base of installed systems. The supply of many OTS components will be

driven by the dependability requirements that satisfy the majority of their mass markets, but are insufficient for specific business and government sectors of the Information Society; this may well remain true in the long run. This view is supported by the recent history of the industries of safety-critical computer applications. As special product lines of high-dependability components became extinct due to competitive pressures these industries have increasingly had to adapt to using general-purpose tools and OTS components, often with insufficient or unknown dependability. The necessary solution is to use fault tolerance against the failures of these components. Side by side with industrial applications of known schemes, the last decade has thus seen a steady growth of the research on the application of fault tolerance specifically to systems built with OTS components.

The fault-tolerant architectures that can be used to preserve system dependability in the presence of (demonstrated or suspected) insufficient dependability of components vary along several dimensions, which it is useful to recall in order to characterise the different lines of currently active research.

First, regarding the form of error detection, confinement and recovery, both architectures using additional components dedicated to monitoring and recovery from failures of OTS components, and ones based on modular (diverse) redundancy appear promising. The former have always been preferred for applications where the cost of failure did not justify the high cost of developing multiple versions of a component. For OTS components, it often takes the form of *wrapping*, in which a custom-made component filters communications between the OTS component and the rest of the system. But modular redundancy with diversity (i.e., fault-tolerant architectures using diverse, functionally equivalent components) becomes an affordable competitor, since for many functions of OTS components (from chips to complete software packages and hardware-plus-software systems), the market offers alternative, diverse OTS products. Diversification at the level of whole software packages or servers has also been widely advocated for protection of large-scale infrastructures, given the current situation of widespread vulnerabilities, whereby an attacker, having once discovered a single software bug that opens a security vulnerability, can exploit this knowledge at minimal cost to attack myriads of hosts.

Fault-tolerant architectural solutions also differ in the level (in the decomposition or functional hierarchies in a system) at which fault tolerance is applied. Some research focuses on application-level, end-to-end fault tolerance, to deal with the well-known problems of mass-marketed operating systems and applications. Generic, application-level fault tolerance (e.g. multiple-version software) will to some extent also protect against failures and vulnerabilities in the lower level (e.g., operating systems, processors). However, other research also considers means that are specialised at lower level of granularity (e.g., wrapping is applied at all levels from complete applications to individual units in libraries) or in the software-hardware hierarchy (e.g. specialised to tolerate processor failures).

A related area of interest concerns developing essential building blocks that make fault-tolerant architectures easier to build out of undependable, mass-market OTS components, by guaranteeing properties of low-level mechanisms (e.g., communication or voting). In an attractive scenario, these building blocks would lead to economically viable OTS product lines of standardised, comparatively simple, highly dependable products, possibly shared among various high-dependability applications. Notable example include the Time-Triggered Architecture (TTA) [Kopetz et al. 2003], increasingly adopted in the automotive industry, safety-critical railway applications and avionics, and the proposal by Avizienis [Avizienis 2000], described further down.

The rest of this chapter outlines important current lines of research on achieving system resilience with OTS components, and then, in more detail, some more recent contributions **[Avizienis 2006]**, **[Becker et al. 2006]** and **[Gashi et al. 2006b]**, included in appendices to this deliverable.

3.1 Lines of research on resilience with OTS components

3.1.1 Identifying vulnerabilities of OTS software, and wrapping against them.

Groups at LAAS, led by J.C. Fabre, J. Arlat and K. Kanoun and at CMU, led by P. Koopman and centred on the BALLISTA project, have worked on two related areas evaluating via fault injection the vulnerabilities of COTS items (POSIX-compliant operating systems, the Chorus microkernel and the CORBA notification service), and specifying wrappers to "cover" such vulnerabilities [Albinet et al. 2004, Arlat et al. 2002, DeVale et al. 2002, Kropp et al. 1998, Marsden et al. 2002, Pan et al. 2001].

The related HEALERS project at AT&T (C. Fetzer's group) aimed to protect library components by automatically generated wrappers (C macros) that intercept calls and check the validity of call parameters and results,[Fetzer et al. 2003].

M. Swift's group at Univ. of Washington developed Nooks, a subsystem that wraps the Linux kernel and detects improper system calls and predefined exceptions [Swift et al. 2004].

In these early, influential studies the emphasis was on *ad hoc* development of wrappers rather than on defining explicit general goals and assessment criteria. For instance, some of this literature does not acknowledge the potential for the wrappers themselves failing, and thus the need to assess at which point increasing the amount of scrutiny performed by wrappers on communications between components becomes pointless or even counterproductive.

With respect to this last deficiency, the recently completed U.K. project DOTS (a collaboration between the ReSIST members Newcastle and City), produced advances in providing a framework for dealing rigorously with the deficiencies of COTS software. A methodical approach was developed in which *protective wrapping* was seen as a way of structuring fault tolerance with OTS components [van der Meulen et al. 2005], to address explicitly the mismatch between what is required from OTS items in a specific system context and what is known about the available candidate OTS items. The approach was demonstrated on a set of case studies (http://www.csr.ncl.ac.uk/dots/bibliog.html).

While the early studies mentioned above are mainly concerned with wrapping at the level of the operating systems, wrapping at other levels is also used, e.g. at the level of application software, as discussed in the next sub-section.

Other approaches dealing with vulnerabilities detected at various levels have also been used. The SWIFT technique, [Reis et al. 2005], is a recent extension to the long standing line of research about programmer transparent software solutions for dealing with transient hardware faults. It is a compiler-based software transformation, effective in detecting transient hardware faults, and recently extended, [Chang et al. 2006] to recovery from the detected failures. Software solutions like this offer the users of the OTS hardware, e.g. CPU and memory, a means of reducing the negative effects of transient faults beyond the levels provided by the OTS hardware.

The 'Immune System Paradigm' proposed recently by Avizienis [Avizienis 2000, **Avizienis 2006**] is an example of compensating in the system architecture for the lack in modern microprocessors of adequate support for fault tolerance.

3.1.2 Recent work on diversity in replication-based fault-tolerant systems

Diverse redundancy has played a major part in the effort to meet high resilience requirements using existing (including legacy) OTS components, especially when custom-built solutions are either impracticable (due to interconnecting previously existing legacy systems, i.e., systems of systems) or not economically viable due to the limited market needs.

Accepting that fault tolerance with OTS components requires diversity, several groups looked at various aspects of using diverse redundancy:

- B. Liskov's group at MIT developed the BASE approach [Castro et al. 2003], extending the "state machine" approach to fault tolerance to allow diverse replicas of a component. A "conformance wrapper" guarantees that the states of the diverse replicas remain consistent with an *abstract common state*, translates between representations of these states, and enables states to be saved and restored. This approach must cope with both faults and permitted behaviour variations between the diverse components; it aims at tolerating Byzantine faults. A prototype demonstrator was developed at MIT for an NFS file system.

- In the abovementioned DOTS Project the City team, undertook an empirical study with complex OTS products, such as several popular database servers to assess the viability of design diversity with these products and to evaluate, via measurement, the potential benefits in terms of both dependability [Gashi et al. 2004] and performance [Stankovic et al. 2006] gains from deploying diversity. The results are summarised further down in this section.

- The 'Immune System Paradigm' by Avizienis**,** mentioned above, provides support for both identical and diverse multichannel computation.

Further insight into design decisions about which fault-tolerant architectures are appropriate came from studies targeted at measuring the actual prevalence of various failure modes in OTS software with the following important contributions:

- At the Univ. of Michigan, Chen and colleagues, used several open-source products to study the appropriateness of general purpose recovery schemes [Chandra and Chen 2000]. They recorded empirical evidence of serious limitation of these schemes and evidence that a significant proportion of reported faults for the products studied lead to non-crash failures, which reinforces the need for diversity;

- At the University of Urbana Champaign, Ravi Iyer and colleagues, recorded empirical evidence of a strong indication of error dependency or error propagation across a network of NT servers [Xu et al. 1999];

- The City team [Gashi et al. 2004, **Gashi et al. 2006b]** provided direct empirical evidence of fault-diversity with OTS database servers.

3.1.3 Diversity for security

This topic is covered in more detail in the chapter on Intrusion Tolerance. The text here is limited to aspects related to the use of diversity with OTS components to improve security.

The security research community directly embraces the notion of protective wrapping, and has also developed a considerable interest in fault tolerance via diversity to compensate for the (security) deficiencies of OTS components. Three important research strands have been:

- the DARPA sponsored OASIS project in the USA (which developed prototype architectures, e.g. [Fraser et al. 2003], for web sites made attack-resistant via multiple diverse copies of a web server),
- the European MAFTIA project, coordinated by Newcastle and including other ReSIST members (University of Lisboa, IBM Zurich, LAAS), which delivered a reference architecture, a rationalised terminology framework, and supporting mechanisms,
- the DIT project, in which LAAS-CNRS was involved (associated to SRI International). The project developed a prototype architecture and implementation of an adaptive intrusion tolerant web server using diversity - http://www.csl.sri.com/projects/dit.

While significant advances have been made in design and verification of fault tolerance for improved security, progress in the area of quantitative evaluation remains limited [Littlewood et al. 2004]. The difficulties and advance in this area are outlined in the Evaluation part of this deliverable.

3.1.4 Adaptive Fault Tolerance

An important aspect of achieving resilience of computer-based systems, including those developed with OTS components, is managing the evolution of the system configuration or environment during the systems' lifetime. A special case of evolution is the change of the deployed fault tolerance mechanisms. Traditionally, such changes would be implemented statically, i.e., changing the system configuration off-line. Changing the deployed mechanism at runtime, however, has also been explored. The main technical problems with such an approach were outlined and discussed in [Kim et al. 1990]:

- the difficulty to adapt the mechanisms by means of architectural and/or algorithmic solutions;
- the adaptation efficiency, i.e., the ability to monitor the operational conditions and to react to configuration/environment changes.

Projects such as ROAFTS [Kim et al. 1998], MEAD [Narasimhan et al. 2005] and AQuA [Sabnis et al. 1999] propose solutions based on middleware, which allows for transparent switching from one mechanism variant to another at the expense of some performance penalty, i.e., by temporarily freezing the service delivered to the user. Chameleon [Kalbarczyk et al. 1999] follows the same approach. The adaptation executed by a *Fault Tolerance Manager* responsible for collecting the user requirements and other pertinent information and then deciding which of the available fault tolerance mechanism will be used. In such systems, adaptation is often driven by thresholds, e.g., as in MEAD [Dumitras et al. 2005] and ROAFTS. In AQuA, instead, the adaptation is driven by the QoS criteria defined by the clients of the services (e.g., in terms of crash and/or value failures). For instance, the requested availability can be obtained by increasing the number of replicas. Another interesting study [Goldberg et al. 1995] advocates an 'Adaptive Fault Resistant System'; it can be seen as a survey of some possible approaches to address the problem of

adaptation (e.g., adaptive recovery blocks) and open issues (e.g., reflection as in FRIENDS [Fabre et al. 1998, Taiani et al. 2005]).

In ReSIST, work carried out at LAAS relies on the notion of *multi-level reflective architectures* [Taiani et al. 2005] to perform the on-line adaptation. The objective is to limit the impact of the modifications on the service delivered to the user, i.e., without freezing the system but by introducing degraded modes of operation at the non-functional level. This work tackles both the architectural and algorithmic issues to simplify the design and the implementation of on-line adaptive mechanisms.

3.1.5 Infrastructure management

An important aspect of achieving resilience of systems built with OTS components is the so called infrastructure management, a collective label for a multitude of administration tasks and processes and the tools enabling them. Adopting standards of infrastructure management promotes interoperability and best practices and thus reduces the likelihood of misconfiguration of complex systems of OTS components and as a result – poor system performance.

The most widespread infrastructure management standards today are Simple Network Management Protocol (SNMP), Web Based Enterprise management (WBEM) and Java Management Extensions (JMX), which are described briefly below:

- SNMP [IETF 1991] is widely spread both in terms of usage and industry support. SNMP is best suited for the management of networks and network elements. It is also adapted for computational platform management, but the lack of real object-oriented information representation and security issues makes it inferior compared with the other two solutions.

- WBEM [DMTF 2004] was developed by the Distributed Management Task Force (DMTF), a subsidiary of the Object Management Group (OMG) [DMTF 2004]. WBEM is a truly object oriented and model based management standard with a UML-compatible information model – the Common Information Model (CIM). However, although the standards comprising WBEM are available for years now and being supported by the major software vendors its industrial penetration has been limited except for Microsoft Windows NT operating systems. The Windows Management Instrumentation (WMI) is partly a CIM compliant implementation.

- JMX [Sun 2004] is an extension of the Java runtime platform with management and manageability capabilities. While .NET utilises WMI, for Java a standard defines the Java Management Extensions framework (JMX). The standard is heavily used by the major Java-based application servers, e.g., WebSphere, Apache Tomcat etc.

3.2 Recent Research Work in ReSIST

3.2.1 An Immune System Paradigm for the Assurance of Dependability of Collaborative Self-Organizing Systems

This contribution by the VMU team addresses an important problem of enhancing the limited support for fault tolerance built-in the modern microprocessors and other hardware OTS components.

Most currently available microprocessors and other hardware OTS components have very limited fault tolerance (i.e., error detection and recovery) functions. They also do not possess built-in support for redundant multi-channel (duplex, triplex, etc.) computation either with identical, or with diverse hardware and software in the channels. Recently Avizienis has proposed [Avizienis 2000] a network of four types of Application-Specific Integrated Circuits (ASIC) components called the *fault tolerance infrastructure* (FTI) that can be used to embed OTS hardware residing in one package (board, blade, etc.) and provide it with a means to receive error signals from and to issue recovery commands to the OTS components. Furthermore the FTI provides support for both identical and diverse multichannel computation. The FTI employs no software and is fault-tolerant itself. The design principle of the FTI is called *the immune system paradigm* because the FTI can be considered to be analogous to the immune system of the human body "hardware".

The most recent work presented here **[Avizienis, 2006]** applies the FTI concept to the protection of collaborative self-organizing systems composed of relatively simple autonomous agents that act without central control. Because of the changing structure of such systems the application of consensus algorithms for fault tolerance becomes impractical, while the FTI provides fault tolerance individually for every agent, and consensus algorithms are not needed to protect the entire self-organizing system.

3.2.2 Towards an Engineering Approach to Component Adaptation

This report by the Newcastle team addresses an important problem in building dependable systems by integrating ready-made components: how to deal with various mismatches between components **[Becker et al. 2006]**. These mismatches are unavoidable because the components are not built directly for the context in which they are used and because developers of various components typically make a number of assumptions about the context which are not consistent or even conflicting. The well-known solution to these problems is introducing adaptors mediating component interactions. Component adaptation needs to be taken into account when developing trustworthy systems, where the properties of component assemblies have to be reliably obtained from the properties of its constituent components. The adaptor development is still an ad-hoc activity, so a more systematic approach to component adaptation is required when building trustworthy systems. In this paper, the authors show how various design and architectural patterns can be used to achieve component adaptation and thus serve as the basis for such an approach. The paper proposes an adaptation model, which is built upon a classification of component mismatches and identifies a number of specific patterns to be used for eliminating them. It concludes by outlining an engineering approach to component adaptation that relies on the use of patterns and provides an additional support for the development of trustworthy component-based systems.

3.3.3 Fault tolerance via diversity for off-the-shelf products: a study with SQL database servers

This report by the City team is a recent update of previous work on fault diversity [Gashi et al. 2004], which presented empirical evidence that design diversity could offer significant dependability gains for OTS relational database management systems (SQL servers) **[Gashi et al. 2006b]**. The report presents results from a second study with more recent faults reported for two open source SQL servers, PostgreSQL and Firebird, the two most advanced and widely used open-source SQL-servers, which have been reproduced on SQL servers from other vendors. The results observed are consistent with the results reported earlier in [Gashi et al. 2004]:

- very few faults cause simultaneous failure in more than one server.
- the failure detection rate in this study is 100% with only 2 diverse server being used.

- The proportion of crash failures is <50%, consistent with the first study and contrary to the common belief that crash failures are the main concern. Such a high proportion of non-crash failures sheds a serious doubt as to how effective protection is provided by the known database replication solution, developed to tolerate crash failures only.

Ways of diagnosing the failed server were also studied, in the cases this is not evident (e.g., non-crash failures). A variant of data diversity [Ammann et al. 1988] was found to be a promising way of building an efficient rule-based diagnosing system, which only requires a handful of rules to diagnose successfully the failed servers for all faults included in the study **[Gashi and Popov 2006]**.

4 – Intrusion Tolerant Architectures

Introduction

Over the last decades, a significant amount of research and technology has been developed in the fields of dependable computing and computer security. This includes dependable distributed computing architectures, methodologies for building reliable communication, and security mechanisms. These are utilized in our everyday life activities, in a wide spectrum of applications, including: network and information infrastructures, web-based commerce and entertainment, banking and payment systems. Classical solutions for dependable computing have often used the tolerance paradigm at their core, where component failures can be masked and tolerated. Most classical work in security, on the other hand, has focused on preventing security faults from occurring in the first place by equipping systems with defense mechanisms that safeguard the systems against attacks. Other security works have tried to identify vulnerabilities in components either by rigorous testing before deployment or by examination of successful attacks after deployment. However, whilst both fields have taken separate paths until recently, the problems to be solved are of a similar nature: systems have to be kept working correctly, despite the occurrence of faults, which can be either of an accidental nature or caused by malicious actions.

Intrusion tolerance is a new approach that has gained momentum during the past decade [Veríssimo 2002, Lala 2003, Veríssimo et al 2003, Deswarte and Powell 2006]. Although traditional security approaches have been effective in handling many attacks, practical experience shows that most systems remain vulnerable, at least to some extent. This is particularly true for distributed systems whose correct functioning can depend on the possibly complex interactions of software running on many nodes. The concept of intrusion tolerance acknowledges the existence of vulnerabilities in the system, and assumes that over the course of time, a subset of these vulnerabilities will be successfully exploited by intruders. Its objective is to cope with a wide set of faults, ranging from random defects and communication failures to malicious, directed attacks, vulnerabilities, and successful intrusions by an attacker. Thus, the focus of intrusion tolerance is on ensuring that systems will remain operational (possibly in a degraded mode) and continue to provide core services despite faults due to intrusions. In other words, instead of trying to prevent intrusions completely, intrusions are allowed and tolerated to some extent, because the system contains mechanisms that prevent an intrusion from generating a security failure (i.e., a violation of the security policy). Traditional security and intrusion tolerance can be combined to provide an effective "defense-in-depth" strategy for achieving dependability in the face of attacks, failures, or accidents.

In the rest of this section, we discuss the main strategies and mechanisms for architecting intrusion-tolerant systems, and report on recent advances by ReSIST partners on distributed intrusion-tolerant system architectures.

4.1 Intrusion-tolerance in computing systems

The idea that intrusions might be considered as a class of tolerable faults finds its roots in early work on dependable computing concepts [Laprie 1985], where intrusions were referred to as deliberate interaction faults. The term "intrusion tolerance" appeared for the first time in a paper where a scheme for fragmentation-redundancy-scattering was proposed [Fraga and Powell 1985]. Later, this scheme was used in the DELTA-4 project to develop an intrusion-tolerant distributed server composed by a set of insecure sites [Deswarte et al. 1991]. In the following years, a number of intrusion-tolerant protocols and systems emerged, aiming at a few classes of applications.

An *intrusion tolerant* system is one that is able to continue providing a correct service, despite the presence of malicious faults, i.e., deliberate attacks on the security of the system by both insiders and outsiders. Such faults are perpetrated by attackers who make unauthorised attempts to access, modify, or destroy information in a system, and/or to render the system unreliable or unusable. Attacks are facilitated by vulnerabilities and a successful attack results in an intrusion upon the system.

In general, an intrusion can result whenever an attacker is successful in exploiting a vulnerability with respect to any mechanism of a system. If that intrusion is not tolerated, then this can lead to a failure of the mechanism, which could in turn introduce a vulnerability in other parts of the system that depend on the mechanism, allowing the original attack that caused the intrusion to propagate further into the system. An intrusion-tolerant system must be able to continue to deliver a correct service, despite the presence of intrusions, and thus, a "defence in depth" strategy is needed to avoid depending on any particular component of the system that could become a single point of failure.

In the following sections, we discuss intrusion-tolerance mechanisms that have an impact on the architecture of the system in relation to three domains, namely networks and communication, software and programs, and computer hardware.

4.1.1 Intrusion-tolerant communication

This section is concerned with techniques that ensure intrusion-tolerant communication. The relevant mechanisms essentially realize the abstractions of secure reliable channels and secure envelopes, and can be coupled with classic fault-tolerant communication techniques. A secure channel provides the abstraction of a private communication link between two endpoints, which lasts a certain amount of time. An implementation of secure channels may use a combination of physical and virtual encryption and data authentication. Secure channels provide per-session security and normally use symmetric cryptosystems like block ciphers for bulk data encryption and message-authentication codes for bulk data authentication. Public-key encryption and signatures are used during an initial handshake phase for establishing a session key. Secure envelopes are used mainly for sporadic transmissions, such as email. They provide per-message security and use a combination of symmetric and asymmetric cryptosystems for protecting a message. Many techniques are known for implementing fault-tolerant communication protocols, and their choice is related to the nature of the communication and to the kind of failures that are expected. Reliable transmission protocols and secure channels can be seen on a continuous spectrum of intrusion-tolerant communication protocols, which address faults ranging from benign crashes and message omissions to arbitrary and adversarial behaviour of a network.

Many approaches to building intrusion-tolerant systems are based on replication, and thus intrusion-tolerant protocols that support communication amongst groups of replicated processes have been a particular focus of

research. State machine replication involves distributing a service through a number of nodes, where each node runs a replica of the server [Schneider 1990]. This kind of solution ensures the availability and integrity of the service despite a number of intrusions in a subset of the replicas. Managing service or state machine replication in the presence of faults requires that the non-faulty replicas be enabled to determine an identical order on client requests [Schneider and Toueg 1993]. Assumptions about synchrony can be exploited by an attacker (e.g., by attacking the failure detectors), but using Byzantine-agreement protocols to build intrusion-tolerant systems over an asynchronous network requires finding a way of circumventing the famous FLP impossibility result [Fischer et al. 1985], which states that the ordering requirement cannot be met deterministically if the network is asynchronous and if replicas fail even merely by crashing, i.e. stopping to function in a quiescent manner.

For example, Castro and Liskov propose a practical approach to Byzantine fault tolerance that sacrifices liveness for safety [Castro and Liskov 1999], whereas SINTRA developed by IBM uses a non-deterministic asynchronous atomic-broadcast protocol that is able to maintain safety and liveness at the same time. Inayat and Ezhilchelvan from Newcastle address the ordering requirement by constructing an abstract process with signal-on-fail semantics [Inayat and Ezhilchelvan 2006]. In contrast, COCA [Zhou et al. 2002] is an intrusion-tolerant certification authority built using Byzantine quorums [Malkhi and Reiter 1998], a weaker form of agreement.

4.1.2 Intrusion-tolerant software

Fault tolerance methods implemented in software were primarily developed in order to tolerate hardware faults in the underlying execution platform. It has long been known that software-based fault tolerance by replication is extremely effective at handling transient and intermittent software faults [Veríssimo and Rodrigues 2001], but in order to tolerate software design faults, a systematic approach must involve design diversity. Furthermore, Byzantine protocols allow for arbitrary failures, but can only tolerate a certain number of simultaneous failures. By using diversity, it is possible to reduce the likelihood of an attacker being able to exploit a common vulnerability. Thus, software-based mechanisms are also useful in achieving intrusion tolerance.

For example, in the case of design or configuration faults, simple replication would apparently provide little help: errors would systematically occur in all replicas. This is true from a vulnerability viewpoint since such errors are bound to exist in all replicas. However, the common-mode syndrome for intrusion tolerance concerns intrusions, or attack-vulnerability pairs, rather than vulnerabilities alone. This gives the architect some chances. Consider the problem of common-mode vulnerabilities, and of common-mode attacks, i.e., attacks that can be cloned and directed automatically and simultaneously to all (identical) replicas. Design diversity can be applied, for example, by using different operating systems, both to reduce the probability of common-mode vulnerabilities (the classic way), and to reduce the probability of common-mode attacks (by obliging the attacker to master attacks to more than one platform) [Canetti et al. 1997]. Automated methods for adding diversity at compile-time or even at load-time have recently been developed as well [Forrest et al. 1997]. These methods reduce the occurrence of common-mode intrusions, as desired.

However, unless the system can adapt its behaviour in response to previously unknown attacks, a determined attacker will eventually be able to bring the whole system down by exhausting the amount of diversity available. Thus, diversity must be coupled with adaptive learning strategies. The ITUA project [Cukier et al. 2001] investigated the use of unpredictability and adaptation to increase intrusion tolerance, and there is a

DARPA-sponsored programme on Self-Regenerative Systems that is calling for research into areas such as biologically-inspired diversity and cognitive immunity and self-healing.

4.1.3 Hardware-based intrusion-tolerance

Software-based and hardware-based design frameworks for fault tolerance are not incompatible. In fact, a combination of hardware and software based approaches may hold the key to building resilient yet high-performing systems [Saggese et al. 2004]. In a modular and distributed systems context, hardware fault tolerance today should rather be seen as a means of constructing *fail-controlled* components, in other words, components that are prevented from producing certain classes of failures. This contributes to establishing improved levels of trustworthiness, and the corresponding improved trust can be used to achieve more efficient fault-tolerant systems.

Physical enclosure of a processor in tamper-resistant or tamper-proof hardware is an established design option for security systems. For example, many enterprise and banking security systems rely on hardware security modules to safeguard cryptographic keys. The secure hardware perimeter guarantees the confidentiality of the secret keys despite successful (logical or physical) penetration of the host computer. The recently introduced trusted platform module chips found in commodity PCs today play a similar role. Their mission is broader than safeguarding communication keys, however, and addresses the integrity and trustworthiness of the operating system of the host computer.

Distributed algorithms that tolerate arbitrary faults are expensive in both resources and time. For efficiency reasons, the use of hardware components with enforced controlled failure modes is often advisable, as a means for providing an infrastructure where protocols resilient to more benign failures can be used, without that implying a degradation in the resilience of the system to arbitrary faults [Powell et al. 1988]. The performance overhead of computationally intensive cryptographic operations can be mitigated by implementing them in hardware [Saggese et al. 2004].

4.2 Intrusion-tolerance mechanisms

After reviewing intrusion-tolerance mechanisms in a computing system according to their application domain, we review, in this section, four mechanisms that can be applied to multiple components of a computer system: auditing, intrusion detection, error processing, and fault forecasting.

4.2.1 Auditing

Logging system actions and events is a good management procedure, and is routinely done in many operating systems. For technical as well as for accountability reasons, it is very important to be able to trace back the events and actions associated with a given time interval, subject, object, service, or resource. Furthermore, it is crucial that all activity be audited, instead of just a few resources. Finally, the granularity with which auditing is done should be related to the granularity of possible attacks on the system. Since logs may be tampered with by intruders in order to delete their own traces, logs should be tamperproof. For these reasons, audit trails are a crucial framework for building secure systems.

4.2.2 Intrusion detection

Intrusion detection is also a classical security technology, which has encompassed all kinds of attempts to detect the presence or the likelihood of an intrusion. Intrusion detection can be performed in real-time or off-line. In consequence, an intrusion detection system is a supervision system that follows and logs system activity, in order to detect and react (preferably in real-time) against attacks (e.g., port scan detection) and intrusions (e.g. through correlation engines).

An aspect deserving mention under an intrusion tolerance viewpoint is the dichotomy between error detection and fault diagnosis, normally concealed in current intrusion detection systems [Powell and Stroud. 2003]. What does it mean and why is it important? Basically, intrusion detection systems are primarily aimed at complementing prevention and only triggering events that must be followed by manual recovery. When automatic recovery (i.e., fault tolerance) of systems is desired, there is the need to clearly separate errors
(i.e., incorrect system states that could lead to violations of the security policy) from faults (i.e., potential causes that lead to an intrusion). Faults (e.g., attacks, vulnerabilities, intrusions) are to be diagnosed, in order that they can be treated (e.g., removed by applying a security patch to the software, eliminated by tracing back the source of an attack and taking retaliation measures). Errors are to be detected, in order that they can be automatically processed in real-time (e.g., system recovery by rebooting from a clean device, elimination of the compromised system).

Intrusion detection as error detection discovers erroneous states in a system computation deriving from malicious action, e.g., modified files or messages. Intrusion detection as fault diagnosis seeks other purposes, and as such, both activities should not be mixed. Regardless of the error processing mechanism (recovery or masking), administration subsystems are of paramount importance for fault diagnosis. This facet of classical intrusion detection fits into fault treatment [Powell and Stroud 2003]. It can serve to give early warning that errors may occur (vulnerability diagnosis, attack forecasting), to assess the degree of success of the intruder in terms of corruption of components and subsystems (intrusion diagnosis), or to find out who/what performed an attack or introduced a vulnerability (attack diagnosis).

4.2.3 Error processing

Typical error processing mechanisms used in fault tolerance can also be employed under an intrusion tolerance perspective, namely: error detection, error recovery, and error masking. Error detection is concerned with detecting the error after an intrusion is activated. It aims at: confining it to avoid propagation, triggering error recovery mechanisms, and triggering fault treatment mechanisms. Examples of typical errors are: forged or inconsistent (Byzantine) messages, modified files or memory variables, or sniffers, worms, viruses, in operation.

Error recovery is concerned with recovering from the error once it is detected. It aims at providing correct service despite the error, and recovering from effects of intrusions. Error recovery includes intrusion response mechanisms or countermeasures such as deletion of virus-infected files, de-activation of certain user accounts, disabling of ports, removal of corrupted components from the system, placement of partially corrupted components in quarantine, and restarting of corrupted components from a safe state. Generally, there are two types of recovery – backward recovery and forward recovery. Examples of backward recovery are: (1) the system goes back to a previous state known to be correct and resumes, and (2) the system having suffered DoS (denial of service) attack, re-executes the affected operation. Forward recovery can also be used: the system proceeds forward to a state that ensures correct provision of service, or the system detects

the intrusion, considers the corrupted operations lost and increases level of security (threshold/quorums increase, key renewal). Combining intrusion detection with automated recovery mechanisms is an important topic that needs further research in the future [Powell and Stroud 2003, Connelly and Chien 2002, Cukier et al. 2001, Knight et al. 2001].

Unfortunately, when dealing with malicious faults and intrusions, error masking is often the only viable method of error processing, because error detection is not reliable enough or can have large latency. Under appropriate failure assumptions, redundancy and error masking can be used systematically in order to provide correct service without a noticeable glitch. As examples: systematic voting of operations, Byzantine agreement and interactive consistency, fragmentation-redundancy-scattering, or sensor correlation (agreement on imprecise values) [Castro and Liskov 2002, Cachin and Poritz 2002, Correia et al. 2006, Deswarte et al. 1991, Cachin and Tessaro 2006]. However, it is still necessary to use diversity techniques, since an attacker might otherwise be able to exploit a common vulnerability and intrude upon a set of replicas with reduced effort.

4.2.4 Fault forecasting

Fault forecasting evaluates the fault occurrence and activation history of the system. For example, attack prediction can be done by analyzing system logs and audits that show increased port scan activity [Panjwani et al. 2005]. Fault forecasting can be probabilistic or non-probabilistic. An example of probabilistic forecasting is determining the probability that the system will satisfy its dependability properties given that certain components have been successfully compromised by the attacker. An example of non-probabilistic forecasting is predicting the set of possible attacks based on the current state of the system with respect to the system attack tree [Stroud et al. 2004]. Forecasting may be necessary to warn about impending attacks, to predict the attacker's next course of action given that he/she has intruded into some parts of the system, and to provide useful information about how to respond to the attacks.

4.3 Examples of intrusion-tolerant systems and architectures

BFT is an efficient state-machine replication scheme that has been used to implement an intrusion tolerant NFS server [Castro and Liskov 2002]. The Rampart system provides tools for building intrusion tolerant distributed services by offering a number of group communication primitives and a membership service [Reiter 1994, Reiter 1995]. A failure detector is utilized to determine which nodes are behaving badly and to remove them from the group. ITUA is an improved implementation of Rampart´s protocols and architecture that was developed in the OASIS program [Ramasamy et al. 2002]. Other intrusion tolerant group communication systems used different architectures and mechanisms to manage the groups and ensure system progress. SINTRA developed by IBM assumes a static number of replicas and utilizes randomization and threshold cryptography in its implementation [Cachin and Poritz 2002]. Another solution that is based on randomization was developed by Lisboa, but in this case, it managed to avoid any costly cryptographic primitives [Correia et al. 2006, Moniz et al. 2006]. A solution mainly devised for small networks is SecureRing because it organizes the nodes in a logical ring [Kihlstrom et al. 2001]. SecureGroup is another system that resorts to randomization techniques [Moser et al. 2000]. Worm-IT was developed by Lisboa to take advantage of an architecture with an wormhole subsystem [Correia et al. 2006]. The wormhole is constructed in order to be secure and to offer a limited number of distributed operations [Veríssimo 2003, Veríssimo 2006]. Another Byzantine-tolerant group communication system, called JazzEnsemble, was recently proposed based on the Ensemble group communication system [Drabkin et al. 2006]. CoBFIT (A Component-Based Framework for Intrusion Tolerance) [Ramasamy et al. 2004] is a platform for building

and testing intrusion-tolerant distributed systems without having to re-implement the common support for each of those systems. Parsimonious asynchronous protocols for state machine replication and protocols for dynamically changing the composition of the replication group have been implemented in CoBFIT [Ramasamy and Cachin 2005, Ramasamy et al. 2005].

Quorum systems also implement a service using a number of replicas distributed through a set of nodes. However, in these systems, the servers are subdivided in several subgroups of a given size (which depends on the implemented scheme), where different data objects can be replicated, possibly with distinct semantics (e.g., mutual exclusion objects, shared variables). Quorums are usually utilized to create intrusion-tolerant data repositories. Recently, a number of proposals for this kind of systems have been presented in the literature [Malkhi and Reiter 2000, Martin and Alvisi 2004].

Fragmentation and scattering schemes divide an object in several parts and store each one in a distinct server (one of the first solutions was proposed by LAAS [Deswarte et al. 1991]). In order to ensure the availability of the object in case of failure, every part is replicated in some servers. The main motivations for employing these schemes, besides improving the dependability, is to increase the confidentiality of the data and potentially to reduce the overall storage size because each part of the objects is only saved in a subset of servers. One way to implement these schemes is through erasure codes [Rabin 1989]. The idea here is to divide a file in *N* fragments is such a way that it is sufficient to get *K* fragments to re-construct the file [Krawczyk 1993, Garay et al. 2000, Alon et al. 2000]. In project PASIS, a system based on this concept was developed [Goodson et al. 2004]. An extension to these erasure-coding protocols that improves the system resilience and prevents actions by faulty clients was recently proposed by IBM [Cachin and Tessaro 2006].

4.4 Recent Research Work in ReSIST

4.4.1 MAFTIA project

MAFTIA (Malicious and Accidental Fault Tolerance for Internet Applications) was a three-year European research project funded as part of the IST programme (IST-1999-11583) that ran from Jan 2000 to Feb 2003 and brought together experts in the fields of dependability, fault tolerance, computer security, distributed systems, intrusion-detection systems, cryptography, and formal methods, including several partners in ReSIST (IBM, LAAS, Lisboa, Newcastle). MAFTIA investigated the 'tolerance paradigm' for security systematically, with the aim of proposing an integrated architecture built on this paradigm, and realising a concrete design that can be used to support the dependability of many applications. The project's major innovation was a comprehensive approach for tolerating both accidental faults and malicious attacks in large-scale distributed systems, including attacks by external hackers and by corrupt insiders. There were three main areas of work:

- the definition of an architecture and conceptual model that provides a framework for ensuring the dependability of distributed applications in the face of a wide class of faults and attacks;
- the design of mechanisms and protocols that provide building blocks for implementing large-scale dependable applications.
- the development of formal assessment techniques that provide rigorous definitions of the basic concepts developed by MAFTIA and enable the verification of selected mechanisms and protocols.

The first area of work aimed to develop a coherent set of concepts for an architecture that could tolerate malicious faults [Powell and Stroud 2003]. Work was done on the definition of a core set of intrusion tolerance concepts, clearly mapped into the classical dependability concepts. Other relevant work included the definition of synchrony and topological models, the establishment of concepts for intrusion detection and the definition of a MAFTIA node architecture. This architecture includes components such as trusted and untrusted hardware, local and distributed trusted components, operating system and runtime environment, software, etc.

Work on the design of intrusion-tolerant mechanisms and protocols included the definition of the *MAFTIA middleware*: architecture and protocols [Veríssimo and Neves 2001, **Veríssimo et al. 2006**]. An asynchronous suite of protocols, including reliable, atomic and causal multicast was defined, providing Byzantine resilience by resorting to efficient solutions based on probabilistic execution [Cachin et al 2000]. Work was also done on protocols based on a timed model that relies on the concept of *wormholes*, enhanced subsystems that provide components with a means to obtain a few simple privileged functions and/or channels to other components, with "good" properties otherwise not guaranteed by the "normal" weak environment [Veríssimo 2003]. For example, the Trusted Timely Computing Base (TTCB) developed in MAFTIA is based on a wormhole providing timely and secure functions on environments that are asynchronous and Byzantine-on-failure. Architectural hybridisation is used to implement the TTCB. In the context of the MAFTIA middleware, an intrusion tolerant transaction service with support for multiparty transactions was also designed and implemented.

Intrusion detection is assumed as a mechanism for intrusion tolerance but also as a service that has to be made intrusion-tolerant. MAFTIA developed an architecture for a large-scale distributed intrusion-tolerant intrusion-detection system [Debar and Wespi 2001]. Problems like handling high rates of false alarms and combining the outputs from several different intrusion detection systems were also explored.

Trusted Third Parties (TTPs) such as certification authorities are important building blocks in today's Internet. MAFTIA designed a generic distributed certification authority that uses threshold cryptography and intrusion tolerant protocols in order to be intrusion-tolerant. Another TTP, the distributed optimistic fair exchange service, was also developed.

MAFTIA defined an *authorization service* based on fine grain protection, i.e., on protection at the level of the object method call [Nicomette and Deswarte 1996]. The authorization service is a distributed TTP which can be used to grant or deny authorization for complex operations combining several method calls. The service relies on a local security kernel.

Finally, with respect to the work on formal assessment techniques, MAFTIA developed a model for reactive cryptographic systems that allows for the formal specification and automatic verification of security properties under a standard cryptographic semantics, and selected components of the MAFTIA middleware were formally verified [Adelsbach and Creese 2003].

4.4.2 DIT project

The DIT (Dependable Intrusion Tolerance) architecture was proposed by SRI International in cooperation with LAAS to build Web servers that continue to provide correct service in the presence of attacks [Valdes et al. 2004]. For this type of application, confidentiality is not essential, but integrity and availability must be ensured, even if the system is under attack from competent attackers. It is thus essential that a successful

attack on one component of the system should not facilitate attacks on other components. The architecture design is thus centered on a diversification approach.

The architecture is composed of a pool of ordinary Web servers, using as much diversification as possible at the hardware level (Sparc, Pentium, PowerPC, etc.), the operating system level (Solaris, Microsoft Windows, Linux, MacOS, etc.) and Web application software level (Apache, IIS, Enterprise Server, Openview Server, etc.). Only the content of the Web pages is identical on each server. There are sufficient application servers at a given redundancy level (see below) to ensure an adequate response time for the nominal request rate. The servers are isolated from the Internet by *proxies*, which are implemented by purpose-built software executed on diversified hardware. Requests from the Internet, filtered by a firewall, are taken into account by one of the proxies acting as a *leader*. The leader distributes the requests to multiple Web servers and checks the corresponding responses before returning them to the request initiator. The back-up proxies monitor the behavior of the leader by observing the firewall/proxy and proxy/server networks. If they detect a failure of the leader, they elect a new leader from among themselves. The proxies also process alarms from intrusion detection sensors placed on the Web servers and on both networks.

Depending on the current level of alert, the leader sends each request to one server (simplex mode), two servers (duplex mode), three servers (triplex mode) or to all available servers. Each server prepares its response, then computes an MD5 cryptographic checksum of this response and sends it to the leader. In simplex mode, the server also sends its response to the leader, which recomputes the checksum and compares it to the one sent by the server. In duplex mode, the leader compares the two checksums from the servers and, if they concur, requests one the responses, which is verified by recomputing the checksum. In triplex or all-available modes, the checksums are subjected to a majority vote, and the response is requested from one of the majority servers.

The alert level is defined as either a function of recent alarms triggered by the intrusion detection mechanisms or other error detection mechanisms (result cross-checking, integrity tests, etc.), or by information sent by external sources (CERTs, other trusted centers, etc.). The redundancy level is raised towards a more severe mode as soon as alarms are received, but is lowered to a less severe mode when failed components have been diagnosed and repaired, and when the alarm rate has decreased. This adaptation of the redundancy level is thus tightly related to the detection, diagnosis, reconfiguration and repair mechanisms. In the case of read-only data servers, such as passive Web servers, repair involves just a simple re-initialization of the server from a back-up (an authenticated copy on read-only storage).

Diversification renders the task of the attacker as difficult as possible: when an attacker sends a Web page request (the only means for him to access the application servers), he does not know towards which servers his request will be forwarded and thus which hardware or software will process it. Even if he were able to design an attack that would be effective on all server types (except maybe for denial-of-service attacks, which are easy to detect), it would be very difficult to cause redundant servers (in duplex mode and above) to reply in exactly the same incorrect way.

4.4.3 Using the Signal-On-Fail Approach to Impose Order in the Streets of Byzantium

Managing service or state machine replication in the presence of faults requires that the non-faulty replicas be enabled to determine an identical order on client requests [Schneider and Toueg 1993]. Work done by Newcastle [Inayat and Ezhilchelvan 2006] addresses this ordering requirement when nodes hosting replicas can fail in a malicious, Byzantine manner and are connected by an asynchronous network, e.g., the Internet, wherein the message transfer delays cannot be bounded with certainty by a known constant. In particular, a

novel approach is proposed, and is evaluated for its effectiveness towards circumventing the well-known FLP impossibility [Fischer et al. 1985].

The approach followed in this project dynamically constructs an abstract process with *signal-on-crash* semantics: it fails only by crash and additionally *fail-signal*s its own imminent crash. Such an abstract process is constructed by pairing up a subset of processes in the system. When failures are signalled, the impossibility result ceases to apply and when they do not involve producing incorrect outputs, a simplified protocol structure, smaller latencies and lower message overhead ensue. This construction of an abstract process however requires an additional assumption, namely that the paired processes cannot fail at the same time. This assumption requires implementation of measures that assure failure-independence and it is argued that such a realisation is practical.

The set of protocols developed here, like BFT [Castro and Liskov 2002], is coordinator-based and deterministic. The paired-up processes, when called upon to act as the co-ordinator, construct an abstract, signal-on-crash process similar to the way in which the component processes of the abstract fail-stop process [Schlichting and Schneider 1983] maintain the signal-on-crash property. They operate in parallel and endorse each other's outputs if the latter are found to be consistent with their own outputs. The endorsed outputs are treated as the outputs of the signal-on-crash process and the endorsement is indicated through digital signatures that are assumed to be non-forgeable. If a process within a pair suspects a failure of its counter-part, it stops all activities related to implementing the signal-on-crash process abstraction and indicates this stopping by outputting a *fail-signal* message. Thus, the signal-on-crash process either outputs verifiably-endorsed messages of correct contents or stops functioning after signalling its stopping. That is, it can only crash and when it does, it *fail-signal*s prior to doing so. It is easy to see that the paired processes operate together as a single non-faulty coordinator (except for the doubly-signed output format), so long as no non-faulty process in the pair observes a failure on its counter-part.

On-going work is concerned with comparing the performance of this suite of optimistic Byzantine fault-tolerant order protocols with that of BFT, a protocol best known for its practicability. The initial results show that the proposed approach works better than BFT in failure-free scenarios.

Conclusions

This part surveys a number of issues related to the architecture and implementation of future resilient ubiquitous systems. These issues were aggregated in four main chapters, each one presenting the current state of knowledge and ongoing investigations by ReSIST partners in a key research area. The first chapter addresses the development of service oriented architectures, which are particularly relevant when dealing with problems of scale, heterogeneity, and evolution of the infrastructures. The chapter on mobile services and their infrastructures covers resilience aspects of mobile architectures. An understanding of these aspects is needed in order to devise solutions for more dynamic and diverse systems. The cost pressures that are felt in most sectors of the economy forces the use of OTS components in most current and future computing systems. Since many of these components are arguably less trustworthy, it is necessary to devise mechanisms to reduce their impact on the overall dependability and security of the systems. These topics are discussed in detail in the chapter on building resilient architectures with OTS components. The chapter on intrusion tolerant architectures explains how a wide set of accidental and malicious faults can be tolerated, ensuring correct service behavior even when attacks are successful. These problems have to be considered in pervasive infrastructures since they provide a fertile ground where malicious actions can be carried out both by hackers and criminals.

The analysis of these areas, however, has shown that there are still some significant obstacles that have to be overcome in order to support scalable ubiquitous systems. There are also extra topics that have to be considered in order to address the challenge of scaling resilience. The identification of this type of gaps is one of the main subjects of the next deliverable, D13. Two concrete examples of these problems are: it is not clear how more dynamic and mobile systems can be made intrusion tolerant, since most existing approaches require a majority of correct participants; it is challenge to make reliable and secure applications out of current web services, since these are based on unreliable and insecure architectures and protocols.

References

[Adelsbach and Creese 2003] A. Adelsbach and S. Creese (editors), “Final Report on Verification and Assessment”, Project MAFTIA IST-1999-11583, Deliverable D22, January 2003.

[Albinet et al. 2004] A. Albinet, J. Arlat and J.-C. Fabre, "Characterisation of the Impact of Faulty Drivers on the Robustness of the Linux Kernel", Proceedings of the International Conference on Dependable Systems and Networks, pp. 867-876, 2004.

[Alon et al. 2000] N. Alon, H. Kaplan, M. Krivelevich, D. Malkhi, and J. Stern, “Scalable Secure Storage when Half the System is Faulty”, In Montanari, U., Rolim, J., and Welzl, R., editors, Proceedings of the 27th International Colloquium on Automata, Languages and Programming, Vol. 1853 of LNCS, pp. 576–587, 2000.

[Alwagait and Ghandeharizadeh 2004] E. Alwagait and S. Ghandeharizadeh, “DeW: A Dependable Web Services Framework”, Proceedings of the 14th International Workshop on Research Issues on Data Engineering: Web Services for E-Commerce and E-Government Applications, March 2004.

[Ammann et al. 1988] P. E. Ammann and J. C. Knight, "Data Diversity: An Approach to Software Fault Tolerance", IEEE Transactions on Computers, Vol. C-37, No. 4, pp.418-425, 1988.

[Arief et al. 2006] B. Arief, A. Iliasov, and A. Romanovsky, "On Using the CAMA Framework for Developing Open Mobile Fault Tolerant Agent Systems", Workshop on Software Engineering for Large-Scale Multi-Agent Systems, pp. 29-35, May 2006.

[Al-Karaki and Kamal 2004] J. N. Al-Karaki and A. E. Kamal, "Stability Helps Quality of Service Routing in Wireless Ad Hoc Networks", Proceedings of the IEEE International Conference on Performance, Computing, and Communications, pp. 329-336, 2004.

[Anderson et al. 2004] S. Anderson, J. Bohren, T. Boubez, M. Chanliau, G. Della-Libera, B. Dixon, P. Garg, E. Gravengaard, M. Gudgin, S. Hada, P. Hallam-Baker, M. Hondo, C. Kaler, H. Lockhart, R. Martherus, H. Maruyama, P. Mishra, A. Nadalin, N. Nagaratnam, A. Nash, R. Philpott, D. Platt, H. Prafullchandra, M. Sahu, J. Shewchuk, D. Simon, D. Srinivas, E. Waingold, D. Waite and R. Zolfonoon, “Web Services Secure Conversation Language (WS-SecureConversation)”, Version 1.1, http://specs.xmlsoap.org/ws/2004/04/sc/ws-secureconversation.pdf, 2004.

[Anderson et al. 2005] S. Anderson, J. Bohren, T. Boubez, M. Chanliau, G. Della-Libera, B. Dixon, P. Garg, M. Gudgin, P. Hallam-Baker, M. Hondo, C. Kaler, H. Lockhart, R. Martherus, H. Maruyama, A. Nadalin, N. Nagaratnam, A. Nash, R. Philpott, D. Platt, H. Prafullchandra, M. Sahu, J. Shewchuk, D. Simon, D. Srinivas,

E. Waingold, D. Waite, D. Walter and R. Zolfonoon, “Web Services Trust Language (WS-Trust)”, ftp://www6.software.ibm.com/software/developer/library/ws-trust.pdf, 2005.

[Andrews et al. 2003] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. T. (Editor), I. Trickovic and S. Weerawarana., “Business Process Execution Language for Web Services, Version 1.1”, 2003.

[Andrieux et al. 2004] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, J. Pruyne, J. Rofrano, S. Tuecke and M. Xu, “Web Services Agreement Specification (WS-Agreement)”, Draft 18 Version 1.1, 2004.

[Arlat et al. 2002] J. Arlat, J.-C. Fabre, M. Rodriguez and F. Salles, “Dependability of COTS Microkernel-based Systems”, IEEE Transactions on computer Systems, Vol. 51, No. 2, pp.138-163, 2002.

[Avizienis 2000] A. Avizienis. "A Fault Tolerance Infrastructure for Dependable Computing with High-Performance COTS Components", Proceedings of the 2000 International Conference on Dependable Systems and Networks, pp. 492-500, June 2000.

[Avizienis 2006] A. Avizienis. "An Immune System Paradigm for the Assurance of Dependability of Collaborative Self-Organizing Systems", Proceedings of the IFIP 19th World Computer Congress, 1st IFIP International Conference on Biologically Inspired Computing, pp. 1-6., 2006.

[Bajaj et al. 2006] S. Bajaj, D. Box, D. Chappell, F. Curbera, G. Daniels, P. Hallam-Baker, M. Hondo, C. Kaler, D. Langworthy, A. Nadalin, N. Nagaratnam, H. Prafullchandra, C. v. Riegen, D. Roth, J. Schlimmer, C. Sharp, J. Shewchuk, A. Vedamuthu, Ü. Yalç nalp and D. Orchard, “Web Services Policy Framework (WSPolicy)”, Version 1.2, http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-polfram/ws-policy-2006-03-01.pdf, 2006.

[Bajaj et al. 2003] S. Bajaj, G. Della-Libera, B. Dixon, M. Dusche, M. Hondo, M. Hur, C. Kaler, H. Lockhart, H. Maruyama, A. Nadalin, N. Nagaratnam, A. Nash, H. Prafullchandra and J. Shewchuk, “Web Services Federation Language (WSFederation)”, Version 1.0, ftp://www6.software.ibm.com/software/developer/library/ws-fed.pdf, 2003.

[Baldoni el al. 2005] R. Baldoni, R. Beraldi, G. Cugola, M. Migliavacca, and L. Querzoni, “Structure-less Content-Based Routing in Mobile Ad Hoc Networks”, Proceedings of the IEEE International Conference on Pervasive Service, 2005.

[Banerji et al. 2002] A. Banerji, C. Bartolini, D. Beringer, V. Chopella, K. Govindarajan, A. Karp, H. Kuno, M. Lemon, G. Pogossiants, S. Sharma and S. Williams, “Web Services Conversation Language (WSCL) 1.0”, W3C Note, http://www.w3.org/TR/wscl10/, 2002.

[Bartel et al. 2002] M. Bartel, J. Boyer, B. Fox, B. LaMacchia and E. Simon, “XML-Signature Syntax and Processing ”, W3C Recommendation, http://www.w3.org/TR/xmldsig-core/, 2002 .

[Becker et al. 2006] S. Becker, A. Brogi, I. Gorton, S. Overhage, A. Romanovsky and M. Tivoli, "Towards an Engineering Approach to Component Adaptation", R. H. Reussner, J. A. Stafford and C. A. Szyperski, editors, Architecting Systems with Trustworthy Components, Vol. 3938 of LNCS, pp. 193-215, 2006.

[Beraldi and Baldoni 2002] R. Beraldi and R. Baldoni, “Unicast Routing Techniques for Mobile Ad Hoc Networks”, The Handbook of Mobile Ad Hoc Networks, Chapter 7, CRC press, December 2002.

[Beraldi et. al. 2006] R. Beraldi, L. Querzoni, R. Baldoni, “A Hint Based Probabilistic Protocol for Unicast Communications in MANETs”, Elsevier Ad Hoc Networks, 2006.

[Beraldi 2006] R. Beraldi, "A Directional Gossip Protocol for Path Discovery in MANETs", Workshop on Wireless Ad hoc and Sensor Networks, July 2006.

[Cabrera et al. 2005a] L. F. Cabrera, G. Copeland, M. Feingold, R. W. Freund, T. Freund, J. Johnson, S. Joyce, C. Kaler, J. Klein, D. Langworthy, M. Little, A. Nadalin, E. Newcomer, D. Orchard, I. Robinson, J. Shewchuk and T. Storey, “Web Services Coordination (WSCoordination)”, Version 1.0, http://www-128.ibm.com/developerworks/library/specification/ws-tx/#atom, 2005a.

[Cabrera et al. 2005b] L. F. Cabrera, G. Copeland, M. Feingold, R. W. Freund, T. Freund, J. Johnson, S. Joyce, C. Kaler, J. Klein, D. Langworthy, M. Little, A. Nadalin, E. Newcomer, D. Orchard, I. Robinson, T. Storey and S. Thatte, “Web Services Atomic Transaction (WSAtomicTransaction)”, Version 1.0, http://www-128.ibm.com/developerworks/library/specification/ws-tx/#atom, 2005b.

[Cabrera et al. 2005c] L. F. Cabrera, G. Copeland, M. Feingold, R. W. Freund, T. Freund, S. Joyce, J. Klein, D. Langworthy, M. Little, F. Leymann, E. Newcomer, D. Orchard, I. Robinson, T. Storey and S. Thatte, “Web Services Business Activity Framework (WS-BusinessActivity)”, Version 1.0, ftp://www6.software.ibm.com/software/developer/library/WS-BusinessActivity.pdf, 2005c.

[Cachin et al 2000] C. Cachin, K. Kursawe, and V. Shoup, “Random Oracles in Constantinople: Practical Asynchronous Byzantine agreement using cryptography”, Proceedings of the 19th ACM Symposium on Principles of Distributed Computing, pp.123-132, 2000.

[Cachin and Poritz 2002] C. Cachin, and J. Poritz, “Secure Intrusion-Tolerant Replication on the Internet”, Proceedings of the International Conference on Dependable Systems and Networks, pp. 167–176, June 2002.

[Cachin and Tessaro 2006] C. Cachin, and S. Tessaro, “Optimal Resilience for Erasure Coded Byzantine Distributed Storage”, Proceedings of the International Conference on Dependable Systems and Networks, pp. 115-124, June 2006.

[Canetti et al. 1997] R. Canetti, R. Gennaro, A. Herzberg, and D. Naor, “Proactive Security: Long-term Protection against Break-ins”, RSA CryptoBytes, No. 3, 1997.

[Castro and Liskov 2002] M. Castro, and B. Liskov, “Practical Byzantine Fault Tolerance and Proactive Recovery”, ACM Transactions on Computer Systems, Vol. 20, No. 4, pp. 398–461, 2002.

[Castro et al. 2003] M. Castro, R. Rodrigues and B. Liskov, "BASE: Using Abstraction to Improve Fault Tolerance", ACM Transactions on Computer Systems, Vol. 21, No. 3, pp.236-269, 2003.

[Ceponkus et al. 2002] A. Ceponkus, S. Dalal, T. Fletcher, P. Furniss, A. Green and B. Pope, “Business Transaction Protocol”, An OASIS Committee Specification, Version 1.0, http://www.oasis-open.org/committees/download.php/1184/2002-06-03.BTP_cttee_spec_1.0.pdf, 2002.

[Chakeres et al. 2005] I. D. Chakeres, E. M. Royer and C.E. Perkins, "Dynamic MANET On-Demand Routing Protocol", IETF Internet Draft, draft-ietf-manet-dymo- 02.txt, June 2005.

[Chandra and Chen 2000] S. Chandra, P. M. Chen "Whither Generic Recovery from Application Faults? A Fault Study using Open-Source Software", Proceedings of the International Conference on Dependable Systems and Networks, June 2000.

[Chang et al. 2006] J. Chang, G. A. Reis and D. I. August. "Atomic Instruction-Level Software-Only Recovery", Proceedings of the International Conference on Dependable Systems and Networks, pp. 83-92, June 2006.

[Cho and Lee 2005] C. Cho and D. Lee, "Survey of Service Discovery Architectures for Mobile Ad Hoc Networks", Computer and Information Sciences and Engineering Department, University of Florida, Gainesville, 2005. http://folk.uio.no/paalee/referencing_publications/ref-sd-cho-cice05.pdf

[Cinque et al. 2005] M. Cinque, F. Cornevilli, D. Cotroneo and S. Russo, "An Automated Distributed Infrastructure for Collecting Bluetooth Field Failure Data", Proceedings of the 8th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, pp. 329-336, May 2005.

[Connelly and Chien 2002] K. Connelly, and A. Chien, “Breaking the Barriers: High Performance Security for High Performance Computing”, Proceedings of the New Security Paradigms Workshop, 2002.

[Cotroneo et al. 2003] D. Cotroneo, C.Di Flora and S. Russo, “Improving Dependability of Service Oriented Architectures for Pervasive Computing”, The Eighth IEEE International Workshop on Object-Oriented Real-Time Dependable Systems, 2003.

[Correia et al. 2006] M. Correia, N. Neves, L. Lung, and P. Veríssimo, “Worm-IT - A Wormhole-based Intrusion-Tolerant Group Communication System”, Journal of Systems & Software, accepted for publication in 2006.

[Correia et al. 2006] M. Correia, N. Neves, and P. Veríssimo, “From Consensus to Atomic Broadcast: Time-Free Byzantine-Resistant Protocols without Signatures”, The Computer Journal, Vol. 49, No. 1, pp. 82-96, January 2006.

[Courtès et al. 2003] L. Courtès, M.-O. Killijian, D. Powell and M. Roy, "Sauvegarde Coopérative Entre Pairs Pour Dispositifs Mobiles", The 2nd French-Speaking Conference on Mobility and Ubiquity Computing (ACM International Conference Proceeding Series, Vol. 120), pp. 97-104, 2005.

[Cukier et al. 2001] M. Cukier, J. Lyons, P. Pandey, H. Ramasamy, W. Sanders, P. Pal, F. Webber, R. Schantz, J. Loyall, R. Watro, M. Atighetchi, and J. Gossett, “Intrusion Tolerance Approaches in ITUA”, Supplement of the 2001 International Conference on Dependable Systems and Networks, June 2001.

[Debar and Wespi 2001] H. Debar, and A. Wespi, “Aggregation and Correlation of Intrusion Detection Alerts”, The 4th Workshop on Recent Advances in Intrusion Detection, Vol. 2212 of LNCS, 2001.

[Defago et al. 2000] X. Defago, A. Schiper and P. Urban, “Totally Ordered Broadcast and Multicast Algorithms: A Comprehensive Survey”, Technical Report DSC/2000/036, Dept. of Communication Systems, EPFL, 2000.

[Deswarte et al. 1991] Y. Deswarte, L. Blain, and J. Fabre, “Intrusion Tolerance in Distributed Computing Systems”, Proceedings of the 1991 IEEE Symposium on Research in Security and Privacy, 1991.

[Deswarte and Powell 2006] Y. Deswarte, and D. Powell, “Internet Security: An Intrusion-Tolerance Approach”, Proceedings of the IEEE, Vol. 94, No. 2, pp. 432-441, February 2006.

[DeVale et al. 2002] J. DeVale and P. Koopman. "Robust Software - No More Excuses", Proceedings of the International Conference on Dependable Systems and Networks, pp. 145-154, June 2002.

[Dialani et al. 2002] V. Dialani, S. Miles, L. Moreau, D. D. Roure and M. Luck, “Transparent Fault Tolerance for Web Services base Architectures”, Proceedings of 8th International Europar Conference, pp. 889-898, 2002.

[DMTF 2004] DMTF, "Web-Based Enterprise Management (WBEM)", 2004.

[Damasceno et al. 2006] K. Damasceno, N. Cacho, A. Garcia, A. Romanovsky, and C. Lucena, "Context-Aware Exception Handling in Mobile Agent Systems: The MoCA Case", Workshop on Software Engineering for Large-Scale Multi-Agent Systems, May 2006.

[Drabkin et al. 2006] V. Drabkin, R. Friedman, and A. Kama, “Practical Byzantine Group Communication”, Proceedings of the 26th International Conference on Distributed Computing Systems, July 2006.

[Dumitras et al. 2005] T.A. Dumitras, D. Srivastava and P. Narasimhan. "Architecting and Implementing Versatile Dependability", R. de Lemos, C. Gacek and A. Romanovsky, editors, Architecting Dependable Systems III, Vol. 3549 of LNCS, pp. 212-231, 2005.

[Fabre et al. 1998] J.-C. Fabre and T. Perennou, "A Metaobject Architecture for Fault-Tolerant Distributed Systems: The FRIENDS Approach", IEEE Transactions on Computers, Vol. 47, No. 1, pp.78-95, 1998.

[Fetzer et al. 2003] C. Fetzer and Z. Xiao, "HEALERS: A Toolkit for Enhancing the Robustness and Security of Existing Applications", Proceedings of the International Conference on Dependable Systems and Networks, pp. 317-322, June 2003.

[Forrest et al. 1997] S. Forrest, A.. Somayaji, and D. Ackley, "Building Diverse Computer Systems", Proceedings of the Sixth Workshop on Hot Topics in Operating Systems, pp. 67-72, 1997.

[Fraga and Powell 1985] J. Fraga, and D. Powell, “A Fault- and Intrusion-Tolerant File System”, Proceedings of the 3rd International Conference on Computer Security, 1985.

[Fraser et al. 2003] T. Fraser, L. Badger and M. Feldman. "Hardening COTS Software with Generic Software Wrappers", J. H. Lala, editor, Foundation of Intrusion Tolerant Systems - Organically Assured and Survivable Information Systems (OASIS), pp. 399-413, 2003.

[Gadiraju and Kumar 2004] S. Gadiraju and V. Kumar, "Recovery in the Mobile Wireless Environment Using Mobile Agents", IEEE Transactions on Mobile Computing, Vol. 3, No. 2, pp. 180-191, April-June 2004.

[Garay et al. 2000] J. Garay, R.. Gennaro, C. Jutla, and T. Rabin, “Secure Distributed Storage and Retrieval”, Theoretical Computer Science, Vol. 243, No. 1-2, pp. 363–389, 2000.

[Gashi and Popov 2006] I. Gashi and P. Popov. "Rephrasing Rules for Off-The-Shelf SQL Database Servers", Proceedings of the 6th European Dependable Computing Conference, October 2006.

[Gashi et al. 2004] I. Gashi, P. Popov and L. Strigini. "Fault Diversity Among Off-the-shelf SQL Database Servers", Proceedings of the International Conference Dependable Systems and Networks, pp. 389-398, June 2004.

[Gashi et al. 2006b] I. Gashi, P. Popov and L. Strigini "Fault Tolerance via Diversity for Off-the-shelf Products: A Study with SQL Database Servers", manuscript, 2006.

[Gerla et al. 2005] M. Gerla, L.-J. Chen, Y.-Z. Lee, B. Zhou, J. Chen, G. Yang, and S. Das, "Dealing with Node Mobility in Ad Hoc Wireless Network", Vol. 3465 of LNCS, 2005.

[Goldberg et al. 1995] J. Goldberg, I. Greenberg, R. Clark, E. D. Jensen, K. Kim and D. M. Wells, "Adaptive Fault-Resistant Systems", SRI-CSL-95-02, SRI International Computer Science Laboratory, 1995.

[Gonczy and Varro 2006] L. Gonczy and D. Varro "Modeling of Reliable Messaging in Service Oriented Architectures", Andrea Polini, editor, Proceedings of the International Workshop on Web Services Modeling and Testing, pp. 35-49, 2006.

[Goodson et al. 2004] G. Goodson, J. Wylie, G. Ganger, and M. Reiter, "Efficient Byzantine-Tolerant Erasure-Coded Storage", Proceedings of the International Conference on Dependable Systems and Networks, June 2004.

[Gorbenko et al. 2004] A. Gorbenko, V. Kharchenko, P. Popov, A. Romanovsky and A. Boyarchuk, "Development of Dependable Web Services out of Undependable Web Components", School of Computing Science, University of Newcastle upon Tyne, Technical Report Series CS-TR-863, 2004.

[Hasan and Char 2004] O. Hasan and B. Char, "A deployment-ready solution for adding quality-of-service features to web services", The 2004 International Research Conference on Innovations in Information Technology, 2004.

[He et al. 2004] J. He, M. A. Hiltunen and R. D. Schlichting, "Customizing Dependability Attributes for Mobile Service Platforms", Proceedings of the International Conference on Dependable Systems and Networks, pp. 617-626, June 2004.

[IBM and Microsoft 2002] IBM and Microsoft, "Security in a Web Services World: A Proposed Architecture and Roadmap", Joint White Paper from IBM Corporation and Microsoft Corporation, http://www-106.ibm.com/developerworks/webservices/library/ws-secmap/, 2002.

[IBM and Microsoft 2003] IBM and Microsoft, "Secure, Reliable, Transacted Web Services: Architecture and Composition", Joint White Paper from IBM Corporation and Microsoft Corporation, http://msdn.microsoft.com/library/en-us/dnwebsrv/html/wsoverview.asp, 2003.

[IETF 1991] IETF, "Simple Network Management Protocol", RFC 3411, 1991.

[Imamura et al. 2002] T. Imamura, B. Dillaway and E. Simon, "XML Encryption Syntax and Processing", W3C Recommendation, http://www.w3.org/TR/xmlenc-core/, 2002.

[Inayat and Ezhilchelvan 2006] Q. Inayat and P. Ezhilchelvan, "A Performance Study on the Signal-On-Fail Approach to Imposing Total Order in the Streets of Byzantium", Proceedings of the International Conference on Dependable Systems and Networks, pp. 578-587, June 2006.

[Ingham et al. 2000] D. B. Ingham, S. K. Shrivastava and A. F. Panzieri, "Constructing Dependable Web Services ", IEEE Internet Computing, Vol. 4, No. 1, pp.25-33, 2000.

[Jeckle and Zengler 2003] M. Jeckle and B. Zengler, "Active UDDI - an Extension to UDDI for Dynamic and Fault-Tolerant Service Invocation", Web, Web-Services, and Database Systems 2002, pp.91-99, 2003.

[Jiang et al. 2005] S. Jiang, D. He and J. Rao, "A Prediction-Based Link Availability Estimation for Routing Metrics in MANETs", IEEE/ACM Transactions on Networking, Vol. 13, No. 6, pp. 1302-1312, December 2005.

[Kalbarczyk et al. 1999] Z. T. Kalbarczyk, R. K. Iyer, S. Bagchi and K. Whisnant, "Chameleon: A Software Infrastructure for Adaptive Fault Tolerance", IEEE Transactions on Parallel and Distributed Systems, Vol. 10, No. 6, pp.560-579, 1999.

[Karjoth et al. 2006] G. Karjoth, B. Pfitzmann, M. Schunter, and M. Waidner "Service-oriented Assurance-Comprehensive Security by Explicit Assurances", Proceedings of the 1st Workshop on Quality of Protection, LNCS, to appear in 2006.

[Kihlstrom et al. 2001] K. Kihlstrom, L. Moser, and P. Melliar-Smith, "The SecureRing Group Communication System", ACM Transactions on Information and System Security, Vol. 4, No. 4, pp. 371–406, 2001.

[Kim et al. 1990] K. Kim and T. Lawrence, "Adaptive Fault Tolerance: Issues and Approaches", Proceedings of the Second IEEE Workshop on Future Trends of Distributed Computing Systems, pp. 38-46, 1990.

[Kim et al. 1998] K. H. Kim and C. Subburaman, "ROAFTS: A Middleware Architecture for Real-Time Object-Oriented Adaptive Fault Tolerance Support", Proceedings of the IEEE High Assurance Systems Engineering, pp. 50-57, 1998.

[Knight et al. 2001] J. Knight, D. Heimbigner, A. Wolf, A. Carzaniga, J. Hill, and P. Devanbu, "The Willow Survivability Architecture", Proceedings of the 4th Information Survivability Workshop. 2001.

[Koopman and DeVale 1999] P. Koopman and J. DeVale, "Comparing the Robustness of POSIX Operating Systems", Proceedings of the 29th Annual International Symposium on Fault-Tolerant Computing 1999.

[Kopetz et al. 2003] H. Kopetz and G. Bauer, "The Time-Triggered Architecture", Proceedings of the IEEE, Vol. 91, No. 1, pp.112-126, 2003.

[Krawczyk 1993] H. Krawczyk, "Distributed Fingerprints and Secure Information Dispersal", Proceedings of the 12th ACM Symposiu on Principles of Distributed Computing, pp. 207–218, 1993.

[Kropp et al. 1998] N. P. Kropp, P. Koopman and D. P. Siewiorek, "Automated Robustness Testing of Off-the-Shelf Software Components", Proceedings of the 28th International Symposium on Fault-Tolerant Computing, pp. 230-239, June 1998.

[Lac 2006] C. Lac, "Software Availability Improvement with the Use of Failure Analysis", Lambda Mu 15, October 2006.

[Lac and Ramanathan 2006] C. Lac and S. Ramanathan, "A Resilient Telco Grid Middleware", IEEE Symposium on Computers and Communications, pp. 306-311, June 2006.

[Lala 2003] J. Lala (editor), "OASIS: Foundations of Intrusion Tolerant Systems", IEEE Computer Society Press, 2003

[Laprie 1985] J.-C. Laprie, "Dependable Computing and Fault Tolerance: Concepts and Terminology", Proceedings of the 15th International Symposium on Fault Tolerant Computing, pp.2-11, June 1985.

[Lee at al. 2006] U. Lee, J-S Park, J. Yeh, G. Pau, and M. Gerla, “CodeTorrent: Content Distribution using Network Coding in VANETs”, ACM Mobishare: 1st International Workshop on Decentralized Resource Sharing in Mobile Computing and Networking, 2006.

[Leymann 2001] F. Leymann, “Web Services Flow Language (WSFL 1.0)”, IBM Software Group, http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf, 2001.

[Liang et al. 2003] D. Liang, C.-L. Fang and C. Chen, “FT-SOAP: A Fault-Tolerant Web Service”, Tenth Asia-Pacific Software Engineering Conference, December 2003.

[Liu and Kaiser 2005] C. Liu and J. Kaiser, "A Survey of Mobile Ad Hoc Network Routing Protocols", http://www.minema.di.fc.ul.pt/reports/report_routing-protocols-survey-final.pdf, October 2005.

[Littlewood et al. 2004] B. Littlewood and L. Strigini, "Redundancy and diversity in security", Proceedings of the 9th European Symposium on Research in Computer Security, pp. 423-438, 2004.

[Looker et al. 2005] N. Looker, L. Burd, S. Drummond, J. Xu and M. Munro, “Pedagogic Data as a Basis for Web Service Fault Models”, IEEE International Workshop on Service-Oriented System Engineering, 2005.

[Looker and Munro 2005] N. Looker and M. Munro, “WS-FTM: A Fault Tolerance Mechanism for Web Services”, Technical Report 02/05, University of Durham, 2005.

[Looker et al. 2004] N. Looker, M. Munro and J. Xu, “WS-FIT: A Tool for Dependability Analysis of Web Services”, The 1st Workshop on Quality Assurance and Testing of Web-Based Applications, 2004.

[Looker and Xu 2003] N. Looker and J. Xu, “Assessing the Dependability of OGSA Middleware by Fault Injection”, Proceedings of the 22nd International Symposium on Reliable Distributed Systems, pp.293-302, 2003.

[Malkhi and Reiter 1998] D. Malkhi, and M. Reiter, “Byzantine Quorum Systems”, Distributed Computing, Vol. 11, pp. 203–213, 1998.

[Malkhi and Reiter 2000] D. Malkhi, and M. Reiter, “An Architecture for Survivable Coordination in Large Distributed Systems”, IEEE Transactions on Knowledge and Data Engineering, Vol. 12, No. 2, pp. 187–202, 2000.

[Marsden 2004] E. Marsden, "Caractérisation de la Sûreté de Fonctionnement de Systemes à base d'intergiciel", LAAS-CNRS, 2004.

[Marsden et al. 2002] E. Marsden, J.-C. Fabre and J. Arlat, “Dependability of CORBA Systems: Service Characterization by Fault Injection”, Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems, pp.276-285, October 2002.

[Martin and Alvisi 2004] J. Martin, and L. Alvisi, “A Framework for Dynamic Byzantine Storage”, Proceedings of the IEEE International Conference on Dependable Systems and Networks, pp. 325–334, June 2004.

[Martin-Guillerez et al. 2006] D. Martin-Guillerez, M. Banâtre and P. Couderc, “A Survey on Communication Paradigms for Wireless Mobile Appliances”, INRIA Report, May 2006.

[Mello et al. 2006] E. Ribeiro de Mello, S. Parastatidis, P. Reinecke, C. Smith, A. van Moorsel, and J. Webber "Secure and Provable Service Support for Human-Intensive Real-Estate Processes", Proceedings of 2006 IEEE International Conference on Services Computing, Chicago, Illinois, September 2006, p495-502. *[This work won FIRST PRIZE in the IEEE International Services Computing Contest, September 2006]*.

[Mian et al. 2006] A. Mian, R. Beraldi, and R. Baldoni, "Survey of Service Discovery Protocols in Mobile Ad Hoc Networks", Technical Report - Midlab 7/06, Dip. Informatica e Sistemistica "Antonio Ruberti", Università di Roma "La Sapienza", 2006.

[Narasimhan et al. 2005] P. Narasimhan, T. A. Dumitras, A. M. Paulos, S. M. Pertet, C. F. Reverte, J. G. Slember and D. Shrivastava, "MEAD: Support for Real-Time Fault-Tolerant CORBA", Concurrency and Computation: Practice and Experience, Vol. 17, pp.1527-1545, 2005.

[Menascé 2002] D. A. Menascé, "QoS Issues in Web Services", IEEE Internet Computing, Vol. 6, No. 6, pp.72-75, 2002.

[Merideth 2005] M. Merideth, T. Tai, S. Rouvellou, I. Narasimhan, "Thema: Byzantine-fault-tolerant Middleware for Web-service Applications", Proceedings of the 24th IEEE Symposium on Reliable Distributed Systems, pp. 131-140, October 2005.

[Microsoft 2004] Microsoft, "Web Services Reliable Messaging Protocol (WS-ReliableMessaging)", http://www-106.ibm.com/developerworks/webservices/library/ws-rm/, 2004.

[Moniz et al. 2006] H. Moniz, N. Neves, M. Correia, and P. Veríssimo, "Randomized Intrusion-Tolerant Asynchronous Services", Proceedings of the International Conference on Dependable Systems and Networks, pp. 568-577, June 2006

[Moser et al. 2000] L. Moser, P. Melliar-Smith, and N. Narasimhan, "The SecureGroup Communication System", Proceedings of the IEEE Information Survivability Conference, pp. 507–516, 2000.

[Murty 2004] R. Murty, "JULIET: A Distributed Fault Tolerant Load Balancer for .NET Web Services", Proceedings of the IEEE International Conference on Web Services, pp. 778-781, 2004.

[Nicomette and Deswarte 1996] V. Nicomette, and Y. Deswarte, "An Authorization Scheme for Distributed Object Systems", IEEE Symposium on Research in Privacy and Security. 1996.

[Pan et al. 2001] J. Pan, P. Koopman, D. P. Siewiorek, Y. Huang, R. Gruber and M. L. Jiang, "Robustness Testing and Hardening of CORBA ORB Implementations", Proceedings of the International Conference on Dependable Systems and Networks, 2001.

[Panjwani et al. 2005] S. Panjwani, S. Tan, K. Jarrin, and M. Cukier, "An Experimental Evaluation to Determine if Port Scans are Precursors to an Attack", Proceedings of the Internatinal Conference on Dependable Systems and Networks, pp. 602-611, June 2005.

[Park et al. 2006] S. Park, J. Song and B. Kim, "A Survivability Strategy in Mobile Networks", IEEE Transactions on Vehicular Technology, Vol. 55, No. 1, pp. 328-340, January 2006.

[Patterson et al. 2002] D. A. Patterson, A. Brown, P. Broadwell, G. Candea, M. Chen, J. Cutler, P. Enriquez, A. Fox, E. Kiciman, M. Merzbacher, D. Oppenheimer, N. Sastry, W. Tetzlaff, J. Traupman and N. Treuhaft,

"Recovery-Oriented Computing (ROC): Motivation, Definition, Techniques, and Case Studies", Technical Report UCB//CSD-02-1175, UC Berkeley Computer Science, 2002.

[Pauty et al. 2005] J. Pauty, P. Couderc and M. Banâtre, "Atomic Token Passing in the Context of Spontaneous Communication", INRIA Report 1679, January 2005.

[Perkins et al. 2000] C. E. Perkins, E. M. Royer and S. R. Das, "Ad-hoc On Demand Distance Vector (AODV) Routing", IETF Internet Draft, draft-ietfmanet-aodv-06.txt, July 2000.

[Porcarelli et al. 2003] S. Porcarelli, F. Di Giandomenico, A. Bondavalli, M. Barbera and I. Mura, "Service-Level Availability Estimation of GPRS", IEEE Transactions on Mobile Computing, Vol. 2, No. 3, pp. 233-247, July-September 2003.

[Powell et al. 1988] D. Powell, G. Bonn, D. Seaton, P. Veríssimo and F. Waeselynck, "The Delta-4 Approach to Dependability in Open Distributed Computing Systems", Proceedings of 18th IEEE Int. Symp. on Fault-Tolerant Computing Systems, pp. 246-51, June 1988.

[Powell and Stroud 2003] D. Powell and R. J. Stroud (editors), "Conceptual Model and Architecture of MAFTIA". Project MAFTIA IST-1999-11583, Deliverable D21, January 2003.

[OASIS 2004] OASIS, "Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)", OASIS Standard 200401, http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf, 2004.

[Rabin 1989] M. Rabin, O. "Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance", Journal of the ACM, Vol. 36, No. 2, No. 335–348, 1989.

[Ramasamy et al. 2002] H. Ramasamy, P. Pandey, J. Lyons, M. Cukier, and W. Sanders, "Quantifying the Cost of Providing Intrusion Tolerance in Group Communication Systems", Proceedings of the International Conference on Dependable Systems and Networks, pp. 229–238, June 2002.

[Ramasamy et al. 2004] H. V. Ramasamy, A. Agbaria, and W. H. Sanders, "CoBFIT: A Component-Based Framework for Intrusion Tolerance", Proceedings of the 30th EUROMICRO Conference, pp. 591-600, 2004.

[Ramasamy and Cachin 2005] H. V. Ramasamy, and C. Cachin, "Parsimonious Asychronous Byzantine-Fault-Tolerant Atomic Broadcast", Proceedings of the 9th International Conference on Principles of Distributed Systems, 2005.

[Ramasamy et al. 2005] H. V. Ramasamy, A. Agbaria, and W. H. Sanders, "Parsimony-Based Approach for Obtaining Resource-Efficient and Trustworthy Execution", Proceedings of the 2nd Latin-American Symposium on Dependable Computing, Vol. 3747 of LNCS, pp. 206-225, October 2005.

[Ramasubramanian et al. 2003] V. Ramasubramanian, Z. J. Haas and E. Sirer, "SHARP: a Hybrid Adaptive Routing Protocol for Mobile Ad Hoc Networks", MobiHoc, pp. 303-314, June 2003.

[Reis et al. 2005] G. A. Reis, J. Chang, N. Vachharajani, R. Rangan and D. I. August, "SWIFT: Software Implemented Fault Tolerance", Proceedings of the 3rd International Symposium on Code Generation and Optimization, 2005.

[Reiter 1994] M. Reiter, “Secure Agreement Protocols: Reliable and Atomic Group Multicast in Rampart”, Proceedings of the 2nd ACM Conference on Computer and Communications Security, pp. 68–80, 1994.

[Reiter 1995] M. Reiter, “The Rampart Toolkit for Building High-Integrity Services”, Theory and Practice in Distributed Systems, Vol. 938 of LNCS, pp. 99–110. 1995.

[Rodrigues et al. 2001] R. Rodrigues, M. Castro and B. Liskov, “BASE: Using Abstraction to Improve Fault Tolerance”, Proceedings of the 18th ACM Symposium on Operating System Principles, pp.15-28, 2001.

[Sabnis et al. 1999] C. Sabnis, M. Cukier, J. Ren, P. Rubel, W. H. Sanders, D. E. Bakken and D. A. Karr, "Proteus: A Flexible Infrastructure to Implement Adaptive Fault Tolerance in AQuA", Proceedings of the 7th IFIP International Working Conference on Dependable Computing for Critical Applications, pp. 137-156., 1999.

[Saggese et al. 2004] G. P. Saggese, C. Basile, L. Romano, Z. Kalbarczyk, and R. K. Iyer, “Hardware Support for High Performance, Intrusion- and Fault-Tolerant Systems”, Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems, pp. 195-204, October 2004.

[Salatge and Fabre 2006] N. Salatge and J.-C. Fabre, "A Fault Tolerance Support Infrastructure for Web Services based Applications", LAAS Research Report No. 06365, May 2006.

[Santos et al. 2005] G. T. Santos, L. C. Lung and C. Montez, “FTWeb: A Fault Tolerant Infrastructure for Web Services”, Proceedings of the Ninth IEEE International EDOC Enterprise Computing Conference, 2005.

[Schlichting and Schneider 1983] R. Schlichting and F. Schneider, “Fail-Stop Processors: An Approach to Designing Fault-Tolerant Computing Systems”, ACM Transactions on Computer Systems, Vol. 1(3), pp. 222-238, August 1983.

[Schneider 1990] F. Schneider, “Implementing Fault-Tolerant Services using the State Machine Approach: A Tutorial”, ACM Computing Surveys, Vol. 22, No. 4, pp. 299–319, 1990.

[Schneider and Toueg 1993] F. Schneider and S. Toueg, “Replication Management Using the State-Machine Approach”, S. Mullender, editor, Distributed Systems, Second Edition, Addison-Wesley, pp. 169-195, 1993.

[Silva et al. 2006] L. Silva, H. Madeira and J. G. Silva, “Software Aging and Rejuvenation in a SOAP-based Server”, Proceedings of IEEE Network Computing and Applications, 2006.

[Slabodkin 1998] G. Slabodkin, "Software Glitches Leave Navy Smart Ship Dead in the Water", 1998.

[Snow et al. 2000] A. P. Snow, U. Varshney, A. D. Malloy, "Reliability and Survivability of Wireless and Mobile Networks", IEEE Computer, Vol. 33, No. 7, pp. 49-55, July 2000.

[Stankovic and Popov 2006] V. Stankovic and P. Popov, "Improving DBMS Performance through Diverse Redundancy", Proceedings of the 25th International Symposium on Reliable Distributed Systems, October 2006.

[Stroud et al. 2004] R. J. Stroud, I. S. Welch, J. Warne, P. Ryan, “A Qualitative Analysis of the Intrusion Tolerance Capabilities of the MAFTIA Architecture”, Proceedings of the 2004 International Conference on Dependable Systems and Networks, pp. 453-464, June 2004.

[SUN 2003] SUN, “Web Services Reliable Messaging TC WS-Reliability”, http://www.oasis-open.org/committees/download.php/5155/WS-Reliability-2004-01-26.pdf, 2003.

[SUN 2004] SUN, "Java Management Extensions", 2004.

[Swift et al. 2004] M. M. Swift, M. Annamalai, B. N. Bershad and H. M. Levy, "Recovering Device Drivers", Proceedings of the 6th ACM/USENIX Symposium on Operating Systems Design and Implementation, 2004.

[Taiani et al. 2005] F. Taiani, J.-C. Fabre and M.-O. Kilijan, "A Multi-level Meta-object Protocol for Fault-Tolerance in Complex Architectures", Proceedings of the International Conference on Dependable Systems and Networks, June 2005.

[Tartanoglu et al. 2003a] F. Tartanoglu, V. Issarny, A. Romanovsky and N. Levy, “Coordinated Forward Error Recovery for Composite Web Services”, Proceedings of the 22nd Symposium on Reliable Distributed Systems, 2003.

[Tartanoglu et al. 2003b] F. Tartanoglu, V. Issarny, A. Romanovsky and N. Levy, “Dependability in the Web Services Architecture”, Architecting Dependable Systems. Vol. 2677 of LNCS, 2003.

[Thatte 2001] S. Thatte, “XLANG (Web Services for Business Process Design)”, Microsoft, http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm, 2001.

[Tosic et al. 2005] V. Tosic, B. Pagurek, K. Patel, B. Esfandiari and W. Ma, “Management Applications of the Web Service Offerings Language (WSOL)”, Information Systems, pp. 564-586, 2005.

[Valdes et al. 2004] A. Valdes, M. Almgren, S. Cheung, Y. Deswarte, B. Dutertre, J. Levy, H. Saidi, V. Stavridou, and T. E. Uribe, "An Architecture for an Adaptive Intrusion-Tolerant Server", Proceedings of the 10th International Workshop on Security Protocols, Vol. 2845 of LNCS, pp. 158-178, 2004.

[van der Meulen et al. 2005] M. van der Meulen, S. Riddle, L. Strigini and N. Jefferson, "Protective Wrapping of Off-the-Shelf Components", Proceedings of the 4th International Conference on COTS-Based Software Systems, pp. 168-177, 2005.

[Varshney and Malloy 2001] U. Varshney and A. D. Malloy, "Improving the Dependability of Wireless Networks Using Design Techniques", Proceedings pf the 26th Annual IEEE Conference on Local Computer Networks, pp. 122-131, November 2001.

[Veríssimo and Neves 2001] P. Veríssimo and N. Neves (editors), “Service and Protocol Architecture for the MAFTIA Middleware”, Project MAFTIA IST-1999-11583, Deliverable D23, 2001.

[Veríssimo and Rodrigues 2001] P. Veríssimo and L. Rodrigues, “Distributed Systems for System Architects”, Kluwer Academic Publishers, 2001.

[Veríssimo 2002] P. Veríssimo, “Intrusion Tolerance: Concepts and Design Principles. A Tutorial”, Technical Report DI/FCUL TR-02-6, Department of Computer Science, University of Lisbon. July 2002.

[Veríssimo 2003] P. Veríssimo, “Uncertainty and Predictability: Can they be Reconciled?”, Future Directions in Distributed Computing, Vol. 2584 of LNCS, 2003.

[Veríssimo et al. 2003] P. Veríssimo, N. Neves, and M. Correia, "Intrusion-Tolerant Architectures: Concepts and Design", Technical Report DI/FCUL TR-03-5, Department of Computer Science, University of Lisbon. April 2003.

[Veríssimo 2006] P. Veríssimo, "Travelling Through Wormholes: A New Look at Distributed Systems Models", SIGACTN: SIGACT News (ACM Special Interest Group on Automata and Computability Theory), Vol. 37, No. 138, 2006.

[Veríssimo et al. 2006] P. Verissimo, N. Neves, C. Cachin, J. Poritz, D. Powell, Y. Deswarte, R. Stroud, and I. Welch, "Intrusion-Tolerant Middleware: The Road to Automatic Security", IEEE Security & Privacy, Vol. 4, No. 4, pp. 54-62, July/August 2006.

[Wikipedia 2006] Wikipedia, "Service-Oriented Architecture", http://en.wikipedia.org/wiki/Service-oriented_architecture, last accessed Sep 19th 2006.

[Xu et al. 1995] J. Xu, B. Randell, A. Romanovsky, C. Rubira, R. Stroud, and Z. Wu, "Fault Tolerance in Concurrent Object-Oriented Software Through Coordinated Error Recovery", Proceedings of the 25th IEEE International Symposium on Fault-Tolerant Computing. June 1995.

[Xu et al. 1999] J. Xu, Z. Kalbarczyk and R. K. Iyer, " Networked Windows NT System Field Failure Data Analysis", Proceedings of the 1999 Pacific Rim International Symposium on Dependable Computing, 1999.

[Yoon et al. 2004] S. Yoon, D. Kim and S. Han, "WS-QDL Containing Static, Dynamic, and Statistical Factors of Web Services Quality", Proceedings of the IEEE International Conference on Web Services, pp. 808 - 809, 2004.

[Yu-jie et al. 2005] M. Yu-jie, C. Jian, Z. Shen-sheng and Z. Jian-hong, "Interactive Web Service Choice-making based on Extended QoS Model", Proceedings of the Fifth International Conference on Computer and Information Technology, pp.1130-1134, 2005.

[Zhang and Mouftah 2005] B. Zhang and H. T. Mouftah, "QoS Routing for Wireless Ad Hoc Networks: Problems, Algorithms and Protocols", IEEE Communications Magazine, October 2005.

[Zhou et al. 2002] L. Zhou, F. B. Schneider, and R. v. Renesse, "COCA: A Secure Distributed On-line Certification Authority", ACM Transactions on Computer Systems, Vol. 20, No. 4, pp. 329-368, 2002.

[Zhu et al. 2005] F. Zhu, M. W. Mutka and L. M. Ni, "Service Discovery in Pervasive Computing Environments", IEEE Pervasive Computing, Vol. 4, No. 4, October 2005.

Part Algo – Resilience Algorithms and Mechanisms

Co-ordinator: Michel Raynal

Contributors: Roberto Baldoni[8], Miguel P. Correia[5], Ludovic Courtes[3], Felicita Di Giandomenico[7], Fabrizio Grandoni[7], Marc-Olivier Killijian[3], Nuno Ferreira Neves[5], Nouha Oualha[1], Thea Peacock[6], David Powell[3], Adriano Rippa[8], Yves Roudier[1], Michel Raynal[2], Peter Ryan[6], Marco Serafini[4], Neeraj Suri[4], Sara Tucci Piergiovanni[8], Paulo Veríssimo[5]

[1]Eurecom, [2] IRISA, [3]LAAS-CNRS, [4]Technische Universität Darmstadt, [5]Universidade de Lisboa, [6]University of Newcastle upon Tyne, [7]University of Pisa, [8]University of Roma

Chapter co-ordinators:

1 - Byzantine Consensus in Asynchronous Message-Passing Systems: Miguel Correia

2 - On-Line Diagnosis of Transients in Distributed Systems: Felicita Di Giandomenico

3 - A Survey of Cooperative Backup Mechanisms: Marc-Olivier Killijian

4 - Wait-free objects: an introduction for the sophomore: Michel Raynal

5 - Cooperation Incentive Schemes: Nouha Oualha &Yves Roudier

6 - Connectivity in Unstructured Overlay Networks: Roberto Baldoni

7 - High assurance voting systems**:** Peter Ryan

Introduction

Background

What is our research domain

Informatics can be defined as the meeting point between mathematics and technology [Lamport and Raynal 2004]. Roughly speaking, its two components, *computer science* and *computer engineering*, can be seen as complementary facets: computer science is to understand, computer engineering is to build. Said in another way, we are concerned with a science of *abstraction*, namely, creating the right model for a problem and devising the appropriate mechanizable techniques to solve it [Aho and Ullman 1992]. This is particularly true in (fault-tolerant/dynamic/large-scale/etc.) distributed computing where finding models that are realistic while remaining abstract enough to be tractable, was, is and still remains a real challenge [Schneider 1993].

Distributed computing was born in the late seventies when people started taking into account the intrinsic characteristics of physically distributed systems. The field then emerged as a specialized research area distinct from networks, operating systems and parallelism. Its birth certificate is usually considered as the publication in 1978 of Lamport's most celebrated paper "*Time, clocks and the ordering of events in a distributed system*" [Lamport 1978] (that paper was awarded the Dijkstra Prize in 2000). Since then, several high level journals and (mainly ACM and IEEE) conferences are devoted to distributed computing.

Distributed computing arises when one has to solve a problem in terms of entities (usually called processes, agents, sensors, peers, actors, processors, nodes, etc.) such that each entity has only a partial knowledge of the many parameters involved in the problem that has to be solved. While parallelism and real-time can be characterized by the words *efficiency* and *on-time computing*, respectively, distributed computing can be characterized by the word *uncertainty*. Mastering one form or another of uncertainty is pervasive in all distributed computing problems. Finally, as the aim of a theory is to codify knowledge so that it can be transmitted [Fischer and Merritt 2003] (to students, engineers, etc), more research to discover and clearly state basic principles underlying resilient and scalable dynamic distributed systems is necessary.

The ReSIST Working Group on Resilience Algorithms

The ReSIST partners are investigating algorithms and mechanisms that enable the implementation of resilience. A major issue is the design of algorithms for fault-tolerant asynchronous computing. Other areas concern advanced diagnosis, backup mechanisms, fault-tolerant paradigms for large-scale services, cooperation schemes, wait-free computing, and overlay connectivity in dynamic peer-to-peer systems. Last, some ReSIST partners are working on algorithms supporting resilince in specific large scale socio-technical systems.

This aim of this short introduction is to present the main contributions made in these areas by members of the ReSIST

Algorithms working group. The following chapters briefly introduce the current state of knowledge on these hot topics. Each has been addressed by a ReSIST partner. It is worth noting that, in one way or another, all these topics are concerned by *scalability* (a ReSIST central issue). The ReSIST student seminar held in San Miniato (5-8 September 2006) has shown that the PhD students involved in ReSIST have already established strong links for future contributions on these topics.

The following sentences are from an interview of Fred Schneider that appeared in the IEEE magazine *Distributed Systems Online*. They can be seen as a statement of a part of the philosophy that guides the ReSIST group on Algorithms.

> "*We need to develop qualitatively different approaches. I would make an analogy with macro-scale physical phenomena like fluid, flows and heat. Understanding the behavior of individual atoms or molecules does not explain things such as turbulence or temperature. And we don't derive those properties from first principles -instead we have additional sets of laws for the different levels of abstraction. I believe that when we start looking at large distributed systems as "networks", as opposed to as interconnected hosts, there will be new abstractions and new laws that let us predict behavior and do so in a way that is not directly related to first principles. We need laws for the various levels of abstractions, and we don't yet have those laws.*"

Agreement in presence of Byzantine failures

Nowadays, distributed algorithms are being used as important building blocks for the construction of secure applications based on a recent approach: Intrusion Tolerance. The idea is to make distributed systems advance to a new level of resilience by combining classical Security techniques with Fault Tolerance mechanisms. The link between the two areas are the Security concepts of attack and intrusion, which can be encompassed in the Fault Tolerance concept of arbitrary (or Byzantine) fault. Therefore, fault-tolerant mechanisms can be used to tolerate the occurrence of these malicious events.

Consensus is a classical distributed systems problem with both theoretical and practical interest. The problem can be stated informally as: how to make a set of distributed processes achieve agreement on a value despite a number of faulty processes. The importance of this problem derives from several other distributed systems problems being reducible or equivalent to it. Examples are atomic broadcast, non-blocking atomic commit, group membership, and state machine replication. These reducibilities and equivalences have two important consequences. The first is that these problems are usually simple to solve using a consensus algorithm as building block. The second is that several theoretical results stated for consensus automatically apply to these other problems.

For these two reasons, the study of consensus algorithms that tolerate arbitrary/Byzantine faults – often called Byzantine Agreement – has an important role in Intrusion Tolerance. One of the survey chapters that follow presents recent work in this area. The survey, titled "*Byzantine consensus in asynchronous message-passing systems*" by M. Correia, N. Neves and P. Veríssimo [Lisbon, Portugal]), starts by presenting several definitions of the problem, since several variations have been studied in the literature.

The second part of the survey discusses several solutions to solve consensus. Consensus has been shown to be impossible to solve deterministically if even a single process is allowed to crash, a result often called the FLP impossibility. Solutions for consensus are classified in terms of the way the FLP impossibility is circumvented: sacrificing determinism, adding time to the model, augmenting the system model with an oracle, or modifying the problem.

The third part of this survey chapter deals with scalability issues, which are central to ReSIST. The above-mentioned relations between consensus and other distributed problems imply that many results stated for consensus automatically apply to these other problems. Among these results, several have to do with how many faulty processes can be tolerated and how efficiently the problem can be solved, i.e., about the performance of algorithms that solve consensus (and related problems). Important metrics of efficiency are the minimum number of asynchronous steps and messages needed to solve consensus. These metrics are important to assess the scalability of the algorithms, although recent work has shown also

their limitations.

Issues on On-Line Diagnosis of Transients in Distributed Systems

The research on on-line diagnosis of transient faults focuses on distributed fault tolerant systems, which are designed to provide sustained delivery of services despite encountered perturbations. In such context, two primary objectives are (a) to consistently identify a faulty node so as to restrict its effect on the system operations, and (b) to support the process of recovery via isolation and reconfiguration of the system resources to continue system operations. If these operations can be performed as on-line procedures, this can help provide a prompt reaction to faults and an effective capability of dynamic resource management, thus mitigating the system's susceptibility to subsequent fault occurrences. Physical faults are distinguished by their nature and duration of impact as being permanent or temporary (these latter being further divided into intermittent and transient). While permanent/intermittent faults are caused by some internal part deviating from its specified behavior and the only way to handle such faults is to remove the affected component, transient faults cannot be easily traced to a defect in a particular part of the system and, normally, their adverse effects tend to disappear after some time. Considering that most perturbations encountered are transient, "proper" diagnosis of transients is a significant issue of interest. Actually, good discrimination between transient and intermittent/permanent faults solves two important problems: i) prevents the undue removal of nodes affected by transient faults, thus avoiding unnecessary depletion of system resources; and ii) helps to maintain the correct coverage of the system fault hypotheses (i.e., the assumption on the number of faults tolerated by the core system protocols within a given time window) by keeping in operation nodes not permanently faulty.

In ReSIST, the above mentioned issues are addressed by the development of a Fault Detection - Isolation - Recovery diagnostic process (FDIR) to cope specifically with transients. The knowledge on diagnosis of transient faults mainly embraces two directions: i) distributed diagnostic algorithms for synchronous systems, and ii) "count-and-threshold" schemes and architectural framework for handling multiple classes of faults in off-the-shelf (COTS) components and legacy-based applications. Details are in the survey on this topic, included here, titled: "*On-line diagnosis of transients in distributed systems*" (by M. Serafini, F. di Giandomenico, F. Grandoni and N. Suri [Pisa, Italy and Darmstadt, Germany]).

Current research interests are going in several directions, including: i) definition of a generic on-line FDIR framework, able to generalize the previously proposed distributed diagnosis protocols and to enhance them with count-and-threshold techniques in order to effectively handle transient faults; ii) a formalization of the diagnosis process, addressing the whole chain constituted by the monitored component, the deviation detector and the state diagnosis; iii) diagnosis in mobile distributed environments. The proposed research addresses to a good extent the resilience scalability issues addressed by ReSIST. Since the basic count-and-threshold mechanism ideally applies to identifiable subsystems, it naturally suits the evolution of the system at large. Practically all design methodologies go through hierarchical subsystem composition to nurture complexity-controlled system growth; then, as higher levels of the hierarchy are constructed, properly implemented mechanisms of this kind can be introduced in subsystems there defined. Being the diagnosis mechanisms applied on-line, the reaction to the discovery of faulty modules/components can be tailored to the real system and environment conditions. On the assessability front, it is easy to recall that the main goal of these mechanisms is exactly to be able to obtain specific performance figures from a device, for which mostly qualitative measures of merit were available.

New paradigms for large-scale services

Over the past decade, the ever-improving performance of Internet access has made it possible to deploy online services. These distributed services are becoming larger and span a wide range of applications. Scalability issues arose as a result of the centralized client-server model that has been used when designing such services. At the same time, servers implementing online services are increasingly relied on, thereby becoming single points of failure.

New distributed programming paradigms emerged recently as an improvement over centralized client-server models for large-scale services. These new paradigms, referred to as "peer-to-peer" or "grid computing" (depending on the context in which they are used), have been designed to address scalability issues at a large scale by distributing service provision among a large, dynamically-changing set of nodes connected to the network. A natural side-effect is that none of the service-providing nodes in this model is a single point of failure. In particular, peer-to-peer systems are explicitly designed to adapt dynamically to the changing execution context, such as arrival or departure of nodes.

Additionally, since the peer-to-peer approach relies on cooperation among participating nodes, it requires that a number of security aspects be addressed, especially when the nodes are mutually suspicious. These concerns yielded a large amount of research work aimed at making open peer-to-peer systems resilient against attacks.

The peer-to-peer approach has been successfully used to implement a variety of services where resources are shared among participating nodes. The most famous and widely deployed peer-to-peer applications are information sharing and bandwidth aggregation. Both applications achieve high resilience thanks to their highly distributed and dynamic nature. Emerging applications include large-scale sharing of computing resources. With the advent of appropriate security primitives, operating system and programming language research are both evolving in a way that would allow for pervasive resource sharing over the network.

The cooperative service approach has been taken in a variety of research areas: file sharing, censorship-resistant publication, packet-forwarding for mobile ad-hoc networks, distributed computations, web crawling and indexing, data backup. A survey of cooperative approaches to backup is given here in the chapter "*A survey of cooperative backup mechanisms*" (by M. Killijian, L. Courtès and D. Powell [LAAS, Toulouse, France]).

The cooperative approach to backup illustrates how the diversity and dynamism inherent to the peer-to-peer paradigm can be leveraged to implement resilience-enhancing services. In particular, the survey shows that data backup is becoming a successful application on local-area and wide-area networks. Recent work promoted the idea that cooperative and peer-to-peer approaches could be applied as well to the context of mobile ad-hoc networks (MANETs) in applications such as file sharing and data backup.

Wait-free asynchronous computing

The notion of *wait-free* computing appeared a long time ago (in the late seventies). It simultaneously addresses notions such as scalability, fault-tolerance, resilience, and liveness in systems where processes can fail (only) by crashing. Let us consider an object that can be concurrently accessed by several processes. That object provides its users with operations, each one implemented by an appropriate algorithm. The implementation of the object is said to be *wait-free* if any operation executed by a process that is fault-free terminates in a finite number of its own processing steps. This means that the operation terminates whatever the number of operations executed concurrently, whatever the speed of the other processes, whether processes crash or not.

As we can see, a wait-free implementation of an object is naturally deadlock-free and copes naturally with the crash of any number of processes. As a consequence, wait-free algorithms are good candidates for solving problems where the system can dynamically evolve and the number of processes is not always known. The introductory survey to wait-free computing constitutes a "lesson number 1" in this domain. This survey is titled "*Wait-free objects: an introduction for the sophomore*" (by M. Raynal [IRISA, Rennes, France]).

More specialized results are described in the research papers in the appendix to this part of the deliverable.

Cooperation schemes

Decentralized system algorithms and protocols have recently received a lot of interest in mobile ad-hoc networks as well as in peer-to-peer (P2P) systems. The development of such techniques is a necessity to be able to attain cost-effective and reliable applications in this setting, yet it brings up far-reaching issues that have to be dealt with. In decentralized systems, decision-making may not be located at a specific and central group of devices (repeaters, bridges, routers, gateways, servers) but can be distributed to end-user devices. Decisions and actions may use the computing power, bandwidth, and disk storage space of all the participants (peers) in the network rather than being concentrated in a relatively low number of special devices. The decentralized structure makes it possible to achieve minimal administrative and operational costs. Peers in this type of system normally have equivalent responsibilities and privileges. The intricate notions of self-organization and self-management require that each peer provide its own contribution for the correct operation of the system.

The idea of handing basic mechanisms of the system over to autonomous peers raises new concerns, in particular with respect to the establishment of trust between peers, to the stimulation of their cooperation, and to the fairness of their respective contributions. Self-organization opens up new security breaches because a peer must be able to defend against others perpetrating new forms of denial of service. Selfishness, as illustrated by the so-called free-riding attack, is a first type of such a threat in which the attacker (called free-rider) benefits from the system without contributing its fair share. Systems vulnerable to free-riding either run at reduced capacity or collapse entirely because the costs of the system weigh more and more heavily on the remaining honest peers encouraging them to either quit or free ride themselves. Flooding is a second type of denial of service: the attack can be launched by sending a large number of query messages asking for resources to a victim peer in order to slow it until it is unusable or crashes. For example, an attacker can attempt to make a lot of read and write operations in a distributed storage application. Cheating (or retention) is a third form of denial of service in which the attacker retains data required for the system to work or does not comply with the normal course of action in order to obtain an unfair advantage over other peers. So-called “cooperation enforcement” mechanisms (which should more properly be called cooperation incentive schemes) provide ways of managing and organizing resources and aim at dealing with the security challenges that traditional security approaches (e.g., authentication, access control) cannot cope with.

The survey titled “*A survey of cooperation incentive schemes*” (by N. Oualha and Y. Roudier [Eurecom, Nice, France]) introduces applications for cooperation incentives, then details how incentive schemes work, and finally discusses how these schemes may be validated.

Overlay network connectivity

Overlay maintenance protocols have been introduced to guarantee overlay network *connectivity* in dynamic large scale peer-to-peer (P2P) systems. These algorithms have been specifically designed to avoid the partitioning of the overlay in clusters (network breakage) despite node dynamism (join/leave). However, their design until now had the main goal of arranging the overlay network in a graph able to remain connected after a massive leave occurrence. Unfortunately, this widely-used approach does not explicitly address the problem of the continual arrival/departure of nodes (*churn*), which in turn heavily affects P2P systems.

The chapter titled “*Overlay Network Connectivity in Unstructured Overlay Networks*” (by R. Baldoni, A. Rippa and Sara Tucci-Piergiovanni [Roma, Italy]) surveys current overlay maintenance protocols with the precise goal of highlighting how they work in several scenarios characterized by different levels of dynamism affecting P2P systems. These protocols are presented by dividing them in two main categories according to the way they refresh network links at each node, namely *reactive* protocols, and *proactive* protocols. In a reactive protocol, a node A changes its links when A is joining the network, or a node B joins the network using A as “introducer”, or A has a link to a node B leaving the network.

In a proactive protocol, in addition to the previous cases, a node A also changes its links periodically by starting a link exchange session with another node.

Then, the survey presents how these overlay maintenance protocols behave under different levels of dynamism. In particular, the levels of dynamism that are investigated include: semi-static, dynamic with constant churn, and fully dynamic. The first level considers a network obtained by a serialized sequence of joins, then several network properties are evaluated either without considering any other join/leave or after some number of leaves. The second level represents a more realistic characterization of the environment: the network is affected by churn but the size of the network is constant along the time. The third level considers a network affected by churn and the size of the network may vary with time arbitrarily.

The parameters of interest for protocol evaluation are reachability (information about how many nodes a peer is able to reach with a multicast message in the network), link distribution (how the links are distributed among nodes of the network), diameter (longest path in the network), overlay clustering (information about how many clusters are in the network and their size).

This survey points out that current protocols, created to be used in dynamic environments, are not able to maintain declared performance in realistic dynamic environments but only in partially dynamic situations. ¿From these results it appears clear that the overlay maintenance problem needs further study in more realistic environments and possibly in fully dynamic networks to give a more practical contribution to the research.

Dependable voting systems

The last chapter in this Part discusses algorithms to ensure resilience in a specific critical application, “E-voting”. Dependable voting systems provide the bedrock on which democracy rests. It is essential not only that voting systems be trustworthy, they must also be seen by the electorate at large to be dependable. The importance of this has been graphically highlighted recently by the debacles in the US 2000 and 2004 presidential elections. This has spurred a significant amount of research into voter- verifiable, cryptographically based schemes over the last few years.

The study of voting schemes is highly interdisciplinary. The information and communication technology is intertwined with human organisations; even for what concerns the technology alone, voting schemes are required to satisfy a number of very subtle and often conflicting technical security requirements: accuracy, privacy, coercion-resistance, universal verifiability, etc. Typically, cryptographic schemes aim to achieve all of these goals with minimal trust requirements, i.e., avoiding any need to trust hardware, software, election officials, suppliers etc. This is in stark contrast to the touch screen style machines widely used in the US in 2000 and 2004. Here, the voter must place complete faith in the voting device to ensure the integrity and secrecy of their vote.

They must also meet a number of social requirements: usability, cost-effectiveness, understandability, accessibility, public trust.

The starting point for the ReSIST work in this area was the analysis performed at Newcastle of a voter-verifiable scheme due to Chaum. This analysis led Ryan (Newcastle) to propose an alternative approach to the creation of the encrypted receipt. This scheme, dubbed “Prêt à Voter”, is significantly simpler both technologically and conceptually whilst retaining the technical properties of the original. Since the publication of the original scheme, numerous enhancements have been developed, for example, the use of randomising encryption primitives such as ElGamal and Paillier to enable re-encryption mixes in place of the original decryption mixes, supervised casting of encrypted ballots along with Verified Encrypted Paper Audit Trails (VEPAT) etc. Besides advances on the technical front, various studies have been conducted on the socio-technical aspects of such schemes: understandability and public trust, socio-technical vulnerabilities and countermeasures.

The most significant advances in this field, with special emphasis on the contributions within ReSIST, are surveyed in the chapter entitled "*High Assurance Voting Systems*", by P.Y. A. Ryan and T. Peacock).

1– Byzantine Consensus in Asynchronous Message-Passing Systems: a Survey

Introduction

This chapter presents a short survey on Byzantine consensus – or Byzantine agreement – in asynchronous message-passing distributed systems. Consensus is a classical distributed systems problem, first introduced in [Pease et al., 1980], with both theoretical and practical interest [Lynch, 1996, Guerraoui et al., 2000]. The problem can be stated informally as: how to ensure that a set of distributed processes achieve agreement on a value despite a number of faulty processes. The importance of this problem derives from several other distributed systems problems being reducible or equivalent to it. Examples are atomic broadcast [Hadzilacos and Toueg, 1994, Chandra and Toueg, 1996, Correia et al., 2006c], non-blocking atomic commit [Guerraoui and Schiper, 2001], group membership [Guerraoui and Schiper, 2001], and state machine replication [Schneider, 1990].

These relations between consensus and other distributed problems are important because many results stated for consensus automatically apply to these other problems. Among these results, the most relevant is probably the impossibility of solving consensus deterministically in an asynchronous system if a single process can crash, often called the FLP result [Fischer et al., 1985]. This result lead to a large number of works that try to circumvent it, i.e., to slightly modify the system model in such a way that consensus becomes solvable. Examples include randomization [Rabin, 1983, Ben-Or, 1983], failure detectors [Chandra and Toueg, 1996, Malkhi and Reiter, 1997], partial-synchrony [Dwork et al., 1988, Dolev et al., 1987], and wormholes [Correia et al., 2005, Neves et al., 2005].

There is another set of results that have to do with how many faulty processes can be tolerated and how efficiently the problem can be solved, i.e., about the performance of algorithms that solve consensus. Important metrics of efficiency are the minimum number of asynchronous steps and messages needed to solve consensus (a survey of early results is given in [Lynch, 1989]). These metrics are important to assess the scalability of the algorithms, although recent work has shown that they can hide as much as they show [Keidar, 2002, Moniz et al., 2006].

Algorithms that solve consensus vary heavily depending on the system model, like algorithms that solve any other distributed systems problem. This chapter considers only message-passing algorithms that tolerate Byzantine (i.e., arbitrary) faults in asynchronous systems (i.e., without time assumptions).

The reason for choosing this system model is not merely its theoretical interest. Today, algorithms based on this model are being used as important building blocks for the construction of secure applications based on a recent approach: *intrusion tolerance* [Fraga and Powell, 1985, Adelsbach et al., 2002, Lala, 2003, Veríssimo et al., 2003, Avizienis et al., 2004]. This approach can be considered to be part of the ongoing effort to make computing systems, Internet included, more secure vis-a-vis the large number of security incidents constantly reported by entities like the CERT/CC[1].

[1] http://www.cert.org/stats

Some brief comments are due on the three main aspects of the system model we consider: message-passing, Byzantine faults and asynchrony.

The message-passing model is the natural choice for algorithms supposed to be executed not in parallel machines but generic distributed systems, like those built on top of the Internet. An alternative system model is shared-memory [Attie, 2002, Friedman et al., 2002, Malkhi et al., 2003, Alon et al., 2005, Bessani et al., 2006], but in the kind of system we are considering the shared memory itself has to be implemented using message passing algorithms.

Arbitrary faults, usually called Byzantine after the paper by Lamport et al. [Lamport et al., 1982], do not put any constraint on how processes fail. This sort of assumption or, better said, of non-assumption about how processes fail, is specially adequate for systems where malicious attacks and intrusions can occur. For instance, a process might be executed by an attacker that modifies the behavior of its implementation of the algorithm arbitrarily. Interestingly, assuming Byzantine faults, instead of the more typical assumption of crash faults, leads to more complex and challenging algorithms.

Asynchrony might also be better described as a non-assumption about time properties. This (non-) assumption is important because attackers can often violate some time properties by launching denial-of-service attacks against processes or communications. For instance, the attacker might delay the communication of a process indefinitely, breaking any assumptions about process timeliness.

This system model – assuming Byzantine faults and asynchrony – is very generic. Algorithms in this model have to deal with two independent degrees of uncertainty: in terms of faults and time. This leads to algorithms that besides being able to deal with malicious behavior, have the virtue of also running correctly in more benign environments, like those in which only crash faults occur or in which delay bounds are attained.

The consensus problem is defined for a set of n known processes. Currently there is a trend of research on large dynamic systems in which the number of involved processes is unknown [Mostefaoui et al., 2005, Aguilera, 2004]. Consensus, however, is still not defined in this context.

1.1. Byzantine Consensus Definitions

This section defines the consensus problem and several of the variations considered in the literature. We say that a process is *correct* if it follows its algorithm until completion, otherwise it is said to be *faulty*.

A *binary consensus* algorithm aims to achieve consensus on a binary value $v \in \{0, 1\}$. Each process proposes its *initial value* (binary) and decides on a value v. The problem can be formally defined in terms of three properties:

- *Validity:* If all correct processes propose the same value v, then any correct process that decides, decides v.
- *Agreement:* No two correct processes decide differently.
- *Termination:* Every correct process eventually decides.

The first 2 properties are *safety* properties, i.e., properties that say that some bad thing cannot happen, while the last is a *liveness* property, i.e., a property that states good things that must happen.

Multi-valued consensus is apparently similar to binary consensus, except that the set of values has arbitrary size, i.e., $v \in V$ and $|V| > 2$. Multi-valued consensus algorithms have been studied in the literature using several Validity properties, while the Agreement and Termination properties remained essentially the same (with minor exceptions for Termination that we will see later). Some papers use the following Validity property [Dwork et al., 1988, Malkhi and Reiter, 1997, Kihlstrom et al., 2003]:

- *Validity 1:* If all correct processes propose the same value v, then any correct process that decides, decides v.

Others use the following [Doudou and Schiper, 1997, Doudou et al., 2002, Baldoni et al., 2003]:

- *Validity 2:* If a correct process decides v, then v was proposed by some process.

Both properties are somewhat weak. Validity 1 does not say anything about what is decided when the correct processes do not propose all the same v, while Validity 2 does not say anything about what is the value decided (e.g., is it the value proposed by the correct processes if all of them propose the same?). Recently, a definition that gives more detail about what is decided has been proposed [Correia et al., 2006c]. The definition has three Validity properties:

- *Validity 1:* If all correct processes propose the same value v, then any correct process that decides, decides v.
- *Validity 2':* If a correct process decides v, then v was proposed by some process or $v = \perp$.
- *Validity 3:* If a value v is proposed only by corrupt processes, then no correct process decides v.

The first two are essentially the Validity properties already introduced, except that Validity 2' allows the decision of a value $\perp \notin V$. The third property is inspired by the original definition in the context of the "Byzantine Generals" metaphor used in the classical paper by Lamport et al. [Lamport et al., 1982]. The definition was "(1) All loyal generals decide upon the same plan of action; (2) A small number of traitors cannot cause the loyal generals to adopt a bad plan.". That paper however, considered a synchronous system, i.e., a system in which there are known delay bounds for processing and communication.

This concern about the practical interest of multi-valued consensus defined in terms of Validity 1 or Validity 2 lead also to the definition of *vector consensus* [Doudou and Schiper, 1997]. The difference in relation to the previous versions of the problem is once again the Validity property. The decision is no longer a single value but a vector with some of the initial values of the processes, at least $f + 1$ of which are from correct processes. The validity property is now stated as:

- *Vector validity:* Every correct process that decides, decides on a vector V of size n:
  - $\forall_{p_i}$: if p_i is correct, then either *V[i]* is the value proposed by p_i or $\perp$;
  - at least $(f + 1)$ elements of V were proposed by correct processes.

Vector consensus is a variation for asynchronous systems of the classical *interactive consistency* problem [Pease et al., 1980]. Interactive consistency is a consensus on a vector with values from all correct processes. However, in asynchronous systems a correct process can be very slow so it is not possible to guarantee that values from all correct processes are obtained and still ensure Termination. Therefore, vector consensus ensures only that $f + 1$ of the values in the vector are from correct processes. This is clearly more interesting than multi-valued consensus since it tells much more about the initial values of the correct processes. Interestingly, vector consensus has been proved to be equivalent to multi-valued consensus defined with the validity properties Validity 1, Validity 2' and Validity 3 [Correia et al., 2006c].

Some other variations of consensus have been studied in the literature. For instance, the *k-set consensus* problem in which the correct processes can decide at most k different values [Chaudhuri, 1993, de Prisco et al., 1999].

A somewhat different kind of definition is the one used in the Paxos algorithms [Lamport, 1998, Lamport, 2001, Lampson, 2001, Zielinski, 2004, Martin and Alvisi, 2005]. The idea is that processes play one or more of the following roles: *proposers* (propose values), *acceptors* (choose the value to be decided) and *learners* (learn the chosen value). The problem can be defined in terms of five properties [Lamport, 2001, Martin and Alvisi, 2005]:

- Only a value that has been proposed may be chosen.
- Only a single value may be chosen.
- Only a chosen value may be learned by a correct learner.
- Some proposed value is eventually chosen.
- Once a value is chosen, correct learners eventually learn it.

The first three properties are safety properties, while the last two are liveness properties. This definition is interesting because it allows a simple implementation of state machine replication in the crash failure model [Schneider, 1990, Lamport, 2001]. However, in the Byzantine failure model this is not so simple [Castro and Liskov, 2002, Martin and Alvisi, 2005].

1.2. FLP Impossibility

The most cited paper about consensus is probably the one that proves the impossibility of solving consensus deterministically in an asynchronous system if a single process can crash [Fischer et al., 1985]. This result is often called the FLP impossibility result, after its authors' names, Fischer, Lynch and Paterson. The consensus definition used to prove the result is even weaker than the first definition in Section 1.1.: validity is more relaxed and termination is required for a single process.

The idea is that the uncertainty in terms of timeliness (asynchrony) combined with the uncertainty in terms of failure (even if failures are only crashes and only one process can fail) does not allow any deterministic algorithm to guarantee the binary consensus definition given in the previous section. More precisely, the reason for the impossibility is that in an asynchronous system it is impossible to differentiate a crashed process from another that is simply slow (or connected by a slow network link). In the years that followed the statement and proof of this result, a few alternative proofs have been given (a discussion of these proofs can be found in [Lynch, 1989]).

The FLP result says when consensus is not solvable. However, from a practical point of view it is more important to know when it can be solved. A first detailed study of this issue was presented by Dolev, Dwork and Stockmeyer for crash faults [Dolev et al., 1987]. The paper identified five relevant parameters that affect solvability: synchrony/asynchrony of the processes; synchrony/asynchrony of the communication; ordered/unordered message delivery; broadcast/point-to-point communication; and atomic/not-atomic receive and send. This lead to 32 different models. The paper showed that different degrees of synchronism allow deterministic algorithms to tolerate different numbers of crash faults (there was no study for Byzantine faults).

To solve consensus, an algorithm has to *circumvent* the FLP impossibility result. This word, circumvent, is quite unprecise so it is important to discuss its meaning. The idea is to slightly modify either the system model or the problem definition considered in [Fischer et al., 1985] to allow the problem to be solvable. These modifications change the conditions in which FLP was proven so, to be precise, the result simply no longer applies. However, researchers in the area prefer to call it "circumventing" the result, to pass the idea that the conditions are close to those in which the result applies.

An observation about FLP with interesting practical consequences is that if we discard one of the properties that define consensus, we can enforce the two others. This observation lead researchers to design consensus algorithms in the following way:

- the algorithm solves consensus if the technique used to circumvent FLP works as it is assumed to;
- the algorithm satisfies the safety properties even if the technique used to circumvent FLP does not work as it assumed to [Guerraoui, 2000, Guerraoui and Raynal, 2004].

The idea is that if something bad happens, like an additional time assumption not being satisfied, the algorithm may not terminate, but Validity and Agreement properties will always be satisfied. This notion has been recently called *indulgence* in the context of system models augmented with eventual failure detectors [Guerraoui, 2000, Guerraoui and Raynal, 2004], but almost all consensus algorithms satisfy it. The only exception that we are aware of is the randomized algorithm in [Canetti and Rabin, 1993], which always terminates but only satisfies Agreement with a certain probability.

There are several ways to look into the techniques to circumvent FLP. We use a classification in four types of techniques,

which we present in more detail in the following section:

- techniques that sacrifice determinism, leading to probabilistic algorithms;
- techniques that add time to the model;
- techniques that enrich the system model with an oracle;
- techniques that enrich the problem definition.

1.3. Circumventing FLP

The following sections introduce the techniques to circumvent FLP and algorithms that use them.

1.3.1. Sacrificing Determinism

The FLP impossibility result applies to *deterministic* algorithms so a solution to circumvent it is by using *randomization* to design *probabilistic* algorithms. More specifically, the idea is to substitute one of the properties that define consensus by a similar property that is satisfied only with a certain probability. For the reasons mentioned above, almost all algorithms choose to modify the Termination property, which becomes:

- *P-Termination:* Every correct process eventually decides with probability 1.

The single exception that we are aware of, already mentioned above, does not modify Termination but Agreement, so agreement on the value decided is reached with a certain probability [Canetti and Rabin, 1993].

Randomized Byzantine consensus algorithms have been around since Ben-Or's and Rabin's seminal papers [Ben-Or, 1983, Rabin, 1983]. Virtually all randomized consensus algorithms are based on a random operation, *tossing a coin*, which returns values 0 or 1 with equal probability.

These algorithms can be divided in two classes depending on how the tossing operation is performed: there are those that use a *local coin* mechanism in each process (starting with Ben-Or's work [Ben-Or, 1983]), and those based on a *shared coin* that gives the same values to all processes (initiated with Rabin's work [Rabin, 1983]).

Typically, local coin algorithms are simpler but terminate in an expected exponential number of communication steps [Ben-Or, 1983, Bracha, 1984], while shared coin algorithms require an additional coin sharing scheme but can terminate in an expected constant number of steps [Rabin, 1983, Toueg, 1984, Ben-Or, 1985, Canetti and Rabin, 1993, Cachin et al., 2000, Friedman et al., 2005]. The original Rabin algorithm required a trusted dealer to distribute key shares before the execution of the algorithm [Rabin, 1983]. This unpractical operation is no longer needed in more recent algorithms [Canetti and Rabin, 1993, Cachin et al., 2000] (the latter from IBM).

Randomized consensus algorithms have often been assumed to be inefficient due to their high expected message and time complexities, so they have remained largely overlooked as a valid solution for the deployment of fault-tolerant distributed systems. Nevertheless, two important points have been chronically ignored. First, consensus algorithms are not usually executed in oblivion, they are run in the context of a higher-level problem (e.g., atomic broadcast) that can provide a friendly environment for the "lucky" event needed for faster termination (e.g., many processes proposing the same value can lead to a quick termination). Second, for the sake of theoretical interest, the proposed adversary models usually assume a strong adversary that completely controls the scheduling of the network and decides which processes receive which messages and in what order. In practice, a real adversary usually does not possess this ability, but if it does, it will probably perform simpler attacks like blocking the communication entirely. Therefore, in practice, the network scheduling can be "nice" and lead to a speedy termination. A recent paper from Lisbon shows that this is true and that these algorithms can be practical [Moniz et al., 2006].

1.3.2. Adding Time to the Model

The notion of *partial synchrony* was introduced by Dwork, Lynch and Stockmeyer in [Dwork et al., 1988]. A partial synchrony model captures the intuition that systems can behave asynchronously (i.e., with variable/unkown processing/communication delays) for some time, but that they eventually stabilize and start to behave (more) synchronously. Therefore, the idea is to let the system be mostly asynchronous but to make assumptions about time properties that are eventually satisfied. Algorithms based on this model are typically guaranteed to terminate only when these time properties are satisfied.

Dwork et al. introduced two partial synchrony models, each one extending the asynchronous model with a time property:

- *M1:* For each execution, there is an unknown bound on the message delivery time Δ, which is always satisfied.
- *M2:* For each execution, there is an unknown global stabilization time GST, such that a known bound on the message delivery time Δ is always satisfied from GST onward.

Chandra and Toueg proposed a third model, which is similar but weaker [Chandra and Toueg, 1996]:

- *M3:* For each execution, there is an unknown global stabilization time GST, such that an unknown bound on the message delivery time Δ is always satisfied from GST onward.

Two Byzantine consensus algorithms, one based on M1 and the other on M2, are presented in the original paper by Dwork et al. [Dwork et al., 1988]. The algorithms are based on a rotating coordinator. Each phase has a coordinator that locks a value and tries to decide on it. The algorithms manage to progress and terminate when the system becomes stable, i.e., when it starts to behave synchronously. There is still no algorithm or proof that M1 allows Byzantine consensus to be solved, although it has been shown to be enough to solve crash-tolerant consensus [Chandra and Toueg, 1996].

The timed asynchronous model enriches the asynchronous system model with hardware clocks that can be used to detect the violation of time bounds [Cristian and Fetzer, 1998]. Cristian and Fetzer have shown that it is possible to solve consensus in this model, although the problem of Byzantine consensus has not been studied [Fetzer and Cristian, 1995].

1.3.3. Augmenting the System Model with an Oracle

The section describes how FLP can be circumvented using oracles. The original idea of circumventing FLP using oracles was introduced by Chandra and Toueg [Chandra and Toueg, 1996]. The oracle in that case is a *failure detector*, i.e., a component that gives hints about which processes are failed or not failed. Remember that FLP derives from the impossibility of distinguishing if a process is faulty or simply very slow. Therefore, intuitively, having a hint about the failure/crash of a process may be enough to circumvent FLP. Notice however that augmenting the system model with a failure detector is equivalent to modifying the time model since (useful) failure detectors cannot be implemented in asynchronous systems. In fact, time assumptions, like those made in partial synchrony models, are usually necessary. The single exception that we are aware of is the requirement for some order pattern in the messages exchanged by the failure models in the solution presented in [Mostefaoui et al., 2003a] (from IRISA).

The following section presents failure detectors and the next one wormholes, which are a more generic concept. Other types of oracles have been presented in the literature, but they have not been used with Byzantine faults. Examples include the Ω detector, which provides hints about who is the leader process [Chandra et al., 1996], and ordering oracles, which provide hints about the order of messages broadcasted [Pedone et al., 2002].

1.3.3.1. Failure Detectors

The original idea of failure detectors was to detect or, more precisely, to suspect the crash of a process. Each process has attached a failure detector module and the set of all these modules formed the failure detector.

Recently, several works applied the idea of Byzantine failure detectors to solve consensus [Malkhi and Reiter, 1997, Kihlstrom et al., 2003, Doudou and Schiper, 1997, Baldoni et al., 2003, Doudou et al., 2002, Friedman et al., 2005]. The main differences in relation to crash failure detectors is that (1) Byzantine failure detectors can neither be made completely independent of the algorithm in which they are used [Doudou et al., 2002], nor (2) detect all Byzantine faults, only certain subsets [Kihlstrom et al., 2003].

Malkhi and Reiter presented a binary consensus algorithm in which the leader waits for a number of proposals from the others, chooses a value to be broadcasted and then waits for enough acknowledgments to decide [Malkhi and Reiter, 1997]. If the leader is suspected by the failure detector, a new one is chosen and the same procedure is applied. The same paper also described a hybrid algorithm combining randomization and an unreliable failure detector. The algorithm by Kihlstrom et al. also solves the same type of consensus but requires weaker communication primitives and uses a failure detector that detects more Byzantine failures, such as invalid and inconsistent messages [Kihlstrom et al., 2003].

Doudou and Schiper presented an algorithm for vector consensus based on a *muteness failure detector*, which detects if a process stops sending messages to another one [Doudou and Schiper, 1997]. This algorithm is also based on a rotating coordinator that proposes an estimate that the others broadcast and accept, if the coordinator is not suspected. This muteness failure detector was used to solve multi-value consensus [Doudou et al., 2002]. Another efficient algorithm based on a muteness failure detector was presented by Friedman et al. [Friedman et al., 2005].

Baldoni et al., from Roma, described a vector consensus algorithm based on two failure detectors [Baldoni et al., 2003]. One failure detector detects if a process stops sending messages (muteness) while the other detects other Byzantine failures. This latter detector is implemented using an interesting solution based on a finite-state automaton that monitors the behavior of the algorithm.

All algorithms based on failure detectors that we are aware of are indulgent, i.e., they satisfy the safety properties of consensus (Validity and Agreement) even if the failure detector does not behave "nicely". Examples of undesirable behavior of a failure detector are not detecting a subset of Byzantine behavior or the muteness of a process.

1.3.3.2. Wormholes

Wormholes are an extension to a system model with stronger properties than the rest of the system, introduced by Lisbon. Wormholes are materialized as enhanced components that provide processes with a means to obtain a few simple privileged functions with "good" properties otherwise not guaranteed by the normal environment [Veríssimo, 2003, Veríssimo, 2006]. For example, a wormhole can provide timely or secure functions in, respectively, asynchronous or Byzantine systems. This contrasts with work on failure detectors, which tries to abstract the minimum requirements on hints about failures to solve consensus. The idea is more generic and has to do with what are the distributed system models that allow to have desirable levels of predictability in systems that are mostly uncertain in terms of properties like time and security [Veríssimo, 2006].

Wormholes are closely related to the notion of *architectural hybridization*, a well-founded way to substantiate the provisioning of those "good" properties on "weak" environments. In the case that we are interested in here, we assume that our system is essentially asynchronous and Byzantine, so when implementing the model we should not simply postulate that parts of it behave in a timely or secure fashion, or these assumptions might naturally fail. Instead, those parts should be

built in a way that our claim is guaranteed with high confidence.

The first paper that presented a consensus algorithm based on a wormhole [Correia et al., 2005] used a specific wormhole, a device called *Trusted Timely Computing Base* (TTCB) [Correia et al., 2002]. Technically, the TTCB is a secure real-time and fail-silent distributed component. Applications implementing the consensus algorithm run in the normal system, i.e., in the asynchronous Byzantine system. However, the TTCB is locally accessible to any process, and at certain points of the algorithm the processes can use it to execute correctly (small) crucial steps. The consensus algorithm relies mostly on a TTCB service called Trusted Block Agreement Service, which essentially makes an agreement on small values proposed by a set of processes. The idea is to use this service to make agreement on the hash of the value proposed by the majority of the processes. Later, a simpler multi-valued consensus algorithm and a vector consensus based on wormholes were also proposed [Neves et al., 2004, Neves et al., 2005]. In this report it is possible to read a recent work on a comparison of randomized consensus protocols with consensus protocols based on wormholes that also use randomized internal protocols [**Correia et al., 2006a**].

1.3.4. Modifying the Problem

This section describes how FLP can be circumvented by weakening the definition of consensus. Currently, we are aware of a single type of algorithm that fits in this category: algorithms based on the condition based approach, introduced by IRISA. These algorithms terminate if the initial values of the processes satisfy certain conditions, but satisfy the safety properties – Validity and Agreement – even if the conditions are not satisfied.

Let us define the *input vector* for an execution of a consensus algorithm as the vector I in which each $I[i]$ is the initial value of process p_i. The condition based approach identifies sets of input vectors for which the consensus algorithm terminates (besides satisfying Validity and Agreement) [Mostefaoui et al., 2003b, Mostefaoui et al., 2004, Friedman et al., 2002]. Conditions on input vectors have been shown to be directly related to error correcting codes. In fact, crash failures correspond to erasure errors in the context of error correcting codes, while Byzantine failures correspond to corruption errors [Friedman et al., 2002].

An argument in favor of this sort of trade-off between Termination and conditions on input vectors is made in [Friedman et al., 2002]. A first reason is that it makes sense to use the approach to efficiently solve consensus problems in which the initial values really satisfy the conditions, but to guarantee safety even if this assumption does not hold. A second reason is that the conditions can serve as a guideline that allows the designer to augment the system (modifying the system model) with the minimum synchrony needed to ensure the solvability of the problem.

The single paper about the condition based approach that we are aware of that deals with Byzantine failures is [Friedman et al., 2002]. This paper presents simple algorithms to solve multi-valued and k-set consensus.

1.4. Performance and Scalability

Byzantine distributed algorithms have been evaluated using several different metrics. Ultimately, the objectives are to understand how an algorithm works and how it behaves in practice:

- How will it *perform*? Or, more precisely, what will be its latency (time needed to run) and throughput (number of executions per unit of time)?
- How will it *scale*, i.e., what is the relation between its performance and the number of processes executing it?
- What will be its *resilience*, i.e., how many faulty processes will it tolerate?

The first two parameters are usually evaluated theoretically in terms of time, message and communication complexities. In asynchronous systems, time complexity is usually measured in terms of the maximum number of *asynchronous*

steps. An asynchronous step involves a process sending a message and receiving one or more messages sent by the other processes. The message complexity is measured by the *number of messages sent* and the communication complexity by the *number of bits sent*. Cryptographic operations often have some impact in the processing time, especially public-key cryptography operations, so the evaluation should also take into account, e.g., the number of signatures made and evaluated. Recently it has been shown that the minimum number of asynchronous steps for Paxos consensus is 2 [Dutta et al., 2005, Martin and Alvisi, 2005].

These metrics are not so simple to assess as it may seem, since they usually depend on the occurrence of faults. Therefore, the evaluations should consider at least two cases: failure-free executions and executions in which the maximum number of processes ($\lfloor (n-1)/3 \rfloor$) is faulty (Byzantine). Other aspects, like the correct processes having the same initial value, can influence the performance evaluation and should also be taken into account.

For probabilistic algorithms, these parameters can only be stated probabilistically so usually the metrics considered are the *expected* number of asynchronous steps, messages sent, bits sent. The literature usually assesses these values in the worst case, i.e., most unfavorable combination of initial values, failures and network scheduling of the messages.

Despite the importance of these theoretical metrics, it has been argued that they may not reflect correctly the behavior of the algorithms in practice [Keidar, 2002]. A recent paper from Lisbon has shown that this is true and that, for instance, randomized binary consensus algorithms that in theory run in high numbers of steps, in practice may execute in only a few communication steps under realistic conditions [Moniz et al., 2006].

The third parameter above, resilience, is probably the simplest since it can be assessed precisely for an algorithm. The optimal resilience for Byzantine consensus in all system models that we are aware of is $n/3$, i.e., less than $n/3$ out of n processes can fail for the algorithm to run correctly [Lamport et al., 1982, Bracha, 1984, Dwork et al., 1988, Correia et al., 2006c].

In relation to resilience, it is important to note that there is no point in making assumptions about the maximum number of processes that can be faulty if there are common modes of failure [Powell, 1992]. For Byzantine failure model, common modes of failure are caused by identical bugs or vulnerabilities in all (or several) processes [Veríssimo et al., 2003]. Independence of failure of processes can be enforced by using diversity of design [Deswarte et al., 1998, Littlewood and Strigini, 2004] (work from LAAS and City).

1.5. Related and Equivalent Problems

In the introduction, we mentioned that there are several distributed systems problems equivalent to consensus. In this section we give more details about this issue.

Given two distributed problems A and B, a *transformation* from A to B is an algorithm that converts any algorithm that solves A into an algorithm that solves B [Hadzilacos and Toueg, 1994]. Problems A and B are said to be *equivalent* if there is a transformation from A to B and a transformation from B to A. Sometimes the equivalence is not generic but assumes some specificity of the system model, like the existence of signatures.

The first equivalences and transformations were established for the crash failure model. In this model, multi-valued consensus has been proved to be equivalent to atomic (or total order) broadcast [Hadzilacos and Toueg, 1994, Chandra and Toueg, 1996]. Transformations from consensus to several problems have been also presented: non-blocking atomic commit [Guerraoui and Schiper, 2001], group membership [Guerraoui and Schiper, 2001], and state machine replication [Schneider, 1990]. Only some of these equivalences/transformations extend to the Byzantine failure model. For instance, non-blocking atomic commit commits a transaction if all resources say 'commit' and aborts it one or more say 'abort'. With Byzantine failure model, a faulty process can simply abort all transactions preventing the system from

working as expected, so clearly there is no transformation from consensus to non-blocking atomic commit.

The equivalence of (Byzantine) atomic broadcast and consensus has been first proved for systems with signatures in [Cachin et al., 2001]. A similar result but without the requirement of signatures has been proved in [Correia et al., 2006c]. Both proofs are independent of the technique used to circumvent FLP. Atomic broadcast, or total order broadcast, is the problem of delivering the same messages in the same order to all processes.

We are not aware of other transformations from Byzantine consensus to other distributed systems problems. However, there is probably a transformation from vector consensus, which has been shown to be equivalent to a variation of multi-valued consensus [Correia et al., 2006c], to group membership. A group membership algorithm makes agreement about a sequence of views, which are numbered events with the identifiers of the members of a group of processes (see, e.g., the survey in [Chockler et al., 2001]). The view can be modified by events like the addition of members to a group, the removal of failed members, and the removal of members by their own initiative. The Byzantine-resilient membership algorithms available give this intuition that a transformation might be defined [Reiter, 1996, Kihlstrom et al., 2001, Correia et al., 2006b].

Several transformations from a variation of (Byzantine) consensus to another have been presented in the literature. Turpin and Coan presented a transformation from binary to multi-valued consensus for synchronous systems [Turpin and Coan, 1984]. Toueg and Cachin et al. presented similar transformations for asynchronous systems, both requiring signatures [Toueg, 1984, Cachin et al., 2001]. Transformations from binary to multi-valued consensus, and from multi-valued to vector consensus, without signatures, were presented in [Correia et al., 2006c].

Conclusion

Consensus is an important problem in distributed systems since it can be considered to be the "greatest common subproblem" of several other problems [Mostefaoui et al., 2000]. This chapter presents a short survey about research on consensus in asynchronous message-passing systems prone to Byzantine faults. Algorithms that solve the several variations of this problem and the equivalent problem of atomic broadcast are currently being used as fundamental building blocks in secure and intrusion-tolerant applications. Therefore, the importance of the consensus problem is undeniable.

2 – On-Line Diagnosis of Transients in Distributed Systems

Introduction

Distributed fault tolerant systems are designed to provide sustained delivery of services despite encountered perturbations. Consequently, two primary objectives are (a) to *consistently identify* a faulty node so as to restrict its effect on the system operations, and (b) to support the process of *recovery* via isolation and reconfiguration of the system resources to sustain ongoing system operations. If these operations can be performed as *on-line* procedures [Walter et al. 1994; Walter et al. 1997], this can help provide a prompt reaction to faults and an effective capability of dynamic resource management, thus mitigating the system susceptibility to subsequent fault occurrences.

Physical faults are distinguished by their nature and duration of impact as being permanent or temporary [Avizienis et al. 2004]. Permanent faults may lead to error whenever the component is activated; the only way to handle such faults is to remove the affected component. Temporary faults can be internal (usually known as intermittent) or external (transient). The former are caused by some internal part deviating from its specified behavior. After their first appearance, they usually exhibit a relatively high occurrence rate and, eventually, tend to become permanent. On the other hand, transient faults, often manifesting the encountered interferences as noise-pulses on the communication channels, cannot be easily traced to a defect in a particular part of the system and, normally, their adverse effects tend to disappear.

In industries like transportation and telecommunications, where operating with permanently faulty modules would carry high risks or costs, it is common that modules, disconnected because considered faulty, are later proved to be free from permanent faults when tested in the repair shop. Therefore, treating transient faults as permanent has a high cost for these industries.

A good discrimination between transient and intermittent/permanent faults solves two important problems: i) prevents the undue removal of nodes affected by transient faults, thus avoiding unnecessary depletion of system resources; and ii) helps to maintain the correct coverage of the system fault hypotheses (i.e., the assumption on the number of faults tolerated by the core system protocols within a given time window) by keeping in operation nodes not permanently faulty. Considering that most perturbations encountered are transient [Siewiorek and Swarz 1998], the issue of "proper" diagnosis of transients is a significant issue of interest.

A related issue is how to overcome the limited coverage of the locally implemented (self-checking) diagnostic tests. In many distributed systems when an error is detected in a node, either through self-checks or by other

nodes, such a node is restarted. A built-in diagnostic test is executed to determine whether the fault is internal, requiring substitution, or external. In the latter case, re-loading the current state from the other nodes is considered sufficient to compensate the transient error. A problem arises as self-check tests generally provide only limited coverage. Thus, permanently faulty nodes can potentially be left operative in the system resulting in problems whose source cannot be easily identified by the system.

A further issue is how to utilize diagnostic information to identify the best recovery action after a fault is detected. In case of transient faults, restarting a node can cause an outage that is longer than that caused by the transient itself. Instead of restarting the node after the first error, it can be better to wait some time before initiating recovery actions, and see if the node is able to locally recover.

This chapter summarizes the state of knowledge in ReSIST on the issue of on-line diagnosis of transients in distributed systems. The above mentioned issues are addressed by the development of a Fault Detection – Isolation – Recovery process (FDIR) to cope specifically with transients. Prior to detailing this state of knowledge, it is first recalled major related approaches in the literature that constituted the background and approaches to the general problem of diagnosis. Also, directions of ongoing and future research for the development of a comprehensive FDIR diagnostic process are indicated at the end.

2.1 Background

A variety of approaches exist that address the FDIR process (or parts of it), and a complete survey is beyond the scope of this document. Therefore, background on diagnosis is here limited to a brief overview of the main existing work in the field. The theoretical problem of diagnosis was set up in the PMC model [Preparata et al. 1967]. The focus of this work, and of many related approaches, was on characterizing system configurations, fault sets and assignments where n active components (units) are able to diagnose, in presence of up to t faulty units, all the faulty units (one-step t- diagnosability) or at least one of them (sequential t-diagnosability). The problem of assignment has been further developed from many viewpoints, trying to define sufficient and necessary conditions when only some combinations of the elements are known.

Many extensions exist to the PMC assumptions; some that are of particular interest with respect to the work done by partners in ReSIST are cited in the following. In [Mallela and Masson 1980] not only permanent but also intermittent faults (i.e., not always detectable) are considered. In this case, the application of tests to faulty units does not necessarily result in the detection of the fault. In some cases, the probability that a test will detect the fault is explicitly taken into account [Blount 1977]. Instead of a PMC multi-processor system with a central diagnostic unit, in [Kuhl and Reddy 1981] a distributed system is assumed, where nodes are connected by links and try to globally assess the health of each other. The possible presence of Byzantine behavior in distributed environment entails the use of agreement techniques [Shin and Ramanathan 1987]. For a discussion on the strong similarities between diagnosis and consensus problems, an excellent survey can be found in [Barborak et al. 1993].

An essential element for the feasibility of on-line diagnosis is the ability of executing diagnostic tests without interrupting the system operation, i.e., without explicit testing capabilities. A well known solution is the comparison approach, introduced in [Malek 1980] and characterized in [Sengupta and Dahbura 1992], where the replication of tasks in the system is used for diagnostic purposes. Pairs of nodes execute the same task and the outcomes are compared by other nodes. The model has been optimized and characterized for distributed systems

with broadcast capability in [Blough and Brown 1999]. In the PMC approach and its derivatives, the diagnosability of the system is mainly characterized by the configuration of testing assignments, i.e., which nodes apply testing stimuli to which other nodes. In the comparison approach, the diagnosability depends on how tasks are redundantly allocated to nodes, such that the results produced by each node can be compared. If nodes are assumed to be fail-silent, group membership protocols can be used to perform FDIR operations. This ensures that all nodes have a consistent view of the distributed state of the system, i.e., all correct nodes have received the same set of messages. The first definition of the group membership problem and a first solution of the problem in asynchronous systems appeared in the ISIS project [3]. One of the first approaches to group membership for synchronous systems was proposed in [Cristian 1991], while a membership protocol is intertwined with clock synchronization in synchronous systems in the TTP protocol [Kopetz & Grunsteidl 1994].

The importance of distinguishing transient faults, so that they can be dealt with specifically, is testified by the wide range of solutions proposed, although with reference to specific systems. One commonly used method, for example, in several IBM mainframes [Spainhower 1992], is to count the number of error events: too many events in a given time frame would signal that the component needs to be removed. In TMR MODIAC, the architecture proposed in [Mongardi 1993], two failures experienced in two consecutive operating cycles by the same hardware component that is part of a redundant structure make the other redundant components consider it as definitively faulty. Another architecture using similar mechanisms, designed for distributed ultra-dependable control systems, is described in [Lala and Alger 1988]. In this case, a combination of diversified design, temporal redundancy and comparison schema is used to obtain a detailed determination of the nature of faults. Counting mechanisms are also used to solve the so called 2-2 splits, i.e., to determine the correct value among four proposals in a QMR system when there is a tie. In [Agrawal 1988], a list of "suspect" processors is generated during the redundant executions; a few schemes are then suggested for processing this list, e.g., assigning weights to processors that participate in the execution of a job and fail to produce a matching result and taking down for diagnostics those whose weight exceeds a certain threshold. Other approaches do, instead, concentrate on off-line analysis of system error logs, and therefore are not applicable on-line. In [Lin and Siewiorek 1990], some heuristics, collectively named Dispersion Frame Technique, for fault diagnosis and failure prediction are developed and applied to system error logs taken from a large Unix-based file system. The heuristics are based on the inter-arrival patterns of the failures (which may be time-varying). For example, there is the 2-in-1 rule, which warns when the time of inter-arrival of two failures is less than one hour, and the 4-in-1 rule, which fires when four failures occur within a 24-hour period. In [Iyer et al 1990], an error rate is used to build up error groups and simple probabilistic techniques are then recursively applied to discern similarities (correlations) which may point to common causes (permanent faults) of a possibly large set of errors.

2.2 State of Knowledge in ReSIST

The knowledge on diagnosis of transient faults mainly embraces two directions: i) distributed diagnostic algorithms for synchronous systems, and ii) "count-and-threshold" schemes and architectural framework for handling multiple classes of faults in components of the shelf (COTS) and legacy-based applications.

Distributed diagnostic algorithms for synchronous systems. Previous work [Walter et al. 1997] introduced a family of distributed diagnostic algorithms for synchronous systems based on the Customizable Fault/Error Model [Walter et al. 1994], where the fault assumptions can be adapted to meet the fault hypothesis of the core fault-tolerant protocols of the system (e.g., clock synchronization). One advantage of the approach is that

diagnosis is not considered as an off-line and fault-free procedure, but as an on-line core fault-tolerant mechanism fully integrated within the system fault tolerance strategy. Instead of executing dedicated, performance-impacting tests like in the PMC model, or constraining the allocation of application-level tasks to nodes like in the comparison approach, it uses error detection information derived by the execution of fundamental system-level activities, like message delivery and clock synchronization, to diagnose the system. In fact, this approach can be seen as complementary to the graph based, application-level approaches. Diagnosis is seen as a special case of consensus under the Hybrid Fault Model. To reach consensus in presence of one malicious asymmetric fault, a two round protocol has to be executed. In distributed diagnosis, nodes exchange operational messages and, by examining them, produce a local syndrome assessing the health of the other nodes. This is analogous to the first round of a consensus algorithm. The local syndromes are then exchanged during a subsequent dissemination round. Consistent health vectors are computed using the same hybrid majority function of the hybrid consensus protocol OMH [Lincoln and Rushby 1993]. The health vectors can then be used to consistently isolate faulty nodes on-line. This reduces the diagnostic latency and bandwidth requirements with respect to other approaches where a local syndrome is first built and then a separate consensus protocol is invoked. The need of recording the duration and recurrence of faults and to assign different severity levels to different syndromes has been pointed out. If transient faults are assumed to be instantaneous, the permanence of the generated errors in a node is bounded and known a priori (e.g., if a periodic memory scrubbing task is executed on the background) a penalty value will record the state of the nodes throughout multiple diagnostic frames to assess the health of the node against such bounds. Most of the previous diagnostic services provide snapshot level information about a single manifestation of faults. Without accumulating penalty information over time, a diagnostic protocol can only declare faulty nodes as either always permanent or always transient faulty. An evaluation of the effect on system reliability of these two different policies on the SPIDER system, which runs a diagnostic algorithm derived from [Walter et al 1997], was conducted in [Latronico et al. 2004]. The intuitive result obtained was that optimal reliability is attained if one does not assume all detected faults to be permanent or all to be transients.

"Count-and-threshold" schemes. In practice, nodes oscillate between faulty and correct behavior. To handle this, a family of mechanisms collectively called "count-and-threshold" schemes has been established in [Bondavalli et al. 1997; Bondavalli et al. 2000]. The idea is that components should be kept in the system until the benefit of keeping the faulty component on-line is offset by the greater probability of multiple (hence, catastrophic) faults. Apart from the class of permanent faults, when the component always fails every time it is activated, a basic discrimination is done in the context of temporary faults spanning intermittent and transient faults: the first are due to faults internal to the component, and show a high occurrence rate which eventually might turn them to permanent faults; the second are due to reasons external to the component, generally have an uncorrelated occurrence rate, and should not determine the exclusion of the component. Therefore, after detecting a mis-behavior in the component under diagnosis, it is advocated to wait and see if the error reappears before isolating the component. The goal was to develop mechanisms characterised above all by: a) tunability through internal parameters, to warrant wide adaptability to a variety of system requirements; b) generality with respect to the system in which they are intended to operate, to ensure wide applicability; c) simplicity of operation to allow high analysability through analytical models and to be implementable as small, low-overhead and low-cost modules, suitable especially for embedded, real-time, dependable systems.

A generic class of low-overhead count-and-threshold mechanisms, called α-count, has been defined in [Bondavalli et al. 1997]. An error counter is associated to each component, which is incremented when the component fails and decremented when it delivers a correct service. The behavior of a component is assumed

evaluated by an error detection subsystem, which periodically issues error detection signals. At period *L*, the error detection signal $J_i^{(L)}$ is filtered using the function α, which can be formulated in multiple ways, for example as:

$$\alpha_i^{(L)} = \begin{cases} \alpha_i^{(L-1)} \cdot K & if \quad J_i^{(L)} = 0 \\ \alpha_i^{(L-1)} + 1 & if \quad J_i^{(L)} = 1 \end{cases} \qquad 0 \le K \le 1$$

$$\alpha_i^{(0)} = 0$$

where $J_i^{(L)} = 0$, means that no error has been detected ($J^{(L)} = 1$ means that an error has been detected), and K represents the ratio in which α is decreased after a time step without error signals, thus setting the time window where memory of previous errors is retained. When the value of $\alpha_i^{(L)}$ exceeds a given threshold α_T, the component is diagnosed as affected by a permanent or an intermittent fault. The choice of the tunable parameters α_T and *K*, so as to properly balance between the two conflicting requirements to signal as quickly as possible all components affected by permanent or intermittent faults and at the same time to avoid signaling as faulty components that are affected by transients, is based on extensively stochastic evaluation using SAN models [Meyer et al. 1980].

This model was then extended in [Grandoni et al. 1998] to include double threshold mechanisms, where an element is temporarily excluded after a first threshold is exceeded, but still given an opportunity to be reintegrated. A thorough elaboration of the α-count mechanism including complex error distributions has been presented in [Bondavalli et al. 2000].

With respect to other count-and-threshold schemes reviewed in the Background section, where the tuning of the threshold has been typically performed by expertise and trial-and-error, the designer is provided with well-defined tools, to exert such actions in a systematic, predictable and repeatable way. Also, when given suitable parameters, their functionalities may easily be obtained by an α-count instance, e.g. the approach proposed in [Mongardi 1993]. The function illustrated previously has been also implemented in the GUARDS architecture [Powell and al. 1999] for distributed diagnosis: a binary accusation on the node health is shared, using consensus, voted, and given as the input for the count-and-threshold. In [Grandoni et al. 2001], a comprehensive analysis of a few integrated fault tolerance organizations obtained coupling the diagnostic mechanism α-count with a variety of error processing techniques in a multiprocessor environment has been carried out. The goal was to gain insights in the effects on the overall system dependability induced by such combined usage.

In [Romano et al. 2002] issues have been addressed, related to goals and constraints of a diagnostic subsystem based on the concept of threshold, which must be able to: 1) understand the nature of errors occurring in the system, 2) judge whether and when some action is necessary, and 3) trigger the recovery/reconfiguration/repair mechanisms to perform the adequate actions. Further, in [Bondavalli et al. 2004] a methodology and an architectural framework for handling multiple classes of faults (namely, hardware-induced software errors in the application, process and/or host crashes or hangs, and errors in the persistent system stable storage) in a COTS and legacy-based application have been defined. The framework includes a fault tolerance manager that collects error reports based on which it chooses and performs a fault recovery strategy, depending upon the perceived severity of the fault. The methodology and the framework have been applied to a case study system and a thorough performability analysis has been conducted via combined use of direct measurements and analytical

modeling. Fault injection on a real system prototype was used to derive realistic fault models (by tracking error propagation through individual system layers), and to extract relevant system parameter values (which were used to populate an analytical model of the system). Although heavily influenced by the specific requirements of the considered target application, numerical results indicate that a reasonably configured threshold-based diagnostic system associated with properly chosen recovery actions can improve the dependability of COTS and legacy-based applications significantly.

Another direction of research has addressed a rigorous mathematical formulation of diagnosis based on Bayesian inference [Pizza et al. 1998]. Bayesian inference provides a standard procedure for an observer who needs to update the probability of a conjecture on the basis of new observations. Therefore, after a new observation is provided by the error detection subsystem, the on-line diagnosis procedure produces an updated probability of the module being affected by a permanent fault. This leads to an optimal diagnosis algorithm, in the sense that fault treatment decisions based on its results would yield the best utility among all alternative decision algorithms using the same information. This higher accuracy with respect to simple heuristics comes at the cost of higher computational complexity.

2.3 On-going and Future Directions

A few indications on current and future research directions on diagnosis carried on and/or planned at the Univ. of Darmstadt, ISTI-CNR in Pisa and Univ. of Florence are given in the following.

One direction of current research is to introduce and examine a generic on-line FDIR framework, able to generalize the previously proposed distributed diagnosis protocols and to enhance them with count-and-threshold techniques in order to effectively handle transient faults. Particularly, the aim is to (a) determine and establish the effect of the duration and recurrence of faults on the effectiveness of the on-line diagnosis protocols, and (b) ascertain the sensitivity and the tradeoffs of choices of some selected FDIR design parameters in determining the correctness and completeness of the FDIR protocols and in improving system reliability.

The system parameters that can influence the effectiveness of the FDIR process are taken into consideration. For example, in a synchronous distributed system, every node exchanges data at an epoch, also known as the *communication frame*. As error detection also takes place over each frame, it can also be considered as a *diagnostic frame*. Over each communication frame, the system health is “sampled” by the different nodes, which compose their local syndrome accordingly and execute the diagnostic protocol. In this context, the assumption that errors manifest only over a single frame, as characterized in previous analyses, is not adequate. The length of the diagnostic frame is a parameter that, together with other count-and-threshold parameters, will influence the likelihood by which a node is excluded from system operation. In fact if the frame is too short, a transient fault may be perceived as permanent, and consequently lead to pessimistic resource isolation. This can be particularly problematic for long-time missions. On the other hand, if the frame rate is too large, one would expect large diagnostic latencies in the system. These might be undesirable for critical applications with short mission times and requirements of rapid response to perturbations. Moreover, this increases the probability of coincident errors within the same frame.

The intended contribution is to study the choice of the diagnostic frame rate and other system design parameters, within an architectural context to highlight the correctness/completeness and reliability tradeoffs. Namely, the following main design parameters are considered:

- Diagnostic frame rate: the rate at which nodes exchange diagnostic data, aggregate it, and consequently update penalties and rewards accordingly,
- Penalty counter threshold values: number of temporally correlated diagnostic frames after which an erroneous node gets isolated, and
- Reward counter threshold values: number of diagnostic frames after which a node (previously suspected as erroneous) displaying correct behavior gets re-admitted into the system as a "good" node.

Another direction focuses on a formalization of the diagnosis process, addressing the whole chain constituted by the monitored component, the deviation detector and the state diagnosis. Hidden Markov Models appear well suited to represent problems where the internal state of a certain entity is not known and can only be inferred from external observations of what this entity emits. The goal is to develop high accuracy diagnosis processes based on probabilistic information rather than on merely intuitive criteria-driven heuristics. Because of its high generality and accuracy, the proposed approach is expected to be usefully employed: i) to evaluate the accuracy of low-cost on-line processes to be adopted as appropriate and effective diagnostic means in real system applications; ii) for those diagnostic mechanisms equipped with internal tunable parameters, to assist the choice of the most appropriate parameter setting to enhance effectiveness of diagnosis; and iii) to allow direct comparison of alternative solutions.

In addition, diagnosis in mobile distributed environments is planned to be investigated, to account for both volunteer and/or accidental disconnections of system components.

Other uses of count-and-threshold mechanisms taken as a system building-block are also interesting to be explored. For example, consider an N-modular redundant fault-tolerant structure, where instances of α-count are employed in each module for its original goal. A higher level diagnosis layer could be added, which would monitor the α-count values of all modules by looking for correlated errors: a common pattern of rising counts on several counters might, for instance, be an alarming symptom.

Finally, the applicability of threshold-based diagnostic techniques, originally tailored to physical faults, to wider categories of malfunctions (like design faults, human faults and malicious attacks) appears to be another interesting research area to explore.

3– A Survey of Cooperative Backup Mechanisms

3.1. Introduction and Motivations

Storage capacity, like computing power, follows its Moore's law and grows dramatically, for instance disk density grows at an impressive annual rate of 100% [Growchowski, 1998]. At the same time, this new storage capacity is consumed by the production of new data. Consequently, the need for backup capacity increases but so does the space available for backup.

Our concern here is on *cooperative* backup services, in which resources belonging to multiple users are pooled as a distributed storage system for backing-up each others' data. Such a cooperative backup service must be distinguished from other forms of distributed storage such as file sharing systems or general file systems.

A file system can be defined as a support for the storage of data on non-volatile medium, typically a hard disk. Data is stored on files that encompass both the data and its associated metadata (name and other attributes such as date of modification, etc.). Usually a file system also provides a directory service on top of a flat file service. The flat file service maps the data with unique file identifiers and stores the data on the storage device. The organization of the data can have several forms, either unstructured or structured as a sequence or hierarchy of records. The directory service maps the metadata to the files' unique identifiers. Typically this mapping is stored on the storage device using the flat-file service itself. There is a number of distributed file systems, the most famous being NFS (Network File System) that is used on many UNIX networks.

File sharing[1] emerged relatively recently as an Internet application and greatly participated to the definition of a new type of distributed system: peer-to-peer systems. The goal of a file sharing system is to enable multiple users to access files. Classical and well-known file-sharing systems are Napster, eDonkey, Gnutella, Kazaa, MojoNation, BitTorrent, etc.

The role of a backup system is to tolerate faults affecting some storage device, be it local or distant, centralized or shared. The type of faults considered here can be permanent failures of the storage devices (e.g., crashed disks), or even localized catastrophes like a fire incident in an office when the backup media are taken off-site.

Given these definitions, one can see that there are quite some differences in the specification of these services even if there are also some strong similarities (primary goal is storage, the concept of file, etc.). If one wants more specific differences, one can consider the following properties: multiplicity of the data readers/writers and mutability of the content. The following table presents these characteristics.

Among the afore-mentioned file-sharing systems, we can identify some features that are centralized, distributed or coop-

[1] We differentiate here between peer-to-peer file sharing systems and distributed file systems that can also be seen as a way to share files.

	File type	Writer multiplicity	Reader multiplicity
File sharing systems	Static	Single	Multiple
File backup systems	Static	Single	Single
General file systems	Dynamic	Multiple	Multiple

Figure 3.1: Typical characteristics distinguishing various distributed storage

erative. Similarly to file systems, file-sharing systems have two main functions: first they have to manage the actual distribution of the shared files, and second they have to organize the lookup, i.e. manage a global directory for users to search for given files. The lookup service of Napster [Napster,] is centralized, the one of eDonkey [Heckmann and Bock, 2002] is distributed among a set of servers and finally, the one of Kademlia [Maymounkov and Mazières, 2002] is cooperatively realized between the participating nodes.

It is clear that file-sharing systems are different from backup systems. These systems do not guarantee long-term survivability of files, especially those files that few users are interested in storing or accessing. Thus, they could hardly be used for the purpose of backup. One could argue that regular file systems could easily be used for such a purpose since long-term survivability and fault-tolerance are very important concerns for this type of service. For instance, a simple solution to back-up on top of file systems would be to use Unix-like facilities, e.g., tar, CVS, etc. However, the specification and the semantics of file systems being so much broader than those of backup systems (multi-writers vs. single-writers; read-write vs. write-once/read-many), it would be unfair to compare these two types of system.

In this chapter we will survey only backup services that use a cooperative approach. We will be concerned both with cooperation between resources pooled directly over a fixed infrastructure such as Internet and with mobile resources that are pooled opportunistically according to locality. This latter class of cooperative backup service is motivated by the observation that users of mobile devices (laptops, personal digital assistants, mobile phones, etc.) often perform a backup of their data only when they have access to their main desktop machine, by synchronizing the two machines. Typically, the first generation of personal digital assistants (PDAs) had only a short distance communication means, generally a serial or infrared device. This meant that the user had to be physically close to the machine on which she performed the synchronization. Nowadays, portable devices usually have several communication interfaces (for instance WiFi, Bluetooth, etc.). When a network infrastructure is available in their vicinity, for instance WiFi access points, those devices could connect to their main desktop machine in order to back-up their data. However, in practice, this is rarely the case, for several reasons :

- the desktop machine must be running, connected to the Internet and available;
- access to a network infrastructure using wireless communications is still rare and expensive, and it can take a while before a device is able to connect to the Internet;
- finally, to our knowledge, the software able to perform such a backup on a remote desktop are still rare.

Another solution to mobile device backup is the use of a trusted third party that guarantees its backup servers' availability. Several commercial offerings enable their customers to back-up their data on a limited storage capacity for a yearly fee.

However, the growth rate of this kind of wireless communicating devices is such that a cooperative approach to back-up is becoming feasible, based on peer-to-peer interactions. These wireless interactions are frequent but ephemeral. Nevertheless, they could be leveraged: whenever two devices meet, a backup service can automatically initiate a request for a partial data backup. As a counterpart, it has to offer the same service to the community, i.e. to form a *cooperative* backup system.

In Section 2 of this survey, we discuss the features that characterize cooperative backup systems. Several existing systems are then described and compared briefly in Section 3. In Section 4, a more in-depth analysis is given with respect to storage

management issues. Then Section 5 focuses on the dependability techniques used in these systems. Finally, Section 6 concludes the survey by a summary and sketches some directions for future work.

3.2. Characterization of Cooperative Backup Systems

In [Chervenak et al., 1998], Chervenak *et al.* described a number of features for the characterization of backup systems: full vs. incremental, file-based vs. device-based, on-line or not, snapshots and copy-on-write, concurrent backups, compression, file restoration, tape management, and finally disaster recovery. On the one hand, most of these features remain of interest in our context. However, on the other hand, some characteristics concerning dependability and the cooperative nature of the considered systems were not addressed: privacy, denial-of-service resilience, trustworthiness management, etc. In their survey, Chervenak et al. characterized backup systems using a set of properties. It must be noted that their focus was on centralized or server-based, system-wide, backup systems, i.e., the target was large multi-user client systems. The type of backup services we are interested in targets personal computers and thus, some of the properties defined by Chervenak et al. are not relevant. We thus propose another characterization of backup systems based on a set of functionalities and of dependability issues, as described in the following sections.

3.2.1. Functionalities of Backup Systems

3.2.1.1. Full vs. Incremental Backups

The simplest solution to back-up a file-system is to copy its entire content to a backup device. The resulting archive is called a full backup or a snapshot of the source data. Both a full file-system and a single lost file can be easily restored from such a full backup. However, this solution has usually two major drawbacks: since it concerns the entire content of the file-system, it is slow and requires a large amount of backup storage space. We will come back onto this issue of resource usage in the next subsection.

As a solution to this, incremental backup schemes can be used. They copy only the data that have been created (added) or modified since a previous backup. To restore the latest revision of a given file, the first full backup along with all the subsequent incremental backups must be read, which can be extremely slow. For this reason, typical incremental schemes perform occasional full backups supplemented with frequent incremental backups. Several variations of incremental backup techniques exist: incremental-only, full+incremental or even continuous incremental where newly created or modified data is backed-up within a few minutes, as it is created or modified, instead of once a day, typically, with traditional incremental backups.

3.2.1.2. Resource Usage

To reduce both storage requirements and network bandwidth usage, backup systems can use classic compression techniques. This can be done at the client-side or at the server-side. Recently, other techniques emerged to reduce the storage space required to back-up several file systems. An example is single-instance storage [Quinlan and Dorward, 2002] which aims to store once every block of data even if it is present on the file systems of several users, or if there are multiple instances in a single file-system.

3.2.1.3. Performance

Backup system performance is measured in terms of the backup time as well as the restoration time. The performance of the backup process is impacted by factors such as incremental backups, compression, etc. Several parameters and features have a dramatic effect on the actual efficiency of the restoration operations. For instance, restoration will be slower in an

incremental backup system, which must begin with the last full backup and apply changes from subsequent incremental backups. An additional concern when restoring an entire file system is that files deleted since the previous backup will reappear in the restored file system. More generally, the unbacked-up changes on the metadata, the structure and the hierarchy of the file system cannot be restored.

It is important to note that scalability is a very important issue when dealing with cooperative systems. The number of nodes participating in the cooperative system can be potentially very large and this raises a number of issues and problems to be dealt with. An important metric for cooperative backup system is thus the number of nodes that the system can accomodate.

3.2.1.4. On-line Backups

While many backup systems require that the file system (or the files) remain quiescent during a backup operation, on-line or active backup systems allow users to continue accessing files during backup. On-line backup systems offer higher availability at the price of introducing consistency problems.

The most serious problems occur when directories are moved during a backup operation, changing the file system hierarchy. Other problems include file transformations, deletions and modifications during backup. In essence, any type of write operation on the files or on the file-system hierarchy during a backup is a potential source of problems. There are several possible strategies to overcome these problems:

1. *Locking* limits the availability of the system by forbidding write accesses while backing-up.
2. *Modification detection* is used to reschedule a backup of the modified structures/files.
3. *Snapshots*, i.e., frozen, read-only copies of the current state of the file-system offer another alternative for online backup. The contents of a snapshot may be backed-up without danger of the file system being modified from subsequent accesses. The system can maintain a number of snapshots, providing access to earlier versions of files and directories.
4. A *copy-on-write* scheme is often used along with snapshots. Once a snapshot has been created, any subsequent modifications to files or directories are applied to newly created copies of the original data. Blocks or file segments are copied only if they are modified, which conserves disk space.

3.2.2. Dependability and Other Orthogonal Issues of Cooperative Backup Systems

In the previous section we presented several functional aspects of backup systems. We now look at the orthogonal issues raised by a cooperative approach to backup: integrity and consistency, confidentiality and privacy, availability, synergy and trust.

3.2.2.1. Integrity and Consistency

A backup service has to guarantee the integrity and consistency of restored data.

Any corruption of the backed up data, be it intentional or not (for instance due to a software or hardware fault on the system actually providing the storage), must be detected by its owner during restoration. Network protocols as well as storage devices commonly use error-detecting or correcting codes to tolerate software and hardware faults. However, to be resilient to intentional corruption, the data owner must be assured that the data restored is the same that which was backed up.

Consistency is an issue when multiple items of data must ensure some common semantics. In such cases, special care must be taken to manage dependencies.

3.2.2.2. Confidentiality and Privacy

The entities participating in a cooperative backup service store some of their data on the resources of other participants with whom they have no a priori trust relationship. The data backed up may be private and thus should not be readable by any participating entity other than its owner, i.e., the service has to ensure the confidentiality of the data. Furthermore, a cooperative backup service must protect the privacy of its users. For instance it should not deliver any information concerning the past or present location of its users.

3.2.2.3. Availability

In a backup system, availability has several dimensions. First, the primary goal of a backup system is to guarantee the long-term availability of the data being backed up. In some sense it is the functional objective of the system. Second, to be useful, the backup system itself must be available, i.e., it must be resilient to failures (hardware, software, interaction, etc.). In the context of a cooperative approach to backup, additional concerns arise, especially with respect to malicious or selfish denial-of-service attacks.

3.2.2.4. Synergy

Synergy is the desired positive effect of cooperation, i.e., that the accrued benefits are greater than the sum of the benefits that could be achieved without cooperation. However synergy can only be achieved if nodes do indeed cooperate rather than pursuing some individual short-term strategy, i.e. being rationale.

Hardin introduced the tragedy of the commons concept in 1968 [Hardin, 1968] to formalize the fact that a shared resource (a common) is prone to exhaustion if the resource consumers use short-term strategy to maximize their benefit out of the resource. Consider the simple example of a grass field shared by 25 farmers. The field can normally accommodate 50 cattle. However, each rational farmer is tempted to maximize his outcome by having more than 2 cattle feeding from the shared field. This short-sighted strategy eventually leads the field to exhaustion through over-consumption. A generalized form of the problem is when a resource market has externalities, i.e., when the cost of using a resource is shared among its consumers.

The tragedy of the commons has recently been extended to the digital world, or “Infosphere”, leading to the tragedy of the digital commons [Greco and Floridi, 2003]. It is relatively intuitive, for instance, to regard the Internet as a shared resource. Each user uses his connection without paying much attention to the presence of other users and to the fact that they share a common bandwidth. Each user thus uses his available bandwidth up to its maximum, only being reminded that other users also consume this resource when there is a network congestion.

One way to ensure synergy in a cooperative backup system is to enforce the “fair exchange” property: if one contributes up to 5 MiB to the system, one wants to get serviced up to 5 MiB too. Reciprocally, it is desirable that a device getting serviced for such an amount of resources offers an equivalent amount to the cooperative service.

3.2.2.5. Trust Management

An important aspect of many cooperative systems is that each node has to interact with unknown nodes with which it does not have a pre-existing trust relationships. The implementation of a cooperative backup service between nodes with no prior trust relationship is far from trivial since new threats must be considered:

1. selfish devices may refuse to cooperate;
2. backup repository devices may themselves fail or attack the confidentiality or integrity of the backup data;
3. rogue devices may seek to deny service to peer devices by flooding them with fake backup requests; etc.

There is thus a need for trust management mechanisms to support cooperative services between mutually suspicious devices.

3.3. Existing Cooperative Backup Systems

In this section, we first give a preliminary description and analysis of various systems devoted to cooperative backup. Cooperative backup are inspired by both cooperative file systems and file sharing systems. Most are concerned with the problem of cooperative backup for fixed nodes with a permanent Internet connection. To our knowledge, there are only two projects looking at backup for portable devices with only intermittent access to the Internet: *FlashBack* [Loo et al., 2003] and *MoSAIC* [Killijian et al., 2004].

3.3.1. Peer-to-peer Backup Systems for WANs/LANs

The earliest work describing a backup system between peers is the one of Elnikety *et al.* [Elnikety et al., 2002], which we will henceforth refer to as *CBS*. Regarding the functions of a backup system (resource localization, data redundancy, data restoration), this system is quite simple. First, a centralized server is used to find partners. Second, incremental backup, resource preservation, performance optimization were not addressed. However, various types of attacks against the system are described. We will come back to this later.

The *Pastiche* [Cox and Noble, 2002] system and its follow-up *Samsara* [Cox and Noble, 2003], are more complete. The resource discovery, storage, data localization mechanisms that are proposed are totally decentralized. Each newcomer chooses a set of partners based on various criteria, such as communication latency, and then deals directly with them. There are mechanisms to minimize the amount of data exchange during subsequent backups. Samsara also tries to deal with the fair exchange problem and to be resilient to denial-of-service attacks.

Other projects try to solve some limitations of the *Pastiche/Samsara* systems, or to propose some simpler alternatives. This is the case for *Venti-DHash* [Sit et al., 2003] for instance, based on the *Venti* archival system [Quinlan and Dorward, 2002] of the *Plan 9* operating system. Whereas *Pastiche* selects at startup a limited set of partners, *Venti-DHash* uses a completely distributed storage among all the participants, as in a peer-to-peer file sharing system.

PeerStore [Landers et al., 2004] uses a hybrid approach to data localization and storage where each participant deals in priority with a selection of partners (like *Pastiche*). Additionally, it is able to perform incremental backup for only new or recently modified data. Finally, *pStore* [Batten et al., 2001] and ABS [Cooley et al., 2004], which are inspired by versioning systems, propose a better resource usage.

Based on the observations that worms, viruses and the like can only attack machines running a given set of programs, the *Phoenix* system [Junqueira et al., 2003] focuses on techniques favoring diversity among software installations when backing up a machine (e.g., trying to not backup a machine that runs a given vulnerable web server on a machine that runs the same web server). The main added value is here in the partnership selection.

In [Cooper and Garcia-Molina, 2002], the authors focus on the specific issue of resource allocation in a cooperative backup system through an auction mechanism called bid trading. A local site wishing to make a backup announces how much remote space is needed, and accepts bids for how much of its own space the local site must “pay” to acquire that remote space.

In [Hsu et al., 2004], the authors implement a distributed backup system, called *DIBS*, for local area networks where nodes are assumed to be trusted: the system ensures only privacy of the backed up data but does not consider malicious attacks against the service. Since *DIBS* targets LANs, all the participating nodes are known *a priori*, partnerships do not evolve, and no trust management is needed.

3.3.2. Cooperative File Systems

As mentioned earlier, a backup system (static data files, single writer) can be implemented on top of any file system (mutable data files, multi-writer). There exist a number of peer-to-peer general file systems such as Ivy [Muthitacharoen et al., 2002], OceanStore [Kubiatowicz et al., 2000], InterMemory [Goldberg and Yianilos, 1998], Us [Randriamaro et al., 2006], etc. We briefly present here two of them for the sake of the comparison although they are outside the scope of this survey.

Us [Randriamaro et al., 2006] provides a virtual hard drive: using a peer-to-peer architecture, it offers a read-only data block storage interface. On top of *Us*, *UsFs* builds a virtual file system interface able to provide a cooperative backup service. However, as a full-blown filesystem, *UsFs* provides more facilities than a simple backup service. In particular, it must manage concurrent write access, which is much more difficult to implement in an efficient way.

OceanStore [Kubiatowicz et al., 2000] is a large project where data is stored on a set of untrusted cooperative servers which are supposed to have a long survival time and high speed connection. In this sense we consider it as a distributed file system using a super-peers approach rather than a purely cooperative system. The notion of super-peers relates to the fact that peers are specifically configured as file servers (with large amount of storage) that can cooperate to provide a resilient service to non-peer clients.

3.3.3. Mobile Systems

The *FlashBack* [Loo et al., 2003] cooperative backup system targets the backup of mobile devices in a Personal Area Network (PAN). The nature of a PAN simplifies several issues. First, the partnerships can be defined statically as the membership in the network changes rarely: the devices taking part in the network are those that the users wear or carry. Second, all the devices participating in the cooperative backup know each other. They can be initialized altogether at configuration time so there is no problem of handling dynamic trust between them. For instance, they may share a cryptographic key.

MoSAIC [Killijian et al., 2004] is a cooperative backup system for communicating mobile devices. Mobility introduces a number of challenges to the cooperative backup problem. In the context of mobile devices interacting spontaneously, connections are by definition short-lived, unpredictable, and very variable in bandwidth and reliability. Worse than that, a pair of peers may spontaneously encounter and start exchanging data at one point in time and then never meet again. Unlike *FlashBack*, the service has to be functional even in the presence of mutually suspicious device users.

3.4. Storage Management

In this section, we present two aspects that are specific to peer-to-peer data storage systems: mechanisms for storage allocation, and techniques for efficient usage of resources.

3.4.1. Storage Allocation

Among the systems studied, one can identify three distinct approaches to the dissemination of the data blocks to be stored:

- the storage can be allocated to specific sets of participants or partners;
- the storage can be allocated across all participants using a distributed hash table (DHT), which has the property of ensuring an homogeneous block distribution;
- the storage can be allocated opportunistically among neighbors met when storage of a block is needed.

In the first case, the relationships between the partners are relatively simple: each participant chooses a set of partners at start-up. Then, for each backup, it directly sends the blocks to be saved to its partners. In Pastiche and in CBS, each

participant chooses a set of partners that will remain almost static. Finally, the FlashBack devices, in a PAN, choose their set of partners according to the amount of time spent in each other's vicinity.

The second approach is based on a technique that is fundamental to peer-to-peer file sharing systems, *virtual networks* or *overlay networks* [Lua et al., 2005], which use the notion of distributed hash tables (DHT) for allocating data blocks. Each node of the network is responsible for the storage of the blocks whose identifier is close (numerically) to its own identifier. The advantage of using a DHT is that the blocks are homogeneously distributed over the network if their identifiers are numerically homogeneously distributed. Both Venti-DHash and pStore use DHTs to store backup data blocks. However, there are two disadvantages to this approach:

- The cost of migration of the data blocks when a node enters or leaves the system can be high (bandwidth-wise) [Landers et al., 2004]. Because of the mathematical mapping between data blocks and node identifiers, no exception is acceptable: when a node enters the virtual network, it must obtain and store all the blocks for which it is mathematically responsible; respectively, when a node leaves the network, the various blocks it was responsible for must be re-distributed using the DHT mechanism.
- A DHT automatically distributes the data blocks homogeneously among the participants, independently of how much storage space each node consumes. Consequently, using a DHT makes it impossible for a system to ensure fair exchange.

For these reasons, PeerStore proposes a hybrid approach where the data blocks are directly exchanged between partners and where the blocks' meta-information (the mapping between a block ID and the node that stores it) are stored using a DHT. For optimization, the set of partners is sometimes extended at runtime to nodes that were not originally in the partnership: when a node needs to store a block, it looks into the DHT to see if the block is already stored (single-instance storage). When that is the case, the block is not stored twice. Instead, the node that already stores it becomes a new partner for the node owning it.

The third approach is very different. The MoSAIC system targets mobile devices, so partnerships cannot be established a priori[2], but have to be defined during the backup itself, opportunistically. MoSAIC is an active backup system - whenever some critical data is modified, the modified blocks need to be backed-up. This is done towards the devices that the user will meet along its way. In this case, the partnership is determined at runtime and is a function of the mobility patterns of the participating nodes.

3.4.2. Storage Optimization

The amount of storage necessary to store backed-up data can be optimized by applying compression techniques. Compression is worthwhile even if data is ultimately backed-up in redundant copies (to ensure backup availability). Indeed, the redundancy that is eliminated using compression techniques can be seen as "accidental", e.g., due to overly prolix data formats. Thus, compression can be thought of as a way to *normalize data entropy* before adding new redundancy. In other words, going through the compression step before adding redundancy is a means to achieve *controlled redundancy*. In particular, controlled redundancy means that the backup software is able to control the distribution of redundant data.

Backup systems often rely on "traditional" stream compression techniques, such as *gzip* and similar tools. Additionally, most backup systems have focused on techniques allowing for storage and bandwidth savings when only part of the data of interest has been modified, i.e., incremental backup techniques. Of course, similar techniques are used by revision control systems [Lord, 2005] or network file systems [Muthitacharoen et al., 2001].

Incremental backup has the inherent property of reducing storage (and bandwidth) usage because only changes need to be

[2]There may be exceptions to this in some application scenarios where mobility patterns are known in advance. For instance, when two mobile device users take the same train every single morning while commuting.

sent to cooperating peers and stored. However, snapshot-based systems can be implemented such that they provide storage and bandwidth efficiency comparable to that of incremental backup systems, while still allowing for constant-time restoration. Namely, *single-instance storage* is a technique that has been used to provide these benefits to a number of backup [Cox and Noble, 2002, Landers et al., 2004, Rubel, 2005], archival [Quinlan and Dorward, 2002, You et al., 2005] and revision control systems [Lord, 2005], as well as distributed file systems [Muthitacharoen et al., 2001, Bolosky et al., 2000].

Single-instance storage consists in storing only once any given data unit. Thus, it can be thought of as a form of compression among several data units. In a file system, where the "data unit" is the file, this means that any given content, even when available under several file names, will be stored only once. The single-instance property may also be provided at a finer-grain level, thus allowing for improved compression.

The authors of Pastiche and PeerStore argue that single-instance storage can even be beneficial at the scale of the aggregated store made of each contributor store. In essence, they assume that a lot of data is common to several participants, and thus argue that enforcing single-instance of this data at a global scale can significantly improve storage efficiency.

While common data may easily be found among participants in the context of Pastiche and PeerStore, where each participant is expected to back up their whole disk (i.e., including application binaries and data), this is certainly not the general case. For example, the mobile users of MoSAIC are expected to explicitly pay attention to their personal, critical data which are unlikely to be shared among several participants. Consequently, single-instance storage may be beneficial to mobile users only when used at a local scale, i.e., on each data owner's local store **[Courtès et al., 2006]**.

3.5. Dependability Techniques

We study, in this section, the various techniques found in the literature to address the dependability issues presented in section 3.2.2..

3.5.1. Integrity and Consistency

Integrity and consistency are two properties that are usually obtained using some kind of data encoding. Apart from CBS, every system studied here systematically fragments the backup files. This is necessary for load-balancing: with small sized fragments, it is easier to adapt the placement to fairly balance the load imposed on the participants.

pStore uses simple data structures to encode the backup files. The files are fragmented in varying size blocks. Along with the blocks themselves, each node also stores a list of blocks that contains, for each version of the considered file, the list of the identifiers of its constituent blocks. Each block list is indexed with a structure containing a cryptographic hash of the file path and the key of its owner. There is thus one namespace per user. In practice, for the restoration of a given file, one needs to know the file path and the key of the owner. Without this metainformation, one cannot access the file's block list and consequently its blocks. The same technique is used in PeerStore, and a similar technique in Pastiche.

In ABS, each fragment is stored along with a block of meta-information about the file from which the block originates, as well as the position of the fragment within the file. These data (fragment and metainformation) are indexed using a cryptographic hash of the fragment to implement single instance storage of each fragment. The metainformation is encrypted using the owner's public key and the set (fragment and metainformation) with the owner's private key. These signatures are used to certify ownership and for ensuring integrity of the blocks.

In a similar manner, Venti-DHash encoding is based on Venti. As with a classical file system, the files are represented as trees whose leaves are the file fragments. Here, all the blocks are indexed using their digest. They are fixed-sized and the underlying storage middleware is not aware of their semantics (leaf nodes, intermediary nodes, data, metadata, etc.). To be able to restore a file, only the knowledge of the identifier of the root node is necessary.

All these techniques provide some guarantee of data block integrity since block addressing is realized using an identifier that depends on the content of the block (using a digest). When a block is restored, one can then check whether or not it is the requested block and if it is correct. If the metainformation concerning a file is stored using the same technique, integrity is thus also guaranteed file-wise. However, from the user point of view, several files can have common semantics and thus should form a consistency unit. Only Pastiche guarantees inter-file consistency. Since it is implemented as a file system, Pastiche can create shadow copies of the blocks being backed-up, so that they can be modified during the backup process without compromising their consistency.

3.5.2. Confidentiality and Privacy

Most of the systems studied here, like many file sharing systems, use *convergent encryption* [Bolosky et al., 2000] to provide some confidentiality despite untrustworthy partner nodes. The objective is to have an encryption mechanism that does not depend on the node performing the encryption, i.e., that is compatible with single-instance-storage. Convergent encryption is a symmetric encryption technique whose key is a digest of the block to be ciphered. The ciphered block can then be stored on the untrustworthy partner nodes. A digest of the ciphered block is commonly used as an identifier of that block. The tuple of digests (ciphered/unciphered block digests) is called digest-key or CHK for "content hash key". The data owner needs the CHK to be able to locate and uncipher a block. The CHKs are themselves backed up and ultimately the data owner only has to "remember" one CHK. Generally this ultimate CHK is stored using a secret that the data owner cannot forget, like his ID for instance. It is important to note that this technique can lead to some loss in privacy for the data owner. Indeed, when several nodes own the same block, since the ciphering scheme depends on the content and not on the nodes, they produce the same ciphered blocks, and the same CHKs. Thus they are each able to know that they share a file.

It is important to note that when single-instance-storage is not considered, it is much simpler to use classic encryption techniques, e.g., based on asymmetric ciphering.

3.5.3. Data Availability

In this section, we explore the techniques described in the literature for improving data availability despite failures while optimizing the use of the system resources: data replication and garbage collection.

3.5.3.1. Data Replication

For the systems that distribute the data among a specific set of participants (Pastiche, PeerStore, CBS, Flashback), the replication mechanism is quite simple. In Pastiche, each participant entering the system looks for 5 other participants having a lot of data in common with itself. These 5 participants then become its backup partners. It can thus tolerate 4 node failures. With PeerStore, the choice of partners is done in a different manner. However, the authors say that there are ideally as many partners as there are data replicas, which is similar to Pastiche. For the systems based on DHTs, thanks to (or because of) the properties of DHTs, the global set of data stored is homogeneously distributed among the nodes. Consequently, to tolerate the departure or the failure of a participating node, the data *has to* be replicated. In practice, the data blocks are generally replicated by the node that is responsible for it (with the closest ID) on a small number of its neighbors in the identifier space. Additionally, the block can also be kept in cache on the nodes that are on the path between the owner and the node responsible for it. ABS, among the systems based on DHTs, proposes an alternative. The data owner can choose the key under which a block will be stored. When a new block is inserted in the DHT, an attempt is made to insert it with a digest of the data as the key. A digest of the key itself (this is called rehashing) can also be used to store the block on some other participant in order to move the data or to tolerate a departure or crash.

Coding techniques are also used to finely control the level of data redundancy. Many different error-correcting coding

techniques can be used: erasure codes [Weatherspoon and Kubiatowicz, 2002] like Tornado [Byers et al., 1999], Fountain codes (also called rateless erasure codes) [Byers et al., 1998] like Raptor [Shokrollahi, 2003], etc. The idea of erasure codes is basically that each data block is fragmented into k fragments. From these k fragments, r other redundancy fragments are computed. From these $k + r$ fragments, any k fragments are sufficient to rebuild the original data block.

Blocks are thus used to produce fragments with a controlled level of redundancy. Venti-DHash uses this technique and stores the fragments on the successors of the node responsible for the original data block. MoSAIC also uses erasure codes for the production of redundant fragments but distributes them opportunistically to the nodes encountered.

3.5.3.2. Garbage Collection

Pastiche, pStore and ABS offer the possibility to delete the backed up data. Only the data owner can request this operation - requests must be signed with the owner's private key. Additionally, when single instance storage is used, as a block can be stored for several owners, an owner list is associated to each data block to permit its deletion only when every owner has requested it. In PeerStore, however, such an owner list does not exist (or is incomplete), i.e., other nodes can rely on a block for their own backup without having notified the node actually storing this block. This is due to the way PeerStore implements inter-node single-instance-storage. For this reason PeerStore does not allow delete operations. FlashBack uses the notion of a "lease" whereby a data block is stored for a given duration. This duration is determined a priori and exceeds the expected duration of unavailability of the data owner. Leases can be renegotiated when they are half-expired.

3.5.4. Service Availability

Failures of a cooperative backup system can lead to the loss of some of the stored data, as discussed in the previous section, but can also lead to the unavailability of the entire backup system, which we address in this section. Resilience to malicious denial-of-service attacks is a wide and active research field. The approaches used to mitigate the lack of trust between the participating nodes and to tolerate these DoS attacks can be based either on the notion of reputation (a level of the trust of the partners that can be acquired either locally or transitively) or on the use of micro-economy (exchange of checks, tokens, etc.) [Grothoff, 2003, Lai et al., 2003, Oualha and Roudier, 2006].

We concentrate here on the attacks that are specific to cooperative backup in general and more specifically the ones we found in the cooperative backup system we studied: selfishness and retention of backup data.

3.5.4.1. Selfishness

Selfishness is a problem for every resource sharing system, as we saw in Section 3.2.2.4.. Some mechanism is required to enforce fairness amongst peers - that they contribute in proportion to what they consume. Many different solutions have been proposed, most of them being based on the notion of micro-economy. We look here only at the solutions adapted to storage systems.

It is worth noting that it is not possible for a system based on DHTs to guarantee that the participants fairly contribute to the system with respect to the amount of resources they consume (see Section 3.4.1.). Consequently, Venti-DHash and pStore are not resilient to this type of attack. The ABS rehashing technique (see Section 3.5.3.1.) can be used to balance the loads on the DHT but it does not take the effective usage of each node into account.

PeerStore proposes a simple solution based on pair-wise symmetrical exchanges, i.e., each one of the two partners offers (approximately) the same storage capacity that it uses. To find partners, newcomers broadcast an offer for a given storage capacity and listen to other participant replies that offer some capacity in exchange that may be different. It is then up to newcomers to decide whether or not to accept an offer. CBS also imposes symmetrical exchange relationships, restricting

data placement.

Pastiche does not deal with this problem but Samsara does: it extends the notion of symmetrical exchanges with the use of *claims*. The data owner issues a claim for the node that accepts to store its data, this exchange constitutes a contract. The value of the claim represents the storage capacity of the stored data. These claims can be forwarded to another contributor when the contributor needs to store some of its own data. Finally, each node periodically checks its co-contractors to ensure that they are adhering to the contract, i.e., to verify that its claims are satisfied, by challenging its contributors. If a node breaches a contract, its partner is free to drop its data. The use of challenges can be seen as a way to compute locally a level of reputation for a contributor.

Another simple solution is proposed in CFS [Dabek et al., 2001]: each contributor limits any individual peer to a fixed fraction of its space. These quotas are independent of the peer's space contribution. CFS uses IP addresses to identify peers, and requires peers to respond to a nonce message to confirm a request for storage, preventing simple forgery. This does not defend against malicious parties who have the authority to assign multiple IP addresses within a block, and may fail in the presence of network renumbering.

Several of these solutions were proposed to be extended with trusted third parties, either centralized or distributed among trusted hardware devices. For instance, PAST [Rowstron and Druschel, 2001] provides quota enforcement that relies on a smartcard at each peer. The smartcard is issued by a trusted third party, and contains a certified copy of the node's public key along with a storage quota. The quota could be determined based on the amount of storage the peer contributes, as verified by the certification authority. The smartcard is involved in each storage and reclamation event, and so can track the peer's usage over time. Fileteller [Ioannidis et al., 2002] proposes the use of micro-payments to account for storage contributed and consumed. Such micro-payments can provide the proper incentives for good behavior, but must be provisioned and cleared by a third party and require secure identities.

It is worth noting that, as a side effect, solutions based on symmetrical exchanges have the advantage of being resilient to flooding attacks, whereby a node tries to obtain many storage resources by flooding the network with requests. On the contrary, DHT based systems are not resilient to this type of attack due to the very nature of DHTs.

3.5.4.2. Retention of Backup Data

Data retention is the situation in which a contributor does not release backed up data when an owner issues a restoration request. This can be non intentional, e.g., the contributor has crashed, or is disconnected, or intentional/malicious, e.g., the contributor did not actually store the data or tries to blackmail the data owner. Generally speaking, unintentional retention should be tolerated whereas malicious retention should be prevented, or even punished.

In CBS, there is a two-fold solution to these problems: first there are periodic challenges to verify that the partners really do store the data for which they are responsible for, and second, there are rules to tolerate temporary node failures. The periodic challenges are actually read requests for randomly chosen data blocks sent to the contributors by data owners. Tolerance of temporary faults is based on a *grace period* during which a participant can be legitimately unavailable. After expiration of the grace period, the data stored for the disconnected node can be erased (the data owner locally decides to associate a bad reputation to the contributor). However, the grace period can be used to gain resources dishonestly without contributing to the system. A countermeasure is to define a *trial period*, that is longer than the grace period, during which backup and challenges are permitted but restoration is not.

This challenge technique is also used by the other studied systems, in an optimized form: a challenge concerns several blocks at a time and the response is a signature of the set of blocks [Cox and Noble, 2003] [Landers et al., 2004].

Samsara and PeerStore also have a slightly different way of punishing *unavailable* nodes: their blocks are progressively

deleted. The probability of deletion of a block is chosen such that, given the number of block replicas, the probability of all the replicas being deleted becomes significant only after a large number of unsatisfied challenges.

3.6. Conclusion

Peer-to-peer/cooperative systems constitute a new emerging approach for the design of heavily distributed systems. They have very good properties regarding scalability and are thus particularly well-adapted to ubiquitous computing scenarios. The application of peer-to-peer coopearation to backup has been rendered feasible by recent dramatic increases in storage capacity and network bandwidth. In this chapter, we have surveyed the technical solutions to this problem.

We first observed that the field of cooperative backup for wide-area networks or local-area networks is very active. This research field has been recently boosted by the peer-to-peer trend and reused many of the P2P techniques: distributed hash tables, single-instance-storage, convergent encryption, etc. However, very little work has targeted mobile devices, even if cooperative backup seems to be quite appropriate for them (new data is frequently produced on many types of devices, even disconnected from the fixed infrastructure: digital cameras, phones, PDAs, laptops. However, mobile devices have their specificities (ephemeral connections, reduced energy, etc.), so many of the techniques developed for WANs and LANs cannot be applied. Much effort is still needed to alleviate the specific problems raised by frequent disconnections, ephemeral connections, limited battery power, inability to access trusted third parties, etc.

From this situation, trails that can be followed to make some progress in this field include: adequate disconnected cooperation incitatives, proper erasure codes with varying parameters, realistic mobility models, and stochastic models of the dependability of mobile devices implementing cooperative services.

4 - Wait-free objects: an introduction for the sophomore

Introduction

Let us consider an object, defined by a sequential specification (e.g., a queue), that is used by concurrent processes. One way to implement the operations enqueue() and dequeue() consists in using a mutual exclusion mechanism (e.g., locks) to allow a single operation at a time to proceed. This works as long as there are no failures. If a process *p* crashes after it has acquired the critical section and before releasing it, it is impossible in an asynchronous system for the other processes to know if this process has crashed or is only very slow, and consequently the whole set of processes can deadlock. Moreover, if *p* is very slow, it momentarily slows down the processes that want to acquire the object.

This observation has motivated the notion of *wait-free* implementation. The implementation of an object is wait-free if any invocation of an operation of that object, by a process that is not faulty, terminates in a finite number of steps. As we can see, wait-free implementations provide starvation-free and $(n - 1)$-resilient tolerant objects.

The wait-free notion originated a long time ago (1977) in a paper by Leslie Lamport [Lamport 1977]. It was then investigated in [Peteron 1983] and formalized much later by Maurice Herlihy [Herlihy 1991] who received the ACM Dijkstra award for that paper in 2004. That paper defines the notion of *consensus number* that can be associated with any object that has a sequential specification. This notion allows objects to be ranked with respect to their power to solve the consensus problem (an object has consensus number *n* if, together with atomic registers[1], it allows a wait-free solution of consensus for a set of processes). It appears that consensus numbers allows ranking the power of synchronization primitives such as Test&Set(),

[1] A register provides the processes with a read operation and a write operation. It is *atomic* if each operation issued by a processappears as if it has been executed instantaneously at some point between its beginning and its end. This definition generalizes

Compare&Swap(), etc., in the presence of process crashes.

This report constitutes a short introduction to the notion of wait-free objects. Instead of presenting an exhaustive survey (that would be very technical) of the domain, it considers three objects and shows how they can be wait-free implemented. The first two objects (a counter and a splitter) are very simple. The third one (a snapshot object) is a little bit more involved. This survey presents wait-free implementations AND their proofs. Understanding the proofs is important to obtain a better insight into wait-free computing, as in a lot of cases, the algorithms can be expressed in a few lines, but are far from being trivial (the third algorithm presented here remains relatively simple compared to a lot of wait-free algorithms).

Among the application domains of wait-free computing, there are fault-tolerance [Attiya and Welch 1998, Lamport 1996] and real-time systems [Raynal 2002].

4.1. Computation model

4.1.1. Processes

The systems we consider are static systems that consist of n sequential processes, denoted $p_1, \ldots, p_n$. The processes cooperate by accessing concurrent objects. A process is a predefined flow of control that can be perceived as a processor executing a sequence of operations defined by a local algorithm associated with it. Such an algorithm can be described by an automaton (with a possibly unbounded number of states).

A process is said to be *correct* in a run when it executes an infinite number of steps in that run. Otherwise, the process is said to be *faulty* in the considered run. Sometimes it is convenient to see a faulty processes as one that crashes and prematurely halts its execution. A process is supposed not to deviate from the algorithm assigned to it (until it possibly crashes). Being sequential, a process executes (at most) one operation at a time. Unless explicitly stated otherwise, the processes are assumed to be *asynchronous* which means that the execution speed of each process is arbitrary (but positive until it possibly crashes). It follows that there is no assumption on the processing speed of one process with respect to another.

An invocation by a process of an operation on object X is denoted $X.op(arg)(res)$ where arg designates the input parameters associated with this invocation, and res designates the associated results returned to the calling process. When the input parameter and the result parameter are not important, $X.op(arg)(res)$ is denoted $X.op()$.

The execution of an operation $op()$ on an object X by a process p_i is modeled by two events, namely, the event $inv[X.op(arg)$ by $p_i]$ that occurs when p_i invokes the operation (invocation event), and the event $resp[X.op(res)$ by $p_i]$ that occurs when the operation terminates (response event). Accordingly, a process can be abstracted as the sequence of the events it generates. Given an operation $X.op(arg)(res)$ invoked by p_i, $inv[X.op(arg)$ by $p_i]$ and $resp[X.op(res)$ by $p_i]$ are said to be *matching* events.

Given an execution, denoted $\widehat{H}$, all the invocation and response event can be totally ordered. The corresponding sequence of events is denoted $<_H$.

4.1.2. Objects

As indicated, the processes cooperate through concurrent objects (also called *shared* objects). An object has a name and a type. A type is defined by (1) the set of possible values for objects of that type; (2) a finite set of specific operations that are the only way to access and manipulate the objects of that type; and

easily to atomic objects. An atomic object is also said to be *linearizable* [Herlihy and Wing 1990]. It follows from the definition of atomicity that an atomic object can always be defined with a sequential specification.

(3) a specification giving the meaning of these operations. Figure 1 presents a structural view of a set of n processes sharing m objects.

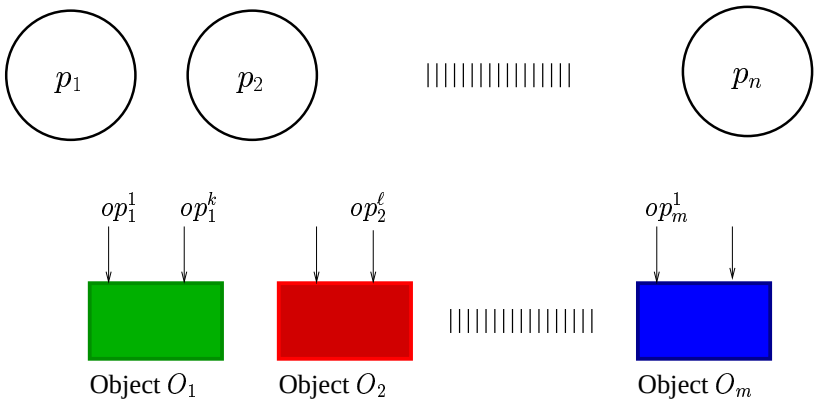

Figure 1: Structural view of a system

Sequential specification Objects are supposed to have a sequential specification. This depicts the semantics of the object when accessed sequentially by a correct process. The specification is a set of traces defining the allowed sequences of operations accessing the object (or equivalently, by the corresponding allowed sequences of invocation and response events). Alternatively, this means that the behavior of each operation can be described by a pre-assertion and a post-assertion. Assuming the pre-assertion is satisfied before executing the operation, the post-assertion describes the new value of the object and the result of the operation returned to the calling process.

An object operation is *total* if it is defined for every value of the object; otherwise it is *partial*.

Example 1: a read/write object To illustrate these definitions, let us consider consider the following objects. The first is the classical *register* that models a shared file. It has two operations:

- The operation $read()$ has no input parameter. It returns a value of the object.
- The operation $write(v)$ has an input parameter, namely v, a new value of the object. The result of that operation is a value ok indicating to the calling process that the operation has terminated.

The sequential specification of the object is defined by all the sequences of read and write operations in which each read operation returns the value of the last preceding write operation. Clearly, the read and write operations are always defined: they are total operations.

The problem of implementing a concurrent read/write object is a classical synchronization problem known under the name *reader/writer* problem.

Example 2: a fifo queue with total operations Our second example is an unbounded queue. This object has a sequential specification defined by all the traces of allowed sequences of enqueue and dequeue operations. As we can see, this definition never prevents an enqueue or a dequeue operation from being executed (dequeuing an empty queue returns a predefined default value). Both operations are total. The problem of implementing a concurrent queue object is a classical synchronization problem known under the name *producer/consumer* problem.

4.2. A very simple wait-free object: a counter

4.2.1. Definition

A shared counter C is a concurrent object that has an integer value (initially 0), and provides the processes with two operations denoted $increment()$ and $get\_count()$. Informally, the $increment()$ operation increases the value of the counter by 1, while the $get\_count()$ operation returns its current value. In a more precise way, the behavior of a counter can be specified by the three following properties.

- Liveness. Any invocation of $increment()$ or $get\_count()$ by a correct process terminates.
- Monotonicity. Let gt_1 and gt_2 be two invocations of $get\_count()$ such that gt_1 returns c_1, gt_2 returns c_2, and gt_1 terminates before gt_2 starts (i.e., $resp[gt_1] <_H inv[gt_2]$). Then, $c_1 \leq c_2$.
- Uptodateness. Let gt be an invocation of $get\_count()$ and c the value it returns. Let c_a be the number of invocations of $increment()$ that terminate before gt starts (i.e., before the event $inv[gt()]$ occurs). Let c_b be the number of invocations of $increment()$ that start before gt terminates (i.e., before the event $res[gt]$ occurs). Then, $c_a \leq c \leq c_b$.

The liveness property expresses that the object is wait-free. The monotonicity and uptodateness properties give its meaning to the object: they define the values that can be returned by a $get\_count()$ invocation.

It is easy to see that the behavior of a counter object defined by the previous specification can also be described by a sequential specification. More specifically, a given counter execution (as defined by a total order $\widehat{H}$ on its invocation and response events) is correct if there is an equivalent sequence $\widehat{S}$ such that (1) the order of the operations in $\widehat{S}$ respects their real-time occurrence order (as defined by $<_H$) and (2) each $get\_count()$ returns the number of $increment()$ operations that precede it in $\widehat{S}$. Item (1) follows from the monotonicity and uptodateness properties, while item (2) follows from the uptodateness property. The previous specification defines consequently an *atomic object* [Herlihy and Wing 1990].

4.2.2. A simple counter construction

A concurrent counter can easily be built as soon as the number of processes n is known, and the system provides one base 1WMR atomic register per process (1WMR means single-writer/multi-reader). More precisely, let $REG[1:n]$ be an array of atomic registers initialized to 0, such that, for each i, $REG[i]$ is written only by p_i and read by all the processes. The algorithms implementing the $increment()$ and $get\_count()$ operations are trivial (Figure 2). The invocation of $increment()$ by p_i consists in asynchronously adding 1 to $REG[i]$. The invocation of $get\_count()$ by any process consists in reading (in any order) and summing up the values of all the entries of the array $REG[1:n]$.

```
operation increment() invoked by p_i:
    aux ← REG[i] + 1;
    REG[i] ← aux;
    return ()

operation get_count():
    aux ← 0;
    for j ∈ {1, ..., n} do aux ← aux + REG[j] end_do;
    return (aux)
```

Figure 2: A wait-free counter

Theorem 1 *The algorithm described in Figure 2 is a wait-free implementation of a counter object.*

Proof The fact that the operations are wait-free follows directly from their code. The proof that the construction provides an atomic counter is based on the atomicity of the underlying base registers. A *linearization point*[2] is first associated with each operation, as follows:

- The linearization point associated with an $increment()$ operation issued by a process p_i is the linearization point of the underlying write operation of the base atomic register $REG[i]$.
- Let c be the value returned by a $get\_count()$ operation. The linearization point associated with that operation is, while reading and summing up the underlying base registers, the point in the execution when the value c is attained.

According to this linearization point definition, and the fact that the base registers never decrease, it is easy to conclude that (1) if two $get\_count()$ operations are sequential, the second one returns a value not smaller than the first one (monotonicity property), (2) a $get\_count()$ operation returns a value not smaller than the number of $increment()$ operations that terminated before it starts, and not greater than the number of $increment()$ operations that have started before it terminates (uptodateness property). $\square_{Theorem\ 1}$

4.3. Another simple wait-free object: a splitter

The splitter object was introduced by Lamport in [Lamport 1987] to solve the *fast mutual exclusion* problem. This problem consists in allowing a process that wants to enter a critical section, to execute only a bounded number of steps before entering when no other process wants to enter the critical section. This means that when a single process wants the critical section, it has to pay a bounded price (counted in number of base operations), while it pays a price that depends on the number of processes when several processes are contending.

The splitter object has been used to design wait-free implementations of several objects. The most popular is the *renaming object* [Afek and Merritt 1999, Attiya et al. 1990, Attiya and Fouren 2000, Attiya and Fouren 2001, Moir 1998, Moir and Anderson 1995]. We limit our presentation to the splitter. The reader interested in its use can consult [Raynal 2004].

4.3.1. Splitter definition

A *splitter* is a wait-free concurrent object that provides the processes with a single operation, denoted $direction()$, that returns one out of three possible values ($stop$, $down$ or $right$) to the invoking process. Assuming that each process invokes at most once the $split()$ operation, a splitter is characterized by the following global property: if x processes invoke $direction()$, then at most one obtains the value $stop$, at most $x-1$ obtain the value $down$, and at most $x-1$ obtain the value $right$ (Figure 3).

More precisely, assuming a process invokes at most once the $direction()$ operation, a splitter is specified by the following properties:

- Liveness. An invocation of $direction()$ by a correct process always returns a value.
- Validity. The value returned by $direction()$ is $stop$, $right$ or $down$.
- Solo execution. If a single process accesses the splitter it obtains the value $stop$ (or crashes).

[2] This is the point on the time line at which the operation appears to have instantaneously executed.

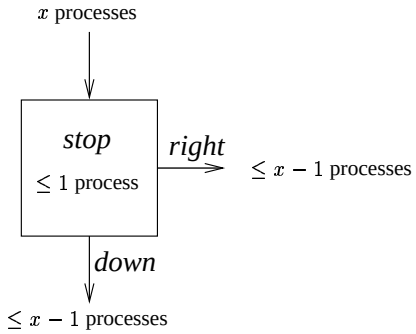

Figure 3: A splitter

- Partitioning. At most one process obtains the value $stop$. When two or more processes access the splitter, not all of them obtain the same value.

As we can see, a splitter is an object that allows an online partitioning of a set of contending processes into smaller groups with certain properties. A process is assigned exactly one value, depending on the current contention and the values already assigned. It is important to see that, when several processes access a splitter, it is possible that none of them obtains the value $stop$.

4.3.2. Construction

A surprisingly simple wait-free implementation of a splitter is described in Figure 4. The internal state of the splitter is represented by two atomic MWMR base registers (MWMR means multi-writer/multi-reader).

- The aim of the first atomic register, denoted $LAST$, is to contain the identity of the last process that entered the splitter (we consider that i is the identity of p_i).
- The second register, denoted $DOOR$, can take two values, namely, $open$ and $closed$. Initially, $DOOR = open$. The aim of that register is to route the processes towards the "right" exit as soon as the door has been closed. Several processes can close the door (closing the door is an idempotent operation).

```
procedure direction() invoked by p_i:
(1)  LAST ← i;
(2)  if (DOOR = closed) then return (right)
(3)                     else DOOR ← closed;
(4)                          if (LAST = i) then return (stop)
(5)                                        else return (down)
(6)  end_if                 end_if
```

Figure 4: A wait-free construction of a splitter

When a process p_i invokes the $direction()$ operation, it first leaves a mark indicating it is the last process that entered the splitter (line 1). Then, it outputs the value $right$ if the door is closed (line 2). If the door was not closed when it checked, it closes it (line 3). (Note that due to asynchrony, it is possible that, in the meantime, the door has been closed by other processes.) Finally, if no other process entered the splitter since the time it executed line 1, the process p_i returns the value $stop$; otherwise, it returns the value $down$.

A process that returns $right$ is actually a *late* process: it arrived late at the splitter and found the door closed. Differently, a process that returns $down$ is actually a *slow* process: it closed the door ($DOOR \leftarrow closed$ at line 3) but was not quick enough during the period starting when it wrote its identity in $LAST$ at line 1, and ending when it read that register at line 4. The management of the routing registers $LAST$ and $DOOR$ ensures that at most one process can be neither late nor slow, such a process is *on time*, and obtains $stop$. As already noticed, it is possible that no process be on time.

4.3.3. Proof of the construction

Theorem 2 *The algorithm described in Figure 4 defines a wait-free implementation of a splitter object.*

Proof As there is no loop, the $direction()$ operation is trivially wait-free. Moreover, it follows from the initialization of the atomic register $DOOR$ that, if a single process invokes $direction()$ it obtains the value $stop$ (if it does not crash before returning a value).

Assuming now that $x > 1$ processes access the splitter object, let us first observe that, due to the initialization of $DOOR$, not all of them can obtain the value $right$; this is because for a process to obtain $right$, another process has to previously close the door (statement $DOOR \leftarrow closed$ at line 3). It follows that at most $x - 1$ processes can obtain the value $right$.

Let us now consider the last process that executes line 1. If it does not crash, according to the fact that it finds the door closed or not (line 2), that process returns the value $right$ or $stop$. Consequently, not all the processes can obtain the value $down$.

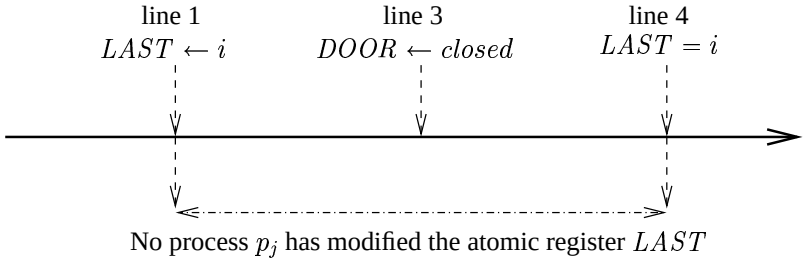

Figure 5: Proof a the splitter construction

Finally, no two processes can obtain the value $stop$. If no process p_i finds $LAST = i$ when it executes line 4, no process returns the value $stop$. So, let us consider the first process p_i (if any) that finds $LAST = i$ at line 4 (see Figure 5 where the time line corresponds to the total order defined by the linearization of the read and write operations of p_i on the base atomic registers $LAST$ and $DOOR$). That process returns the value $stop$ (if it does not crash). This means that no process p_j has modified $LAST$ while p_i was executing the lines 1-4. It follows that any $p_j \neq p_i$ that will modify $LAST$ (at line 1) will find the door closed (line 2). Consequently, p_j cannot obtain the value $stop$. $\square_{Theorem\ 2}$

4.4. A less trivial wait-free object: a snapshot object

The concept of *snapshot* object was introduced in [Afek et al. 1993]. It has been used as a building block for implementing a lot of wait-free objects [Attiya and Fouren 2001]. Moreover, from a design point of view, it uses basic principles encountered in other wait-free algorithms. A snapshot object provides two operations, denoted $update()$ and $snapshot()$, whose meaning is defined by the following properties:

- An invocation of $snapshot()$ returns n values (one per process) to the invoking process. The returned value v_i associated with the process p_i is such that $update(v_i)$ is the last $update()$ invoked by p_i before the $snapshot()$, or is an $update()$ invoked by p_i concurrent with the $snapshot()$.
- Any $update()$ or $snapshot()$ operation invoked by a correct process terminates.
- The snapshot object is atomic. This means that it is possible to totally order the $update()$ and $snapshot()$ operations in such a way that the order of non-concurrent operations is preserved, and each $snapshot()$ operation returns, for each i, the value v_i such that $update(v_i)$ is the last $update()$ operation issued by p_i preceding (in that total order) the $snapshot()$ operation.

A snapshot object provides the processes with a high level cooperation abstraction that can greatly simplify both the design and the proof of asynchronous concurrent programs. The $update()$ operation allows a process to inform the other processes on its progress (by writing the last relevant value it has computed in the snapshot object). The $snapshot()$ operation allows a process to obtain a global picture of the system state (i.e., the last value deposited by each process) as it is was obtained instantaneously.

4.5. A snapshot construction

4.5.1. A first attempt based on sequence numbers

A simple representation of a snapshot object consists in an array $REG[1 : n]$, where each $REG[i]$ is a 1WMR atomic register that can be written only by p_i.

A first idea that comes to mind to implement a snapshot object consists in associating a sequence number with each value written by a process. So, each atomic $REG[i]$ register contains two fields, denoted $REG[i].sn$ and $REG[i].val$. The corresponding $update()$ operation is as follows, where sn_i is a local variable that p_i uses to generate sequence numbers):

operation $update\ (v)$ **invoked by** p_i:
$sn_i \leftarrow sn_i + 1$; % local sequence number generator %
$REG[i] \leftarrow (v, sn_i)$ % atomic write of a pair %.

Then, thanks to the sequence numbers, the idea is for a $snapshot()$ operation to use a "double scan" technique. The $scan()$ function asynchronously reads the last (sequence number, data) pairs deposited by each process ($reg[1 : n]$ is an auxiliary array whose scope is the $scan()$ function):

function $scan()$: **for** $j \in \{1, \ldots, n\}$ **do** $reg[j] \leftarrow REG[j]$ **end_do**; *return* (reg).

The *double scan* technique consists in repeatedly reading the whole array $REG[1 : n]$, until there are two consecutive $scan()$ invocations such that, for each register $REG[j]$, both read the same sequence number. When this occurs, the double scan is said to be *successful*. The $snapshot()$ algorithm based on this technique is as follows. (Given an array aa, the notation $aa.val$ denotes the array $[aa[1].val, \ldots, aa[n].val]$.)

operation $snapshot()$:
$aa_i \leftarrow scan()$;
while $true$ **do**
$bb_i \leftarrow scan()$;
if $(\forall j : aa_i[j].sn = bb_i[j].sn)$ **then** *return* $(aa_i.val)$ **end_if**;
$aa_i \leftarrow bb_i$
end_while.

To simplify, we consider in the following theorem that no process crashes. This restriction will be removed in the next section.

Theorem 3 *The set of operations that terminate implements an atomic object.*

Proof As the base registers are atomic, we can consider that their read and write operation occur instantaneously at some point in time. Let us define the linearization point of an $update()$ operation issued by p_j as the time instant when p_j writes $REG[j]$.

Considering now a $snapshot()$ operation that terminates, let $sc1$ and $sc2$ be its last two $scan()$ invocations. Moreover, let t_1^i (resp., t_2^i) be the time at which $sc1$ (resp., $sc2$) reads $REG[i]$. As both $sc1$ and $sc2$ obtain the same sequence number from $REG[i]$, we can conclude that, for any i, between t_1^i and t_2^i, no $update()$ operation issued by p_i has terminated and modified $REG[i]$. It follows that, between $\max(\{t_1^i\}_{1\leq i\leq n})$ and $\min(\{t_2^i\}_{1\leq i\leq n})$, no atomic register $REG[i]$, $1 \leq i \leq n$, has been modified. It is consequently possible to associate with the corresponding $snapshot()$ operation, a linearization point t of the time line such that $\max(\{t_1^i\}_{1\leq i\leq n}) < t < \min(\{t_2^i\}_{1\leq i\leq n})$. Hence we can consider that the $snapshot()$ occurred instantaneously at t (see Figure 6) after all the $update()$ operations whose it reads the values, and before the $update()$ operations that terminate after it.

It follows from the previous definition of the linearization points, that the operations that terminate define an atomic snapshot object. $\square_{Theorem\ 3}$

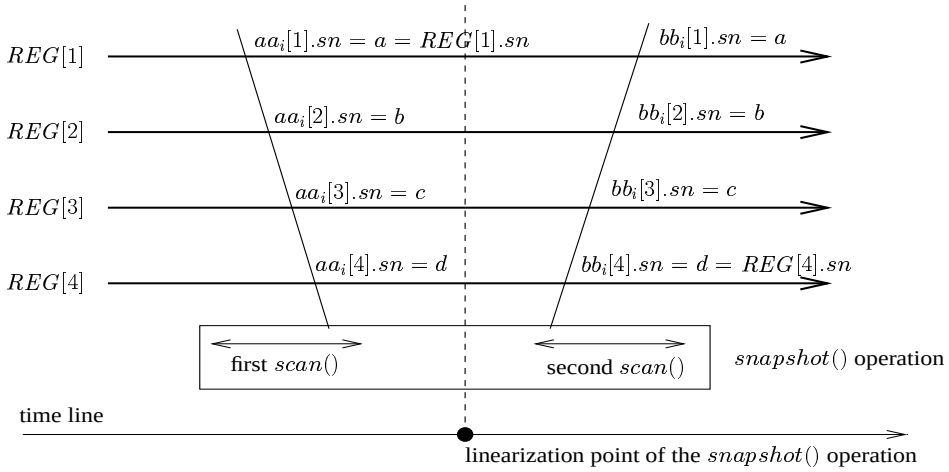

Figure 6: Linearization point of a $snapshot()$ operation (Case 1)

Let us notice that an $update()$ operation always terminates. This is an immediate consequence of the code of the corresponding algorithm. Unfortunately, it is possible that, as implemented by the previous double scan algorithm, a $snapshot()$ operation $snap$ never terminates. This can occur when one or several processes continuously execute $update()$ operations in such a way that the termination predicate $(\forall j : aa_i[j].sn = bb_i[j].sn)$ is never satisfied when it is checked by $snap$. It is important to see that this is not due to the fact that processes can crash, but to the fact that some processes continuously execute $update()$ operations. This is a typical *starvation*[3] situation that has to be prevented in order to obtain a wait-free construction.

[3] A process starves when it is forever prevented from terminating the execution of its operation.

4.5.2. A bounded wait-free snapshot construction

A basic principle to obtain wait-free constructions (when possible) consists in using a helping mechanism. Here, that principle can be translated as follows. If, continuously, there are $update()$ operations that could prevent a $snapshot()$ operation from terminating, why not to try using these $update()$ operations to help the $snapshot()$ to terminate. The solution we present relies on this nice and simple idea.

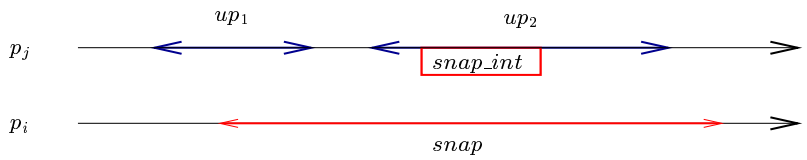

Figure 7: Each $update()$ operation includes a $snapshot()$ operation

Analyzing the previous attempt, we can make the two following observations:

- The fact that a double scan is unsuccessful (i.e., does not allow the $snapshot()$ operation to terminate) can be attributed to one or more $update()$ operations, namely the ones that have increased sequence numbers, and consequently made false the test that controls the termination of the algorithm.
- If a $snapshot()$ operation (say $snap$) issued by a process p_i sees two distinct $update()$ operations from the same process p_j (these updates, say up_1 and up_2, write distinct sequence numbers in $REG[j]$), we can conclude that up_2 was entirely executed during $snap$ (see Figure 7). This is because the update by p_j of the base atomic register $REG[j]$ is the last operation executed in an $update()$ operation.

As the second update up_2 is entirely executed during $snap$ (it starts after it and terminates before it), these observations suggest that up_2 might help $snap$. This help can be realized as follows:

- Each update is required to include an "internal" $snapshot()$ invocation (see Figure 7, where the rectangle in the $update()$ operation up_2 corresponds to the $snapshot()$ operation -denoted $snap\_int$- invoked by that update). Let $help$ be the array of values read by the internal snapshot $snap\_int$.
- As the execution of $snap\_int$ is entirely overlapped by the execution of $snap$, $snap$ can borrow the array $help$ and return it as its own result. (Notice it is possible that the $snapshot()$ invocation inside up_1 be also entirely overlapped by $snap$. But, there is no way to know it.) This overlapping is important to satisfy the atomicity property, namely, the values returned have to be consistent with the real-time occurrence order of the operations.

The proof will show that the addition of a $snapshot()$ operation in each $update()$ does not prevent the wait-free property, and this helping technique is consistent in the sense that the resulting snapshot object is atomic. The resulting $update()$ and $snapshot()$ algorithms are described in Figure 8. The function $scan()$ is the same as before.

A 1WMR atomic register $REG[i]$ is now made up of three fields, $REG[i].val$ and $REG[i].sn$ as before, plus the new field $REG[i].help\_array$ whose aim is to contain a helping array as previously discussed. The final version of both $update()$ and $snapshot()$ algorithms is a straightforward extension of their previous versions.

The main novelty lies is the local variable $could\_help_i$ that is used by a process p_i when it executes $snapshot()$. The aim of this set, initialized to $\emptyset$, is to contain the identity of the processes that have terminated an $update()$ invocation since p_i started its current $snapshot()$ operation. Its use is described at

lines 11-15. More precisely, when a double scan is unsuccessful, p_i does the following with respect to each process p_j that made the double scan unsuccessful (i.e., such that $(aa_i[j].sn \neq bb_i[j].sn)$):

- If $j \notin could\_help_i$ (line 14): p_i discovers that p_j has terminated an $update()$ since it started its $snapshot()$ invocation. Consequently, p_j could help p_i if it executes a new $update()$ while p_i has not yet terminated its $snapshot()$.
- If $j \in could\_help_i$ (line 13): in that case, p_j has entirely executed an $update()$ operation while p_i is executing its $snapshot()$ operation. As we have seen, p_i can benefit from the help provided by p_i by returning the array that p_j has stored in $REG[j]$ at the end of that $update()$ operation.

```
operation update(v) invoked by p_i:
(1)  help_array_i ← snapshot();
(2)  sn_i ← sn_i + 1;
(3)  REG[i] ← (v, sn_i, help_array_i)

operation snapshot():
(4)  could_help_i ← ∅;
(5)  aa_i ← scan();
(6)  while true do
(7)    bb_i ← scan();
(8)    if (∀j ∈ {1, ..., n} : aa_i[j].sn = bb_i[j].sn)
(9)      then return (aa_i.val)
(10)     else for_each j ∈ {1, ..., n} do
(11)                  if (aa_i[j].sn ≠ bb_i[j].sn) then
(12)                    if (j ∈ could_help_i)
(13)                      then return (bb_i[j].help_array)
(14)                      else could_help_i ← could_help_i ∪ {j}
(15)                  end_if end_if
(16)           end_for
(17)   end_if;
(18) aa_i ← bb_i
(19) end_while
```

Figure 8: Atomic snapshot object construction

4.6. Proof of the construction

The proof of the construction described in Figure 8 consists in two parts: (1) showing that every operation issued by a correct process terminates (wait-free property), and (2) showing that the object is atomic.

4.6.1. The snapshot object construction is bounded wait-free

This section shows that the previous construction not only is wait-free, but is bounded wait-free. This means that any operation invoked by a correct process terminates in a bounded number of operations on base objects (here the base 1WMR atomic registers $REG[j]$).

Theorem 4 *Each $update()$ or $snapshot()$ operation issued by a correct process returns after at most $O(n^2)$ operations on base registers.*

Proof Let us first observe that an $update()$ by a correct process always terminates as soon as the $snapshot()$ operation it invokes returns. So, the proof consists in showing that any $snapshot()$ issued by a correct process p_i terminates.

Let us consider that p_i has not returned after having executed n times the **while** loop (lines 5-19). This means that, each time it has executed that loop, it has found an identity j such that $aa_i[j].sn \neq bb_i[j].sn$ (line 11), which means that the corresponding process p_j has executed a new $update()$ operation between the last two $scan()$ operations of p_i. Each time this occurs, the corresponding process identity j is a new identity added to the set $could\_help_i$ at line 14. (If j was already present in $could\_help_i$, p_i would have executed line 13 instead of line 14, and would have consequently terminated the $snapshot()$ operation.)

As by assumption, p_i executes n times the loop, it follows that we have $could\_help_i = \{1, 2, \ldots, n\}$, i.e., this set contains all the process identities. It follows that, when it executes the loop for $(n+1)$th time and the test at line 8 is false, whatever the processes p_j such that $aa_i[j].sn \neq bb_i[j].sn$ at line 11, we necessarily have $j \in could\_help_i$ at line 12, from which it follows that p_i returns at line 13. The construction is consequently wait-free as p_i terminates after a finite number of operations on base registers have been executed.

Let us now replace "finite" by "bounded", i.e., let us determine a bound of the number of accesses to base registers. A $scan()$ costs $O(n)$ accesses to base registers. The cost of each iteration of the **for** loop (lines 11-15) is $O(1)$, and there are n iteration steps, which means that the cost of that loop is $O(n)$. Finally, as the enclosing **while** loop is executed at most $n+1$ times, it follows that a process issues at most $O(n^2)$ accesses to base registers when it executes a $snapshot()$ or $update()$ operation. $\square_{Theorem\ 4}$

4.6.2. The snapshot object construction is atomic

Theorem5 *The object built by the algorithms described in Figure 8 is atomic.*

Proof The proof that the object is atomic consists in providing, for each execution, a total order $\widehat{S}$ on $update()$ and $snapshot()$ operations that: (1) $\widehat{S}$ includes all the operations issued by the processes, except possibly, for each process, its the last operation if that process crashes, (2) $\widehat{S}$ respects the real-time occurrence order on these operations (i.e., if $op1$ terminates before $op2$ starts, $op1$ appears before $op2$ in $\widehat{S}$), and (3) $\widehat{S}$ respects the semantics of each operation (i.e., a $snapshot()$ operation has to return, for each process p_j, the value v_j such that, in $\widehat{S}$, there is no $update()$ operation by p_j between $update(v_j)$ and that $snapshot()$ operation).

The definition of the sequence $\widehat{S}$ relies on the fact that the scan invocations in a $snapshot()$ are sequential, and on the atomicity of the base registers. This means that the read and write operations on these registers can be considered as being executed instantaneously, each one at a point of the time line, and no two of them at the same time.

The sequence $\widehat{S}$ is built as follows. The linearization point of an $update()$ operation is the time at which it atomically executes the write in the corresponding 1WMR register (line 3). The definition of the linearization point of a $snapshot()$ operation issued by a process depends on the line at which it returns.

- The linearization point of a $snapshot()$ operation that terminates at line 9 (successful double $scan()$) is at any time between the end of the first $scan()$ and the beginning of the second $scan()$ (see Theorem 3 and Figure 6).
- The linearization point of a $snapshot()$ operation that terminates at line 13 (i.e., p_i terminates with the help of another process p_j) is defined inductively as follows. (See Figure 9 where a rectangle below an $update()$ operation represents the $snapshot()$ invoked by that $update()$. The dotted $help\_array$

arrow shows the way an array can be conveyed from a successful double collect by a process p_k until a $snapshot()$ operation issued by a process p_i in order to help that $snapshot()$ operation).

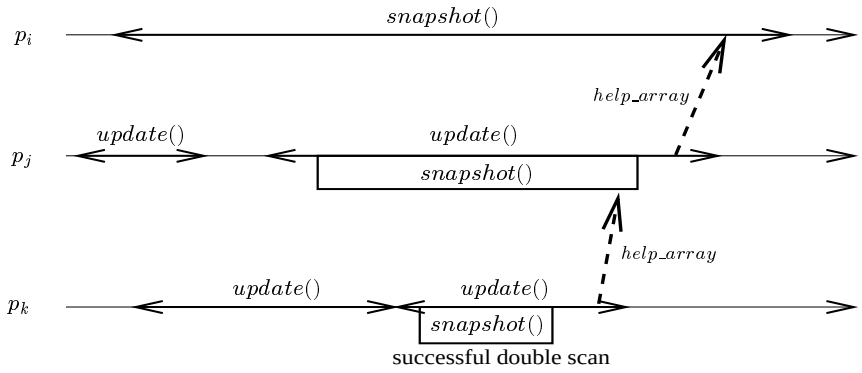

Figure 9: Linearization point of a $snapshot()$ operation (case 2)

The array (say $help\_array$) returned by p_i has been provided by an $update()$ operation executed by some process p_j. As already seen, this $update()$ has been entirely executed within the time interval of p_i's current $snapshot()$. This array has been obtained by p_j from a successful double scan, or from another process p_k. If it has been obtained from a process p_k, let us consider the way $help\_array$ has been obtained by p_k. As there are at most n concurrent $snapshot()$ operations, it follows by induction that there is a process p_x that has executed a $snapshot()$ operation and has obtained $help\_array$ from a successful double scan. Moreover, that $snapshot()$ was inside an $update()$ operation that was entirely executed within the time interval of p_i's current $snapshot()$.

The linearization point of the $snapshot()$ operation issued by p_i is defined as the internal $snapshot()$ whose double scan determined $help\_array$. If several $snapshot()$ operations are about to be linearized at the same time, they are ordered according to the total order in which they have been invoked.

It follows directly from the previous definition of the linearization points associated with the $update()$ and $snapshot()$ operations issued by the processes that $\widehat{S}$ satisfies the items (1) and (2) stated at the beginning of the proof. The fact that item (3) is also satisfied comes from the fact that the array returned by a $snapshot()$ has always been obtained from a successful double scan. $\square_{Theorem\ 5}$

4.7. Our (2006) contribution to wait-free computing

This report has surveyed the notion of wait-free objects (objects provided with a wait-free implementation). It has visited three wait-free implementations that give a first flavor of wait-free computing. As soon as fault-tolerance or real-time are concerned, wait-free computing becomes a first class notion. For more information on wait-free algorithms, the reader can consult the list of papers cited in the references.

Our current research at IRISA on wait-free computing is mainly oriented towards computability. Here we summarize two of our main results. In addition to the results themselves, these works open research directions that have not yet been investigated by the research community.

- The adaptive renaming problem consists in designing an algorithm that allows p processes (in a set of n processes) to obtain new names despite asynchrony and process crashes, in such a way that the size

of the new renaming space M be as small as possible. It has been shown that $M = 2p - 1$ is a lower bound for that problem in asynchronous atomic read/write register systems.

Our work described in [Mostefaoui et al. 2006] is an attempt to circumvent that lower bound. To that end, considering first that the system is provided with a k-set object, the paper presents a surprisingly simple adaptive M-renaming wait-free algorithm where $M = 2p - \lceil \frac{p}{k} \rceil$. To attain this goal, the paper visits what we call Gafni's reduction land, namely, a set of reductions from one object to another object as advocated and investigated by Gafni [Borowsky and Gafni 1993-1, Borowsky and Gafni 1993-2, Gafni 2004, Gafni and Rajsbazum 2005, Gafni et al. 2006]. Then, the paper shows how a k-set object can be implemented from a leader oracle (failure detector) of a class denoted Ω^k. To our knowledge, this is the first time that the failure detector approach is investigated to circumvent the $M = 2p - 1$ lower bound associated with the adaptive renaming problem. In that sense, the paper establishes a connection between renaming and failure detectors.

- Asynchronous failure detector-based set agreement algorithms proposed so far assume that all the processes participate in the algorithm. This means that (at least) the processes that do not crash propose a value and consequently execute the algorithm. It follows that these algorithms can block forever (preventing the correct processes from terminating) when there are correct processes that do not participate in the algorithm. Our work described in [Raynal and Travers 2006] investigates the wait-free set agreement problem, i.e., the case where the correct participating processes have to decide a value whatever the behavior of the other processes (i.e., the processes that crash and the processes that are correct but do not participate in the algorithm). The paper presents a wait-free set agreement algorithm. This algorithm is based on a leader failure detector class that takes into account the notion of participating processes. Interestingly, this algorithm enjoys a first class property, namely, design simplicity.

4.8. Scalability issues in wait-free computing

The scalability issue is very easy to state when one is interested in wait-free protocols: How to wait-free implement a concurrent object (such as a counter or a snapshot object) in presence of process crashes when we know neither the maximal number of participating processes nor an upper bound of it? Moreover, the cost of such algorithms should be sub-linear with respect to the current number of processes.

This difficult problem has received attention only very recently [Aguilera 2004]. It requires the statement of appropriate distributed models that deal with an a number of processes that can be a priori infinite [Merritt and Taubenfeld 2000].

5 – Cooperation Incentive Schemes

Introduction

Decentralized system algorithms and protocols have recently received a lot of interest in mobile ad-hoc networks as well as in peer-to-peer (P2P) systems. The development of such techniques is a necessity to be able to implement cost-effective and reliable applications deployable on a large scale, yet it brings up far-reaching issues that have to be dealt with. In decentralized systems, decision-making may not be located at a specific and central group of devices (repeaters, bridges, routers, gateways, servers) but can be distributed to end-user devices. Decisions and actions may use the computing power, bandwidth, and disk storage space of all the participants in the network rather than being concentrated in a relatively low number of special devices. The decentralized structure makes it possible to achieve minimal administrative and operational costs. Participants in this type of system are "peers" in the sense that they normally have equivalent responsibilities and privileges. The intricate notions of self-organization and self-management require that each peer provide its own contribution towards the correct operation of the system.

The handing of basic mechanisms of the system over to autonomous peers raises new concerns, in particular with respect to the establishment of trust between peers, to the stimulation of their cooperation, and to the fairness of their respective contributions. Self-organization opens up new security breaches because a peer must be able to defend against others perpetrating new forms of denial of service. Selfishness, as illustrated by the so-called free-riding attack, is a first type of such threats in which the attacker (called free-rider) benefits from the system without contributing its fair share. Systems vulnerable to free-riding either run at reduced capacity or collapse entirely because the costs of the system weigh more and more heavily on the remaining honest peers encouraging them to either quit or free ride themselves. Flooding is a second type of denial of service: the attack can be launched by sending a large number of query messages asking for resources to a victim peer in order to slow it until it is unusable or crashes. For example, an attacker can attempt to make a lot of read and write operations in a distributed storage application. Cheating (or retention) is a third form of denial of service in which the attacker retains data required for the system to work or does not comply with the normal course of action in order to obtain an unfair advantage over other peers. So-called "cooperation enforcement" mechanisms (which should more properly be called cooperation incentive schemes) provide ways of managing and organizing resources, and aim at dealing with the security challenges that traditional security approaches (e.g., authentication, access control) can not cope with.

The following sections introduce motivating applications for cooperation incentives, then detail how incentive schemes work, and finally discuss how these schemes may be validated.

5.1 Applications

Cooperation incentive mechanisms are present in various application domains. It is generally suggested that cooperation will help entities to succeed better than via competition. [Buttyán and Hubaux 2003] demonstrated that the best performance in mobile ad-hoc routing is obtained when nodes are very cooperative. In a cooperation incentive mechanism, cooperative behavior should be more beneficial than an uncooperative behavior. The two main categories of incentives are reputation and remuneration. This section describes several applications that benefit from cooperation enforcement.

5.1.1 Infrastructure based P2P applications

Infrastructure based P2P applications (sometimes termed P2P networks or even simply P2P) have become famous in several domains: file sharing is the flagship of such applications, yet other applications exist like the enabling of P2P file systems, or file backup systems.

5.1.1.1 File sharing

Peer-to-peer file sharing has become so widespread over the Internet that it now accounts for almost 80% of total traffic [Bolton and Ockenfels 2000] .

The Napster[1] protocol was historically the first to provide this service. Napster is based on a hybrid peer-to-peer infrastructure in which the index service is provided centrally by a coordinating entity, the Napster server. The functionality of the server is to deliver to a requesting peer a peers' list having the desired requested MP3 files. Then, the peer can obtain the respective files directly from the peer offering them.

In contrast, Gnutella[2] functions without any central coordination authority. Search requests are flooded into the network until the TTL (Time-To-Live hop counter) of the message has expired or the requested file has been located. Positive search results are sent to the requesting peer who can then download the file directly from the peer offering it. Both Napster and Gnutella focus more on information retrieval than on publishing.

Freenet [Cox and Noble 2002] provides anonymous publication and retrieval of data. Anonymity is provided through several means encrypted search keys and source-node spoofing. In Freenet, when peer storage is exhausted, files are deleted according to the least-recently-used principal so the system keeps only the most requested documents. Another drawback is the complexity of file search process. In fact there is a significant difference between Freenet and the systems presented so far which is that files are not stored on the hard disk of the peers providing them, but they are intentionally stored at other locations in the network. They are stored at peers having the numerically closest identification number to their IDs. The document lookup is a routing model based on keys to locate data similarly to Distributed Hash Tables (DHT). Free riding has been notably observed in such applications, and first attempts at using reputation incentives to counter it were made in systems like NICE [Lee et al. 2003].

NICE is a platform for implementing cooperative distributed applications, in particular P2P applications. The NICE system aims at identifying the existence of cooperative peers; it claims to efficiently locate the minority of cooperating users, and to form a clique of users all of whom offer local services to the community. The system is based on peer reputation which is stored in the form of cookies. These cookies

[1] http://www.napster.com/
[2] http://www.gnutella.com/

provide a signed acknowledgment that a peer did (or did not) correctly provide the resources it promised, and are stored on that very peer. Whenever a server peer interacts with a client peer, the client will retrieve the cookie he previously sent to the server; if no interaction happened before, he might obtain some information out of the cookies stored by other client peers it may know.

5.1.1.2 Distributed file system

A generation of P2P applications uses the promising DHT-based overlay networks. DHTs such as CAN, Chord, Pastry, and Tapestry allow a decentralized, scalable, and failure-tolerant storage. Well-known approaches are PAST [Druschel and Rowstron 2000] based on Pastry and OceanStore [Kubiatowicz et al. 2000] based on Tapestry. Each PAST node can act as a storage node and a client access point. These schemes have basic similarities in the way they are constructed. Participants receive a public/private key pair. Keys are used to create an unambiguous identification number for each peer and for the stored files with the aid of a hash function. To use the storage, a peer has to pay a fee or to make available its own storage space. Key generation, distribution and monitoring are handled by "special" peers who have to be highly capable and highly available. Both PAST and OceanStore aim at ensuring a high data availability through means of file replication and random distribution of the identification numbers to peers. The procedure guarantees geographically-separated replicas which increases the availability of a given file.

Compared with PAST and OceanStore, Free Haven [Dingledine 2000] is designed for more anonymity and persistence of documents than for frequent querying. An author in Free Haven generates a public/private key pair, signs his document fragments, and uploads them into the server. Each server hosts data from the other servers in exchange for the opportunity to store data of its own into the community of servers, servnet. Trading of document fragments adds to author anonymity. When a reader wishes to retrieve a document from the servnet, he requests it from any server, including a location and key which can be used to deliver the document in a private manner. This server broadcasts the request to all other servers, and those which are holding shares for that document encrypt them and deliver them to the reader's location.

5.1.1.3 Data backup

The latest generation of peer-to-peer systems is a generation of storage systems having data backup as its primary function. Pastiche [Cox and Noble 2002] is based on Pastry for locating nodes and exploits excess disk capacity to perform peer-to-peer backup with no administrative costs. Each Pastiche node minimizes storage overhead by selecting peers that share a significant amount of data. It replicates its archival data on more than one peer. Most of these replicas are placed nearby to ease network overhead and minimize restoration time. To address the problem of storing data on malicious nodes, Pastiche uses a probabilistic mechanism to detect missing backup state by periodically querying peers for stored data. However it sacrifices a fair amount of privacy because nodes can grab some information about the backup data. This issue is less critical for the CIBS (Cooperative Internet Backup Scheme) [Lillibridge et al. 2003] scheme where fragments of a file are stored at different geographical locations, and partners are tracked by a central server. To ensure a high reliability, the scheme adds redundancy through Reed-Solomon erasure correcting code.

5.1.2 Wireless networks

Collaboration does not only benefit infrastructure based applications, but also proves essential in several areas of wireless networks. This section gives three examples of applications that critically depend on

cooperation incentives to be effectively enabled: mobile ad-hoc routing wireless ad-hoc backup, and wireless data dissemination.

5.1.2.1 Mobile ad-hoc routing

Multi-hop ad-hoc networks, frequently referred to under the term MANETs (Mobile Ad hoc NETworks), can be set up rapidly and spontaneously. Connections are possible over multiple nodes. These nodes operate in a decentralized and self-organizing manner and do not rely on a fixed network topology. Intermediate nodes in a route have to act as routers to forward traffic towards its destination. To achieve this operation, incentives for cooperation between nodes become a requirement, because rational users would rather preserve the energy of their personal devices rather than spend it on cooperative routing. There has been a wealth of work on cooperative network forwarding.

In the Watchdog/Pathrater [Marti et al. 2000] scheme, the watchdog detects non-forwarding nodes by overhearing the transmission, and the pathrater keeps a rating of every node and updates it regularly. The two components enable nodes to route messages avoiding misbehaving nodes in their route. Misbehaving nodes are detected and avoided in the routing path but not punished.

In CONFIDANT (Cooperation Of Nodes, Fairness in Dynamic Ad-hoc NeTworks) [Buchegger and Le Boudec 2002], the response to misbehaving nodes is more severe than just avoiding them for routing; it also denies them cooperation. Similarly to Watchdog/Pathrater, in CONFIDANT reputation is self-carried by nodes. Nodes monitor their immediate neighborhood and also gather second-hand information from others. By Bayesian estimation, they classify other nodes as normal or misbehaving.

In CORE (COllaborative REputation) [Michiardi and Molva 2002], the information collected is classified into subjective reputation (direct information), indirect reputation (positive reports from other nodes), and functional reputation (task-specific information). The combined reputation value is used to make decisions regarding a given node, that is, to either cooperate with it or gradually isolate it.

TermiNodes ([Buttyán and Hubaux 2001]) uses a different approach based on a tamper-proof security module for each node maintaining a nuglet counter. When the node wants to send a packet, it decreases its nuglet value by a number of credits proportional to the estimated number of intermediate nodes in the route. When the node forwards a packet, its nuglet purse becomes bigger.

While TermiNodes uses a tamper-proof hardware placed at each node, Sprite [Zhong et al. 2003] does not require any tamper-proof hardware at any node. Sprite is based on a central Credit Clearance Service (CCS). Every node is supposed to have a digital ID obtained from a Certification Authority (CA). When a node receives a message, the node keeps a receipt of the message. Sprite assumes that every source node knows the entire path to the destination node through a secure ad hoc routing protocol based upon DSR. The underlying ad hoc routing protocol only exists for packet delivery, not for routing decision making. When the node has a fast connection to the CCS, which is reachable via an overlay network, it reports to the CCS the messages that it has received/forwarded by uploading its receipts. Depending on the reported receipts of a message, the CCS then determines the charge and credit to each node involved in the transmission of a message. Zhong et al. introduce a formal model in order to prove the effectiveness of Sprite in restraining selfish behavior at the network layer, however Sprite presents a weakness for it relies on the accessibility of the CCS.

5.1.2.2 Wireless ad-hoc backup

The Flashback [Loo et al. 2002] application is a first example of wireless ad-hoc backup system, which has been proposed for Personal Area Networks (PAN). Flashback is a solution designed to provide peer-to-peer power-aware backup for self-managing, mobile, impoverished devices. In Flashback, each device has an identifier assigned out-of-band during the installation of the Flashback. To ensure that devices only participate in the PAN to which they are assigned, a lightweight certificate-based authentication scheme is used. In order to keep track of replicas, Flashback maintains a replica table in persistent storage. It uses an opportunistic model to replicate local data according to the constraints imposed by the power and storage resources.

The MoSAIC project[3] [Killijian et al. 2004] is another example of data backup system and recovery service based on mutual cooperation between mobile devices. Such a service aims to ensure availability of critical data managed by wireless mobile devices. Data availability is ensured by replicating data. To finely control the level of data redundancy, MoSAIC uses an erasure coding technique for the production of redundant fragments [Killijian et al. 2006]. In contrast with Flashback, which assumes a single authority – the user – preinstalling identity certificates on every device to be backed up, MoSAIC targets ephemeral and self-organizing networks that come into existence spontaneously by virtue of physical proximity of mobile devices and where peers are mutually suspicious. A cooperation incentive scheme is critical in the MoSAIC system.

5.1.2.3 Nomadic computing

Nomadic computing aims at offering end users access to data or information from any device and network while they are mobile. There are two fundamental approaches to these applications: the first one, as proposed by the DataMan project[4] or by the Ubibus prototype [Banâtre et al. 2004], assumes the availability of information servers (called info-stations), placed at fixed positions (e.g., traffic lights, building entrances, and airport lounges), and that supply mobile users with contextual information; the second one assumes that data are distributed among neighboring mobile users in a cooperative fashion. 7DS [Papadopouli and Schulzrinne 2001] is an illustration of the latter approach: this system allows a peer to browse the content of the cache of another peer in order to search for URLs or keywords. This operation can be performed either on-demand or in a prefetching mode. When prefetching, 7DS anticipates the information needs of the peer, while on-demand retrieval only searches for information when the peer requests it. Mobile devices therefore do not need any base stations to gain access to the service in 7DS. However, while this system relies on the cooperation of nodes, users are not encouraged to provide storage space for information caching, nor to answer neighbor requests, nor even to provide accurate answers. History-based credentials used in context-aware applications like [Bussard et al. 2004] for instance provide means to implement a self-carried reputation scheme, thus aiming at ensuring such a cooperation of users or devices in nomadic computing applications. One-time capabilities [Bussard and Molva 2004] also aim at discouraging non-cooperation but using a remuneration based punishment scheme.

[3] The MoSAIC Project, project partners: Institut Eurécom, IRISA, LAAS. http://www.laas.fr/mosaic/
[4] DATAMAN, http://planchet.rutgers.edu/~badri/dataman/research-projects.html

5.1.3 Web commerce

Cooperation incentives are often used in web commerce sites. This section presents some examples of such incentives: auction sites, and review or recommendation sites. All schemes are reputation systems where the reputation is computed either using a voting scheme or using the average of ratings.

5.1.3.1 Auction sites

Auction sites allow sellers to list items for sale, buyers to bid for these items, then the items to be sold to the highest bidder. In general, the person who puts the item up for auction pays a fee to the auctioneer. In some cases, there is a minimum or reserve price; if the bidding does not reach the minimum, the item is not sold. Reputation systems are used in auctioning in order to help users making good choices when selecting transacting partners. eBay[5] is one popular online auction site. The feedback forum on eBay allows sellers and buyers to rate each other as positive, negative, or neutral. Ratings of buyers and sellers are conducted after the completion of a transaction, which is monitored by eBay. The reputation system relies on a centralized repository that stores and manages ratings. The overall reputation of a participant is the sum of ratings about him over the last 6 months. eBay also provides one month old and seven day old ratings to let users know about recent behavior of the participant. The eBay system makes it possible to perform fake transactions. Even though, this incurs a cost, since eBay charges a fee for listing items: this still opens up opportunities to acquire undue ratings.

5.1.3.2 Review and recommendation sites

In review sites, individual reviewers, who are generally individuals, provide information to fellow consumers. In these systems, a reputation rating is applied to both products and reviewers themselves, in particular to discourage product bashing. One example of such a system is Amazon[6], an online bookstore that allows members to write book reviews. A user can become an Amazon member by simply signing up. Reviewers' reviews of a book are made of some text and a rating in the range of 1 to 5 stars. Members and users rate reviews as being helpful or not. Amazon ranks reviewers based on their rating and other parameters (which are not publicly revealed). Reviewers with a high ranking are given the status of top reviewers. To reduce repetitive ratings from the same users, Amazon only allows one vote per registered cookie for any given review. Epinions[7], a similar review site charges product manufacturers and online shops by the number of clicks that consumers generate as a result of reading about their products. This makes it possible for top reviewers to get paid, while in contrast, Amazon does not give any financial incentive for well-reputed reviewers.

5.2 Incentive Schemes

Cooperation is a central feature of decentralized systems, and even more so ad-hoc ones, to compensate for the lack of a central and dedicated entity and still achieve some general function. However, cooperation to achieve some functionality may be hampered by the fact that users have full authority on their devices and, as proven by experience, will on average try to maximize the benefits they get from the network. In general, the cooperative behavior of a device will indeed result in an increase in its resource consumption or missed

[5] http://ebay.com/
[6] http://www.amazon.com/
[7] http://www.epinions.com/

opportunities to take more than its fair share of a resource (e.g. network, CPU, storage space). In case of mobile ad hoc forwarding for instance, the node forwarder is confronted with additional energy and bandwidth usage for reception and transmission of packets, as well as with the increase of computational resource consumption. Knowing that mobile devices have inherently scarce resources, each of these devices should better not cooperate from its point of view. In case of file sharing applications, a node can take advantage of the system by downloading files without contributing to it. To counterbalance this, and achieve an overall better result, it is primordial to design incentive mechanisms for cooperation that discourage uncooperative behaviour, be it passive or malicious. At the same time, these mechanisms can not prevent the non cooperative behavior of devices due to valid and reasonable reasons (e.g., crashing, energy shortage, route breaks), which should normally not be punished as if they were malicious non-cooperation.

As seen in section B, there are many cooperation incentive schemes which are diverse not only in terms of the applications for which they are useful or critical, but also in terms of the features they implement, the type of reward and punishment used, and their operation over time. [Obreiter and Nimis 2003] classifies cooperation enforcement mechanisms into trust-based patterns and trade-based patterns. The authors make a distinction between static trust, thereby referring to pre-established trust between peers, and dynamic trust, by which they refer to reputation-based trust. Oreiter et al. analyze trade-based patterns as being based either on immediate remuneration, which they term barter trade, or on deferred remuneration, which they term bond-based. While this classification was the first to try to address so many different incentive schemes together, other authors describe cooperation only in self-organized systems, in which case they classify cooperation schemes into reputation based (*"dynamic trust-based"* for Obreiter et al.) and remuneration based (*"trade-based"*) approaches. Trust establishment, a further step in many protocols, easily maps to reputation systems but may use remuneration systems as well. The following sections use the reputation versus remuneration classification.

5.2.1 Reputation based mechanisms

In reputation-based mechanisms, the decision to interact with a peer is based on its reputation. Reputation mechanisms need reputation management systems for which the architecture is either centralized, or decentralized, or both.

5.2.1.1 Reputation based system architecture

The estimation of reputation can be performed either centrally or in a distributed fashion. In a centralized reputation system, the central authority that collects information about peers typically derives a reputation score for every participant and makes all scores available online. In a distributed reputation system, there is no a central authority for submitting ratings or obtaining reputation scores of others. However, it might be some kind of distributed storage where ratings can be submitted. One example of such architecture is FastTrack [Liang et al. 2006] architecture which is used in P2P networks like KaZaA[8], Grokster[9], and iMesh[10]. These networks have two-tier hierarchy consisting of ordinary nodes (ONs) in the lower tier and supernodes (SNs) in the upper tier. SNs are generally more powerful in terms of connectivity, bandwidth, processing and non-NATed (Network Address Translation) accessibility. SNs keep tracks of ONs and other SNs and act as directory servers during the search phase. Such an architecture can be convenient to manage peer reputations using supernodes as distributed storage; unfortunately this is not the case in the existing

[8] http://www.kazaa.com/
[9] http://www.grokster.com/
[10] http://imesh.com

FastTrack-based P2P networks. In KaZaA for example, each node has a participation level based some QoS (Quality of Service) parameters that is stored locally. The participation level score is used in prioritizing peers during periods of high demand. Most of the time in a distributed architecture, ratings are estimated autonomously by each peer. Each peer records ratings about its experiences with other peers and/or tries to obtain ratings from other parties who have had experiences with a given target peer. A good example of a decentralized reputation-based approach to trust management is NICE [Lee et al. 2003]. This system searches the network at runtime and builds a trust graph where each edge represents how much the source trusts the destination. A reputation value is calculated based on this trust graph. Then, the NICE algorithm selects a trust path based on whether it is the strongest path or using a weighted sum of strongest disjoint paths.

The centralized approach to reputation management is not fault-tolerant. In the decentralized approach, it is often impossible or too costly to obtain cooperation evaluations resulting from all interactions with a given peer. Instead reputation is based on a subset of such evaluations, usually obtained from the neighborhood. The reputation mechanism should therefore be designed such as to avoid inconsistencies. Distributed reputation management systems are most probably a more appropriate design than centralized ones to achieve a scalable solution to cooperation incentives. A distributed algorithm will allow applications to scale up to a large community of users by making it possible to have local standard of reputation. Some applications or networks being typically decentralized, like wireless ad hoc networks, and especially MANETs, they require the use of a reputation management system itself decentralized.

5.2.1.2 Reputation system operations

A reputation-based mechanism is composed of three phases (Figure 5.1):

1. Collection of evidence: Peer reputation is constructed based on the observation of the peer, experience with it, and/or recommendations from third parties. The semantics of the information collected can be described in terms of a specificity-generality dimension and a subjectivity-objectivity dimension:

- Specific vs. general information: specific information about a given peer relates to the evaluation of a specific functionality or aspect of this peer such as its ability to deliver a service on time. Whereas, general information refers to all functionalities (e.g., measured as a weighted average).

- Objective vs. subjective information: a peer obtains objective information (also known as direct or private information) about a given peer through his personal but concrete interactions with the considered peer, and subjective information (also known as indirect or public information) by listening to messages or negotiations that are intended to other peers, or by asking neighboring peers their opinions about the actual peer.

2. Cooperation decision: Based on the collected information, a peer can make a decision whether he should cooperate with another peer, based on the reputation of that other peer. There exist a variety of methods for computing the reputation of an entity, some of which being described below:

- Voting scheme: the simplest method to compute reputation is to compute the sum of positive ratings minus the sum of negative ratings. This is the principle used in eBay[11]'s reputation forum.

[11] http://ebay.com

- Average of ratings: another simple scheme is to compute the reputation score as the average of all ratings given by peers or users. The Amazon[12] recommendation system uses such a scheme (although it does not provide standard deviation and may therefore be less precise than eBay's reputation system).

- Bayesian based computation: reputation is computed based on the previous estimated reputation with the new evaluation score. Reputation systems (like [Jøsang and Ismail 2002] and [Mui et al. 2001]) use the beta PDF (Probability Density Function) denoted by *beta(p |α , β)* using the gamma function *Γ*.

$$beta(p \mid \alpha, \beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)+\Gamma(\beta)} p^{\alpha-1}(1-p)^{\beta-1}; 0 \le p \le 1, \alpha > 0, \beta > 0$$

  Where α and β represent the amount of positive and negative ratings respectively, and p represents the probability variable. The PDF expresses the uncertain probability that future interactions will be positive (cooperation).

- Flow model: "systems that compute trust or reputation by transitive iteration through looped or arbitrarily long chains can be called flow models" [Jøsang et al. 2005]. One example of such a reputation computing function (although not a cooperation measurement) is PageRank [Page et al. 1998], Google[13]'s algorithm to rank web pages[14] based on their "reputation" (number of its referrals). The page rank of a web page *u* is defined as:

$$R(u) = cE(u) + c \sum_{v \in N^-(u)} \frac{R(v)}{|N^+(v)|}$$

  where $N^-(u)$ denotes the set of web pages pointing to *u*, $N^+(v)$ denotes the set of web pages that *v* points to, and *E* corresponds to a source of rank. A hyperlink is a positive referral of the page it points to, and negative referrals do not exist because it is impossible to blacklist a web page using the above equation. EigenTrust [Kamvar et al. 2003] is another flow model that computes a global trust value for a peer by multiplying iteratively normalized local matrices of trust scores of each peer in the system. With a large number of matrices, the system will converge to stable trust values. In this model, trust and reputation are evaluated similarly.

- Other computation models are described in [Jøsang et al. 2005]: *(a) the discrete trust model*, in which the trustworthiness of the neighbor is taken into account before considering subjective information; *(b) the belief model*, which relates to the probability theory, and in which the sum of probabilities over all possible outcomes does not necessarily add up to 1, the remaining probability being interpreted as uncertainty; *(c) fuzzy models* finally, in which reputation and trust are considered as fuzzy logic concepts.

3. Cooperation evaluation: The occurrence of interaction with a peer is conditional on the precedent phase. After interaction, a node must provide an evaluation of the degree of cooperation of the peer involved in the

[12] http://www.amazon.com/
[13] http://www.google.com/
[14] The public PageRank measure does not fully describe Google's page ranking algorithm, which takes into account other parameters for the purpose of making it difficult or expensive to deliberately influence ranking results in what can be seen as a form of "spamming".

interaction. Peers performing correct operations, that is, behaving cooperatively, are rewarded by increasing their local reputation accordingly. A peer with a bad reputation will be isolated from the functionality offered by the group of peers as a whole. The evaluation of the current interaction can convey extra information about other past interactions (piggybacking) that can be collected by the neighboring peers.

5.2.1.3 Attacks and counter-measures

Cooperative mechanisms have to cope with several problems due to node misbehavior. Misbehavior ranges from simple selfishness or lack of cooperation to active attacks aiming at denial of service (DoS), attacks to functionality (e.g., subversion of traffic), and attacks to the reputation system (liars).

To guard against the impact of liars, the CORE mechanism [Michiardi and Molva 2002] for instance takes into account only positive reputation from indirect information, together with reputation from direct information (does the node hear the packet forwarded by its peer?): defamation is thus avoided, yet unjustified praising is still possible. In a more restrictive manner, RPG (Reputation Participation Guarantee) [Barreto et al. 2002] forbids the diffusion of reputation between peers. Only direct information is taken into account; selfishness is detected by sending probe packets.

A different approach, relying on indirect information, is taken in Watchdog/Pathrater [Marti et al. 2000]. A watchdog is in charge of identifying the misbehaving nodes, and a pathrater is in charge of defining the best route avoiding these nodes. It is pretty much the same approach that is taken in CONFIDANT [Buchegger and Le Boudec 2002]: a neighborhood monitor has the role of identifying misbehavior, which is rated.

A trust manager sends and receives alarm messages to and from other trust managers, while a path manager maintains path ranking. As a result, nodes in the network will exclude misbehaving nodes by both avoiding them for routing and by denying them cooperation. So, misbehaving nodes will be penalized by being isolated. Contrary to many proposals, Watchdog/Pathrater evaluates cooperation but does not enforce it: non cooperative nodes in Watchdog/Pathrater will not be punished like in CORE or CONFIDANT, their messages are still forwarded while they are not forced to forward the messages of the other nodes.

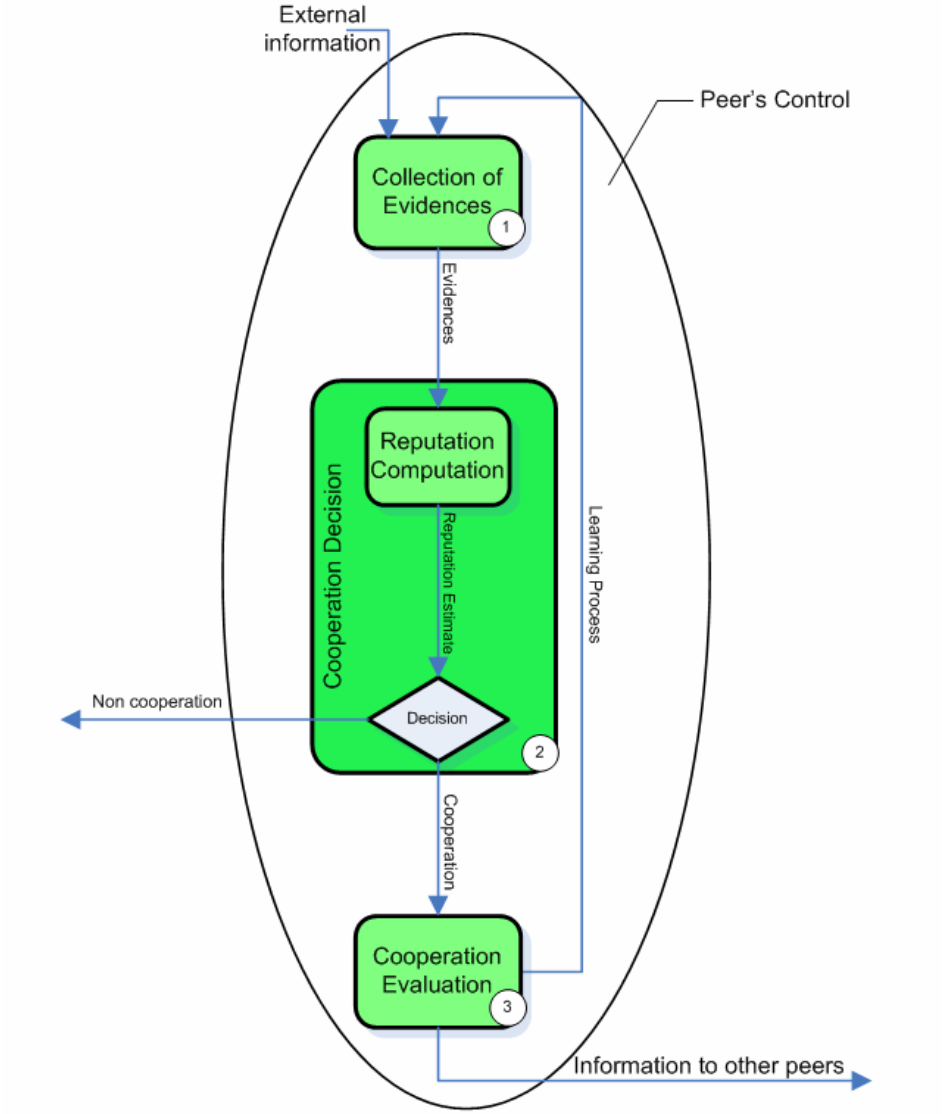

Figure 5.1: Reputation-based Mechanism

The systems presented so far focus on network layer forwarding. In this type of application, cooperation evaluation is immediate, yet other mechanisms may require the evaluation to take place on a longer timescale. Reputation estimates then need to be preserved: this may mean that it is self-carried by the peer if reputation is based on direct information; otherwise, reputation should not depend on the proximity of the peer since nearby nodes are likely to move away over a long period of time. This is for instance the case for distributed backup applications.

In the CIBS (Cooperative Internet Backup Scheme) scheme [Lillibridge et al. 2003], each computer has a set of geographically-separated partner computers that collectively hold its backed up data. In return, the computer backs up a part of its partner's data. To thwart free riding attacks, a computer can periodically challenge each of its partners by requesting him to send a block of the backed up data[15]. An attack can then

[15] Some disruption attacks, i.e. attacks aiming at disrupting, impairing or destroying a system or a particular user, can be avoided by limiting reads to mutually chosen random blocks.

be detected and the data blocks of the attacker that are stored in the attacked computer are consequently dropped. In this scheme, each peer takes note of its direct experience with a partner, and if this partner does not cooperate voluntarily or not beyond some threshold, the peer may decide to establish a backup contract with a different partner.

Another example of reputation-based mechanisms for distributed storage is the Free Haven project [Dingledine 2000]. The overall design of the project is based on a community of servers, called the servnet where each server hosts data from other servers in exchange of the opportunity to store data of its own in the servnet. The incentives for cooperation are based on a reputation mechanism. A trust module on each server maintains a database of each other server, logging past direct experience as well as what other servers have said.

5.2.2 Remuneration based mechanisms

In contrast to reputation-based mechanisms, remuneration based incentives are an explicit counterpart for cooperation and provide a more immediate penalty to misconduct. Remuneration brings up requirements regarding the fair exchange of the service for some form of payment [Asokan et al. 1997]. This requirement in general translates to a more complex and costly implementation than for reputation mechanisms. In particular, remuneration based mechanisms require trusted third parties (TTP) such as banks to administer remuneration of cooperative peers; these entities do not necessarily take part in the online service, but may be contacted in case of necessity to evaluate cooperation. Tamper proof hardware (TPH) like secure operating systems or smart cards have been suggested or used to enforce in a decentralized fashion the fair exchange of the remuneration against a proof that the cooperative service was undertaken by a peer node.

5.2.2.1 Remuneration based system architecture

A remuneration based mechanism comprises four main operations (see Figure 5.2):

- **Negotiation:** The two peers may often negotiate the terms of the interaction. Negotiating the remuneration in exchange for an enhanced service confers a substantial flexibility to the mechanism. The negotiation can be performed either between the participating peers or between peers and the authority.
- **Cooperation decision:** The peer in a self-organizing network is always the decision maker. During negotiation and based on its outcome, a peer can decide if it is better to cooperate or not.
- **Cooperation evaluation:** Cooperation has to be evaluated by the service requesting party, in terms of adequacy of the service to the request, as well as by the service providing party, in terms of adequate remuneration. Ensuring the fairness of both evaluations may ultimately require involving a trusted third party. Depending on the service, this TTP will ensure a fair exchange for every interaction, or may only be involved if arbitration is requested by one party (see below). The TTP, which may be centralized or distributed itself, may for instance give access to information unavailable to a peer, or more generally provide a neutral execution environment.
- **Remuneration:** The remuneration can consist in virtual currency units (a number of points stored in a purse or counter) or real money (banking and micropayment), or bartering units (for instance quotas defining how a certain amount of resources provided by the service may be exchanged between entities). The latter can even be envisioned in the form of micropayments [Jakobsson et al. 2003]. Regarding real money, this solution assumes that every entity possesses a bank account, and that banks are enrolled in the cooperative system, directly or indirectly through some payment scheme. The collaborating peer is remunerated by issuing a check or making a transfer of money. In the first case, remuneration implies a number of points added to a counter connected with the collaborating peer. The remuneration can be guaranteed at once or only after a certain number of steps (deposit, remuneration for data storage, remuneration for data retrieval…).

These operations can be used repeatedly to perform some cooperative service on a finer granularity basis, which may ease cooperation enforcement. In particular, micropayment is often envisioned rather than an actual (macro-)payment in remuneration based cooperation enforcement mechanisms.

5.2.2.2 Fair exchange

As mentioned in [Asokan et al. 1997], "*many commercial transactions can be modeled as a sequence of exchanges of electronic goods involving two or more parties. An exchange among several parties begins with an understanding about what item each party will contribute to the exchange and what it expects to receive at the end of it. A desirable requirement for exchange is fairness. A fair exchange should guarantee that at the end of the exchange, either each party has received what it expects to receive or no party has received anything*." Fair exchange protocols thus provide ways to ensure that items held by two or more parties are exchanged without one party gaining an advantage. In remuneration systems, obtaining an efficient cooperation incentive depends upon devising a protocol that enforces a fair exchange of the remuneration (virtual or not) against some task. This property can only be attained by intricately integrating the remuneration operation with the application functionality. Fair exchange protocols rely on the availability of a trusted (and neutral) third party (TTP) caring for the correctness of the exchange. Two types of protocols should be distinguished: online protocols, which mediate every interaction through the TTP, which can lead to performance and reliability problems with the TTP constituting a bottleneck as well as a single point of failure; offline ones, also called optimistic fair exchange protocols, which resort to the TTP intermediation only if one of the parties wants to prove that the exchange was not fairly conducted.

The TermiNodes project ([Buttyán and Hubaux 2001]) addresses the security of the networking function of packet forwarding through remuneration schemes. Each device possesses a security module that manages its account by maintaining a counter called nuglet, interpreted as virtual money. The project proposes two models for remuneration aiming at enforcing fair exchange for stimulating a cooperative behavior. In the first one, called Packet Purse Model, each packet carries a given number of nuglets and intermediate nodes get paid with some nuglets, which get removed from the packet purse when forwarded by the node. In the second model, called Packet Trade Model, each intermediate node buys packets from the previous node on the route then sells them to the next node for more nuglets, until the destination node, which finally pays the total cost of forwarding packets.

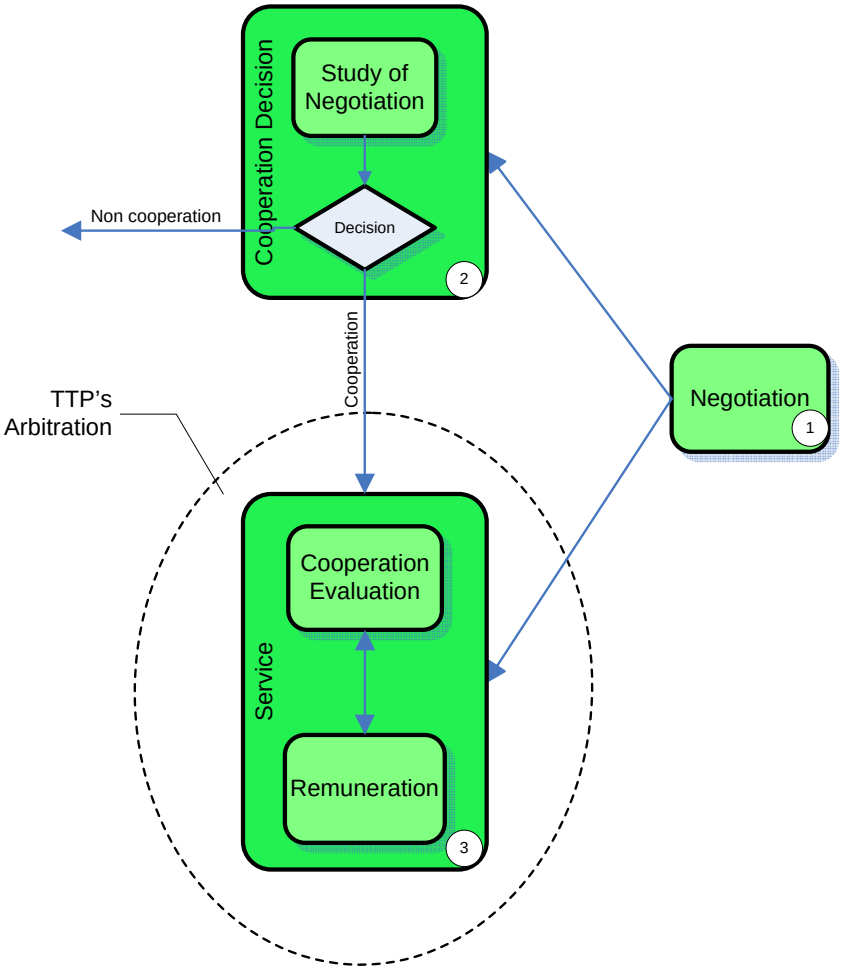

Figure 5.2: Remuneration-based Mechanism

The architecture of Sprite [Zhong et al. 2003], a credit-based system for stimulating cooperation among selfish nodes in mobile ad hoc networks, is not very different from TermiNodes except for the fact that it does not use security modules. It consists of a Credit Clearance Service (CCS) and of mobile nodes. In Sprite, a node transmitting its own messages loses some credits (i.e., virtual money paid by the node to the CCS), which will be used to cover the costs for packet forwarding by intermediate nodes. In order to earn credits, a node must transmit the CCS receipts of forwarded messages. The system does not guarantee balanced payments, i.e., it does not require that the sender's total debt equal the total credit received by intermediate nodes for their forwarding activity. In fact, to prevent cheating behavior, the CCS debits the sender with a higher amount than that due to intermediate nodes; only later does the CCS uniformly share the exceeding credit among nodes, or give a fixed credit amount to each node. Sprite focuses on combating cheating behavior and on promoting cooperation among network nodes; it does not prevent active attacks on the system (e.g., Denial of Service attacks).

Akin to Sprite, Mojo Nation [McCoy 2001], a content distribution technology, is built on a micropayment system. A common digital currency, the Mojo, is used to buy disk space, bandwidth, and processing cycles contributed to the system. Peers who have contributed resources to the system are credit for their exact participation. Interaction between peers across the network involves the exchange of Mojo currencies. A TTP ensures honest transfers between agents within the network.

In contrast, in OceanStore [Kubiatowicz et al. 2000], the remuneration of cooperative peers is monetary as the service is envisioned to be provided by a confederation of companies. In exchange for economic compensation, computers joining the system contribute with storage or provide access to local users. Each user is supposed to pay a fee to one particular provider who buys storage space from and sells it to other providers. Legal contracts and enforcement can be used to punish peers that do not keep their end of the bargain, based on planned billing and auditing systems.

Networking context		Application	Incentives for cooperation		No Incentive for cooperation (or closed system)
			Reputation-based incentives	Remuneration-based incentives	
Infrastructure-based	P2P network	File-sharing	NICE	Mojo Nation	Napster[16], Gnutella[17], Freenet
		File system	Free Haven	PAST, OceanStore	
		Backup	CIBS		Pastiche
Wireless	Ad hoc network	Packet forwarding	CORE, RPG, Watchdog/pathrater, CONFIDANT	TermiNodes	Ad hoc IEEE 802.11 standards[18]
		Backup	MoSAIC[19]	FlashBack	
	Nomadic computing	Nomadic computing	History-based certificates	One-time capabilities	Ubibus, 7DS
Centralized network		Web commerce	Ebay[20], Amazon[21], Epinions[22]		Most review sites

Table 5.1: Cooperation enforcement schemes in various applications

Smart cards (or similar forms of tamper-proof hardware like secure operating systems) have been proposed for some time now as a means to implement an optimistic fair exchange protocol. The use of smart cards is especially interesting since it provides both the convenient form factor of a personal token and ideally a perfect implementation of a secure purse. Smart cards also offer a tamper-resistant area for a trusted third party to store secrets, or implement critical security functions, in particular for revoking the user from the system. [Vogt et al. 2001] for instance proposes the use of one card for all parties involved in a transaction and focuses on providing a neutral platform for enforcing fair exchange. Another solution is to use one card for each party [Terada et al. 2004], which aims at reducing the number of messages exchanged; this solution may also be interesting, although this benefit is not mentioned by the authors, for gaining a better

16 http://www.napster.com/
17 http://www.gnutella.com/
18 http://grouper.ieee.org/groups/802/11/
19 The MoSAIC (http://www.laas.fr/mosaic/) project partners: Institut Eurécom, IRISA, LAAS are members of ReSIST
20 http://ebay.com/
21 http://www.amazon.com/
22 http://www.epinions.com/

understanding of the liability of each party involved and thus ease the reconciliation of the data by the TTP during the online phase of the protocol, in case of a problem in the offline phase. In addition, the latter technique makes it possible to attach data like some credit to every user and let the smartcard manage this "currency" as part of the fair exchange protocol. As discussed above, however, remuneration has to be integrated with the application: this means that interactions with such hardware must be carefully thought out in the application protocol in order to prevent its bypassing or abuse: with smart cards for instance, this involves the mediation of terminals, which are distrusted with respect to remuneration handling.

A smart card based remuneration mechanism is for instance used in the peer-to-peer storage system PAST [Druschel and Rowstron 2000]. PAST is based on the Pastry routing scheme that guarantees that peers contributing to cooperative storage are geographically separated. The storage scheme relies on the use of smart cards to ensure that clients cannot use more remote storage than they are providing locally, which is optional in PAST. Smart cards are held by each PAST user and issued by a third party, and support a quota system that balances supply and demand of storage space in the system. With fixed quotas and expiration dates, users are only allowed to use as much storage as they contribute.

Table 5.1 summarizes the various approaches to cooperation and their respective features as discussed in the last two sections.

5.1 Validation techniques

In the context of self-organizing networks like for instance wireless mobile ad hoc networks, cooperative mechanisms have to be investigated in terms of performance, fairness, and resilience to attacks, as well as cooperation enforcement. Experimentation is an obvious validation approach, yet it suffers from scalability issues. Otherwise, cooperation incentives may be validated using either simulation or game theory.

5.1.1 Prototype-based evaluation

A cooperation mechanism can be validated by building a prototype, that is, a physical model of a proposed product concept that allows demonstration, evaluation, or testing of the most representative attributes and idiosyncrasies of a mechanism. Prototypes are especially important to fine tune parameterized schemes. The literature offers a lot of examples of P2P or ad hoc cooperation mechanisms whose evaluation process was based on prototypes. To validate their incentive system for ad hoc networking, a prototype was used for Sprite [Zhong et al. 2003] to determine how much overhead was necessary for the incentive scheme and to evaluate the packet routing performance of the system (percentage of packets successfully relayed from the sender to the destination). Results show that the overhead of the Sprite system is insignificant, and that nodes in the system cooperate and forward each other’s messages unless their resources are extremely low. The Pastiche [Cox and Noble 2002] scheme was also evaluated using the prototyping approach. Pastiche's prototype consists of two main components: the chunkstore file system, implemented in user space and written in C, and a backup daemon. With the evaluation of the prototype, it was demonstrated that the backup service does not penalize the file system performance unduly and also that node discovery was effective. CIBS [Lillibridge et al. 2003] was also prototyped for the validation of the backup scheme. To measure the performance of their Internet backup scheme, its authors used a number of personal computers running instances of the prototype software. Each instance was partnered with the other instances located in different PCs so that all communication between partners went through the network. Experiments on the prototype have shown that the backup scheme performance is acceptable in practice and that the technique is feasible and cheap. Validating cooperation incentive schemes using pure experimentation however proves difficult because of scale for many applications that were considered in the previous sections. This

experimental approach has however proven quite successful with real-world evaluations in P2P file sharing systems, especially so because of their widespread usage.

5.1.2 Simulation

An alternative validation technique for cooperative systems consists in taking advantage of existing network simulators in order to obtain results for a virtual deployment on a large scale of distribution. These simulators are tailored to fit the simulation context and match the objectives of a given application by the use or development of patches and/or individual adjustments. According to [Brown and Kolberg 2006], "simulation can be defined as the process of designing a model of a real system and conducting experiments with this model for the purpose of understanding the behavior of the system and/or evaluating various strategies for the operation of the system". This means that simulating cooperation incentives with the purpose of testing their efficiency would also require simulating non-cooperative behaviors (and not only an ideal cooperative behavior).

Many applications considered beyond layer 3 require simulating overlay networks, which proves a bit difficult. Firstly, most overlay networks need to be scalable (thousands of simultaneous users) which is difficult to realize due to memory constraints even for most powerful machines. However, some tools allow a simulation to be distributed over a set of machines (distributed simulation). Additionally, it is generally desirable that a simulation behaves in accordance with real network parameters (packet delay, traffic and network congestion, bandwidth limitations etc). These considerations increase the overhead on the host machine. Packet-level network simulators such as ns-2, OMNET++[23], GloMoSim[24]/QualNet[25], and OPNET[26] must be distinguished from overlay network simulators like PeerSim[27].

Ns-2. There is no doubt that the most popular network simulator is ns (version 2)[28]. Ns-2 is an object-oriented, discrete event[29] driven network simulator developed at UC Berkeley.. The simulator ns-2 is written in C++ and OTcl. Ns-2's code source is split between C++ for its core engine and OTcl, an object oriented version of Tcl for configuration and simulation scripts. The combination of the two languages offers an interesting compromise between performance and ease of use. An ns-2 simulation scenario is a Tcl file that defines the topology and the movement of each host that participates in an experiment. Implementing a new protocol in ns-2 typically requires adding C++ code for the protocol's functionality, as well as updating key ns-2 OTcl configuration files in order for ns-2 to recognize the new protocol and its default parameters. The C++ code also describes which parameters and methods are to be made available for OTcl scripting. Debugging is difficult in ns-2 due to the dual C++/OTcl nature of the simulator. For the moment, there is only one P2P simulation available for ns-2 which is Gnutella. More troublesome limitations of ns-2[30] are its large memory footprint and lack of scalability as soon as simulations of a few hundred to a few thousand of nodes are undertaken. Ns-2 is well documented with active mailing lists.

---

[23] The OMNeT++ Community Site, http://www.omnetpp.org/

[24] Global Mobile Information Systems Simulation Library, GloMoSim, http://pcl.cs.ucla.edu/projects/glomosim/

[25] Scalable Network Technologies (SNT), http://www.scalable-networks.com/products/

[26] OPNET, http://www.opnet.com/

[27] Biology-Inspired techniques for Self-Organization in dynamic Networks, BISON Project, http://www.cs.unibo.it/bison/

[28] The network Simulator ns-2, http://www.isi.edu/nsnam/ns/index.html

[29] A discrete-event simulator is a simulator where state variables change only at discrete points in time at which events occur caused by activities and delays

[30] The network Simulator ns-2, http://www.isi.edu/nsnam/ns/index.html

OMNET++. OMNET++[31] is another discrete event simulator. It is an open-source, component-based environment with a strong focus on supporting the user with a Graphical User Interface (GUI). The simulator is very well structured and modular, modules being programmed in C++ and assembled into larger components using a high level language (NED). It is possible to simulate peer-to-peer networks with OMNET++ which can also run distributed simulations over a number of machines. OMNET++ has a rapidly increasing user base now, with lots of useful modules, an active mailing list and even workshops. Both ns-2 and OMNET++ are packet-level simulators; so scalability is also a major issue; just like ns-2, OMENT++ is more suitable for small networks.

GloMoSim[32]/QualNet[33]. GloMoSim is built as a scalable simulation environment for wireless and wired network systems. It is designed using the parallel discrete-event simulation capability provided by PARSEC (C-based simulation language). PARSEC is designed to cleanly separate the description of a simulation model from the underlying sequential or parallel simulation protocol, used to execute it. GloMoSim is built using a layered approach. Standard APIs are used between the different layers. This allows the rapid integration of models developed at different layers. To specify the network characteristics, the user has to define specific scenarios in text configuration files: app.conf and Config.in. The first contains the description of the traffic to generate (application type, bit rate, etc.) and the second contains the description of the remaining parameters. The statistics collected can be either textual or graphical. With GloMoSim, it is difficult to describe a simple application that bypasses most OSI layers. Bypassing the protocol stack is not obvious to achieve as most applications usually lie on top of it which makes the architecture much less flexible. The free GloMoSim version is for academic use only. The commercial GloMoSim-based product is QualNet. The development framework is in C/C++ and mostly provided in source form. It includes a graphic development tool for adding/revising protocols. The two simulators provide substantial support for simulation of routing protocols over wired and wireless networks.

OPNET. OPNET Modeler[34] is a commercial tool part of the many tools from the OPNET Technologies suite (Optimized Network Engineering Tools). OPNET is an event-driven scheduled simulator integrating analysis tools for interpreting and synthesizing output data, graphical specification of models and hierarchical object-based modeling. It can simulate all kinds of wired networks, and an 802.11 compliant MAC layer implementation is also provided. OPNET is a well established product used by large companies to diagnose or reorganize their networks. It can simulate wired and wireless networks. Models built with OPNET are hierarchically structured. At the lowest level, the process domain is structured as a finite state machine (FSM). The FSM can be structured with the help of a graphical editor that allows the user to specify the relation between the single states and their transitions. The single states and the transition conditions can then be programmed with a C-like language called Proto-C. Basically, the deployment process goes through the following phases. First, one has to choose and configure the node models required in the simulations, (for example a wireless node, a workstation, a firewall, a router, a web server, etc.). Then the network is built and organized by connecting the different entities. The last step consists in selecting the statistics to collect during the simulations. Most of the deployment in OPNET is done through a hierarchical GUI. OPNET scales quite well but not many data in the literature demonstrate its capabilities.

[31] The OMNeT++ Community Site, http://www.omnetpp.org/
[32] Global Mobile Information Systems Simulation Library, GloMoSim, http://pcl.cs.ucla.edu/projects/glomosim/
[33] Scalable Network Technologies (SNT), http://www.scalable-networks.com/products/
[34] OPNET, http://www.opnet.com/

PeerSim. In addition to these network simulators, there also exist simulators that only focus on overlay networks. A good example is PeerSim[35], which has been developed for large-scale overlay systems within the BISON project. It makes it possible to simulate scalable and dynamic overlays. PeerSim is written in the Java language. It is composed of two simulation engines: a cycle-based one and a more traditional event-based engine. The cycle-based engine does not model the overhead of the transport layer and subsequently is more scalable. The event-based engine is less efficient but more realistic. The simulation engines are supported by many simple, extendable, and pluggable components, with a flexible configuration mechanism.

Other simulators may be found in the literature about peer-to-peer or overlay systems, e.g., **3LS** [Ting and Deters 2003], **Query-Cycle Simulator** [Schlosser and Kamvar 2002], **Anthill** [Babaoglu et al. 2002] and **NeuroGrid[36]**. None of these simulators seems really satisfactory. 3LS, Anthill and NeuroGrid have scalability limitations. Query-Cycle is limited to file-sharing. All seem to lack enough support for dynamicity. In conclusion, ns-2[37], GloMoSim[38]/QualNet[39], OMNET++[40], OPNET[41] and PeerSim are potential candidates to build up a simulation environment for evaluating cooperation incentives in an ad hoc or P2P system. OPNET and ns-2 possess an extensive set of models, protocols and algorithms already produced, but less than OMNET++. The modular nature of OMNET++ makes it possible to carry out studies over a wide range of situations in detail. Also, regarding the ease of use and extensibility, OMNET++ appears to be the best simulator. OPNET and QualNet are also more than satisfactory with respect to this capability, however ns-2 scores poorly.

Some of the incentives schemes for cooperation listed above were investigated using a network simulator. In 7DS [Papadopouli and Schulzrinne 2001] simulation scenarios, hosts were modeled as ns-2 mobile nodes. Mobile nodes move according to the random waypoint mobility model, which is commonly used to model the movement of individual pedestrians. A waypoint model breaks the movement of a mobile node into alternating motion and rest periods. A mobile node moves at a speed uniformly chosen from an interval to a randomly chosen location where it stays for a fixed amount of time; then it chooses another random location and moves towards it, and so on. An ns-2 simulation study was also carried out for the ORION project [Klemm et al. 2003] where the performance of ORION was compared to off-the-shelf approaches based on a P2P file-sharing system for Internet, TCP, and a MANET routing protocol. In the simulated scenarios, an IEEE 802.11 standard MAC layer was used along with the standard physical layer, the two-ray ground propagation model. Ns-2 was widely employed for simulation principally in wireless mobile networks more than in P2P or ad-hoc networks where other simulators like QualNet[42] [43] were mostly adopted. CORE was evaluated with QualNet simulations [Michiardi 2004], and CONFIDANT [Buchegger and Le Boudec 2002] with GloMoSim[44] ones.

[35] Biology-Inspired techniques for Self-Organization in dynamic Networks, BISON Project, http://www.cs.unibo.it/bison/
[36] NeuroGrid, http://www.neurogrid.net
[37] The network Simulator ns-2, http://www.isi.edu/nsnam/ns/index.html
[38] Global Mobile Information Systems Simulation Library, GloMoSim, http://pcl.cs.ucla.edu/projects/glomosim/
[39] Scalable Network Technologies (SNT), http://www.scalable-networks.com/products/
[40] The OMNeT++ Community Site, http://www.omnetpp.org/
[41] OPNET, http://www.opnet.com/
[42] Global Mobile Information Systems Simulation Library, GloMoSim, http://pcl.cs.ucla.edu/projects/glomosim/
[43] Scalable Network Technologies (SNT), http://www.scalable-networks.com/products/
[44] Global Mobile Information Systems Simulation Library, GloMoSim, http://pcl.cs.ucla.edu/projects/glomosim/

	Simulator	P2P protocols	Language	Distributed simulation	Conditions
Packet-level network simulator	ns-2[45]	Gnutella	C++/OTcl	Yes	Open source
	OMNET++[46]	None	C++/NED	Yes	Academic public license
	GlomoSim[47]/ QualNet[48]	---	C/C++	Yes	Free for universities / commercial
	OPNET[49]	---	Proto-C	Yes	Commercial
Overlay network simulator	PeerSim[50]	Collection of internally developed P2P models	Java	No	Free
	3LS	Gnutella	Java	No	---
	NeuroGrid[51]	Gnutella, NeuroGrid, Pastry, FreeNet	Java	No	Free

Table 5.2: Characteristics of discussed simulators

In addition to existent simulators, it is also possible to devise an ad-hoc simulation model to validate a cooperative incentive scheme. The mobile P2P file-sharing simulation model from [Oberender et al. 2005] contains a "network" component to model the network and devices' particularities and restrictions. It also includes a "source traffic" component to model the data transmitted and the behavior of peers in the network. The mobile P2P architecture used in this model is based on the eDonkey P2P file-sharing protocol and is enhanced by additional caching entities and a crawler. In the simulation, a mobile peer is described by an ON/OFF-process to reflect the fluctuating connection status of a mobile peer. ON and OFF periods are determined by exponential distributions, while the arrival of file requests is modeled by a Poisson process. An abstract model using a subset of the parameters of the detailed simulation is proposed to reduce the computing time. The abstract model is used to identify which cache replacement strategy fits the best for the mobile P2P system. In order to do so, the request arrival process is simulated in detail while the used transport mechanism and the upload queue mechanism are neglected.

5.1.3 Game theory

Results obtained through simulation studies give a proof-of-concept of the proposed cooperative mechanism. The results do not demonstrate if the incentives for cooperation are crucial or work by chance for instance. Game theory provides an alternative tool to decide if a cooperative mechanism is a cooperation strategy. Game theory models strategic decision situations where self-interested users follow a strategy aiming at

[45] The network Simulator ns-2, http://www.isi.edu/nsnam/ns/index.html
[46] The OMNeT++ Community Site, http://www.omnetpp.org/
[47] Global Mobile Information Systems Simulation Library, GloMoSim, http://pcl.cs.ucla.edu/projects/glomosim/
[48] Scalable Network Technologies (SNT), http://www.scalable-networks.com/products/
[49] OPNET, http://www.opnet.com/
[50] Biology-Inspired techniques for Self-Organization in dynamic Networks, BISON Project, http://www.cs.unibo.it/bison/
[51] NeuroGrid, http://www.neurogrid.net

maximizing their benefits and minimizing their resource consumption. Game theory offers different methods for study, e.g., non-cooperative game, cooperative game, and evolutionary game. *Non-cooperative game* focuses on users' strategies. It describes the strategy of a user that has to make a decision about whether to cooperate or not with a randomly chosen user. *Cooperative game* focuses on mutually advantageous results for the different parties. In this game, users are able to enforce contracts and make binding agreements. Finally, *evolutionary games* address the evolution of various strategy profiles over time and space.

The decision making process that a peer will undertake when participating in a non-cooperative game is often illustrated through the example of a classical game, *the prisoner's dilemma*. In this well-known game, two players are both faced with a decision to either cooperate (C) or defect (D). The two players' decisions are made simultaneously with no knowledge of the each other's decision. If the two players cooperate they receive a benefit R. If both defect they receive a punishment P. If one player defects and the other one cooperates, the defecting player receives a given benefit T and the cooperator a punishment S. The canonical form of the prisoner's dilemma pay-off is shown in the table below:

		Player 2	
		C	D
Player 1	C	(R, R)	(S, T)
	D	(T, S)	(P, P)

Table 5.3: Prisoner's dilemma pay-off matrix

In order to have a dilemma, the following expressions must hold:

$$T>R>P>S \text{ and } R>(S+T)/2$$

A player in this game has better to defect regardless of the decision of the other player because the strategy D strictly dominates the strategy C (T>R and P>S). The solution to this game is called a *Nash equilibrium*, that is, the set of strategies for which no player could improve his payoff (by changing his strategy) while the other players keep their strategies unchanged. The analysis of the interaction between decision-makers involved in the prisoner's dilemma game can be extended to *repeated (or iterated) games*. There are several strategies that a player can adopt to determine whether to cooperate or not at each of its moves in the repeated game. The basic strategy known as tit-for-tat corresponds to a player that cooperates in the first place and then copies his opponent's last move for all subsequent periods. Another strategy called Spiteful is to cooperate in the first period and for later periods cooperate if both players have always cooperated; if either player defects then defect for the remainder of the game. A cooperation enforcement mechanism can be translated into a strategy for a player and compared to these straightforward strategies.

Another important concept is the idea of *evolutionary stable strategy*. A set of strategies is at evolutionary stable strategy equilibrium if (a) no individual playing one strategy could improve its payoff by switching to one of the other strategies in the population and (b) no individual playing a different strategy (called a mutant) could establish itself in (invade) the population, i.e., make other individuals in the population choose his strategy. With these different concepts, one can give a good analysis of the cooperation enforcement mechanism in terms of both promoting cooperation and the evolution of cooperation. Cooperation and coalition formation can be explained using a two-period structure. Players first decide whether or not to join a coalition and in the second step the coalition and non-cooperative peers choose their behavior non-cooperatively. A coalition is defined as stable if no peer in the coalition has an incentive to leave. In this sense, a preference structure was suggested by the *ERC-theory* [Bolton and Ockenfels 2000]. In this theory,

the utility of a decision-maker is not solely based on the absolute payoff but also on the relative payoff compared to the overall payoff to all peers. The model explains observations from games where equity, reciprocity or competition plays a role.

The CORE mechanism [Michiardi 2004] was for instance validated following two methodologies. The first approach was to use a simulation tool, GloMoSim[52], while the second validation used a game-theoretic methodology. For the latter, two models, cooperative and non-cooperative game theory, were evaluated to demonstrate the need for cooperation incentives in the network. The CORE mechanism was then translated into a strategy model and its evolutionary stability was proven. Further, it was shown that in a more realistic scenario (communication errors, failures, etc.), CORE outperforms other basic cooperation strategies. Sprite [Zhong et al. 2003] also used a game-theoretic model to prove that the scheme proposed prevents cheating behaviors. The main results are intended for packet forwarding in unicast communication. The most important role of the game-theoretic validation of Sprite algorithms is in determining payments and charges of nodes in the system to motivate each node to cooperate honestly and to report its behavior to the CCS (Credit Clearance Service). Finally, remuneration fairness was studied from a game theoretic point of view in [Buttyán and Hubaux 2000], in which the authors discuss different equilibrium concepts as a model for the various types of fair exchange.

Conclusion

Different approaches can be taken for cooperation enforcement, yet cooperation evaluation is clearly the part most dependent on application-specific requirements and constraints, in particular concerning deployment. The choice of a particular technique for validating the effectiveness of cooperation, a critical step to ensuring that the application will reach its objectives, depends heavily on the application chosen. This may in particular hamper the use of one technique because of scalability issues or because of the properties that need to be proven.

Trust, as one would like to evaluate it in the applications mentioned above, can be static (based on identity for instance) or dynamic (self-organized). Static trust refers to a statement of trustworthiness that remains the same until it is revoked, whereas dynamic trust exhibits self-learning and self-amplifying characteristics. The latter arises from behaviors experienced in the system and continuously changes accordingly to them. An entity trusts a peer more when it has information about that peer that shows its trustfulness. [Carbone et al. 2003] for instance introduces a trust model that does not only concentrate on the content of evidence but also on the amount of such evidence.

Trust, although closely related with cooperation, may not be valuably accounted for by all cooperation evaluation metrics of the mechanisms listed above. In particular, whereas reputation seems well adapted to reason with the trustworthiness of a peer, remuneration may be much poorer semantically, especially if the payment may be used to enforce cooperation for different self-organized services. This does not mean that trust does not require cooperation as a prerequisite, but instead that trust establishment might not be as reliable as expected if the evaluation of its cooperative component relies on an unsuitable incentive mechanism.

[52] Global Mobile Information Systems Simulation Library, GloMoSim, http://pcl.cs.ucla.edu/projects/glomosim/

6– Connectivity in Unstructured Overlay Networks

Introduction

Peer-to-peer (P2P) systems are at present a widespread technology supporting many different applications and services that does not rely on any centralized control or on a unique administrative domain. Examples of such applications or services are presented in **[Baldoni et al. 2006-10, Baldoni et al. 2006-11]**. A P2P system is a highly dynamic distributed system in which nodes perpetually join and leave (churn). For these characteristics, a P2P system can reach a potentially infinitely wide scale with a transient population of nodes. Peer-to-peer systems induce a connected overlay network across the Internet. The overlay should maintain an "ideal" structure in order to scale and to implement efficient lookup operations.

Unstructured overlay networks have emerged as a viable solution to settle such issues in order to effectively support large scale dissemination and flooding based content searching. In the case of unstructured overlay, lookups means flooding the network and waiting for matches. To implement this operation efficiently and in a scalable way, one needs to keep an overlay network showing nice global properties like connectivity (for reliability), low-diameter and constant-degree (for scalability) without relying on a deterministic topology.

Unfortunately, the churn tends to move out the overlay network from its ideal topology and this turns out in reducing the lookups efficiency. An overlay maintenance protocol (OMP) tries to move back the overlay to its ideal topology despite churn. Therefore "overlay maintenance" is a fundamental problem in peer-to-peer (P2P) systems as it affects the performances of all of the services offered by a P2P system.

In this chapter we survey most known overlay maintenance protocols (i.e., [Ganesh et al. 2003, Allavena et al. 2006, Voulgaris et al. 2005, Pandurangan et al. 2001, Jelasity et al. 2003, Jelasity et al. 2004]) and provide a unifying vision of the performance tests that have been carried out in the literature in different studies [Ganesh et al. 2003, Allavena et al. 2006, Voulgaris et al. 2005, Pandurangan et al. 2001, Jelasity et al. 2003, Jelasity et al. 2004] **[Baldoni et al. 2006-07] [Baldoni et al. 2005-12]**. We first propose a classification of these protocols according to the fact that they act in a proactive or reactive manner. Secondly, we analyze their performance in semi-static environment (after an ideal overlay has been built, a percentage of peers leave the systems) and then in a dynamic environment with constant and variable churn rate. In case of constant churn the size of the network is constant along the time. In the case of variable churn, the size of the network may vary along the time arbitrarily.

The parameters of interest for the protocols evaluation are:

- Reachability: information about how many nodes a peer is able to reach with a multicast message in the network.
- Links Distribution: it shows how links are distributed among nodes of the network.
- Diameter: indicates the longest path in the network.
- Overlay Clustering: the parameter gives information about how many clusters are in the network and their size.

Reactive	**Proactive**
Scamp	Cyclon
PRU	ADH
	NewsCast
	PeerSampling

Table 6.1: Families of Overlay Maintenance Protocols

The survey points out that the current protocols could fail to maintain declared performance in a continuously dynamic environment while working well in environments where dynamic situations and static ones alternates. In particular:

1. All current solutions behave well under the semi-static level of dynamism (as tested by their authors);
2. proactive protocols show better behavior than reactive ones under network affected by constant churn rate **[Baldoni et al. 2006-07][Baldoni et al. 2005-12]**[Allavena et al. 2006] (second level of dynamism). However, both heavily degrades along the time;
3. current solutions have never been tested under dynamic environments with variable churn.

The remainder of the chapter is structured as follows. Section 6.1 presents a taxonomy of overlay maintenance protocols and section 6.2 details the description of the protocols with a table that briefly points out the major characteristic of the protocols. Section 6.3 shows the evaluation of the protocols under churn and conclusions conclude the chapter.

6.1. Taxonomy of Overlay Maintenance Protocols

An overlay can be viewed as a directed graph in which vertices are called *nodes* and edges are called *links*. Each node has a set of out-going links (*view*) and a set of in-going links (*in_view*). An out-going link (i, j) exists if and only if the nodes i's view contain node j's identification (id). Two nodes can communicate only if there exists a link between them. Communication between two nodes i, j is symmetric if there exists a link in both directions, e.g. an outgoing link from i to j and an outgoing link from j to i. The latter is also an in-going link for i.

Every overlay maintenance protocol is able to properly change node's views in order to adapt the overlay under the occurrence of joins/leaves and failures. For the sake of presentation, all the protocols described here are divided in two categories, looking at the way they refresh views at each node:

- **Reactive Protocols**: a node i changes its view only when:
  - i is joining the network;
  - a node j joins the network using i as *introducer*;
  - i has a link to a node j, and j is leaving the network.
- **Proactive Protocols**: a node i changes its view:
  - when one of the previous events happen;
  - when periodically, by starting a view exchange with another node.

In Table 6.1 the two main families of Overlay Maintenance Protocols are presented. For each family some representative protocols are included. In the next section these protocols are described in details.

6.2. Protocols Description

In this section overlay maintenance protocols are presented and Table 6.1 summarizes this description.

Each protocol differentiates from another by defining its own way to manage the following operations: join, leave, view exchange (the latter only for proactive ones). In Table 6.2 the description of each protocol points out how these operations are handled. Moreover, when some protocol uses some additional mechanism, its description is reported in the last column.

SCAMP

SCAMP [Ganesh et al. 2003] is a reactive gossip-based protocol whose main innovative feature is that the size of the view is adaptive w.r.t. a-priori unknown size of the whole system. More precisely, view size in SCAMP is logarithmic of the whole system size.

Join

When a node i joins the network it chooses one node as *introducer* among those already present in the overlay and sends a join request to it. The new joined node starts with a view consisting of just its introducer. When a node receives a new join request, it forwards the join request to all members of its own view. It also creates c additional copies of the new join request (c is a design parameter that determines the proportion of failures tolerated) and forwards them to randomly chosen nodes in its view. When a node receives a forwarded join request, provided i is not already present in its view, it integrates the new node i in its view with a probability equal to $1/(|view| + 1)$. If it decides not to keep the new node, it forwards the join request to a node randomly chosen from its $view$. If a node j decides to keep the join request, it places i in its $view$. It also sends a message to node i telling it to add j in its $in - view$.

Leave

When a node leaves, it orders the ids in its $view$ as $i(1), i(2), ..., i(l)$ and the ids in $in - view$ as $j(1), j(2), ..., j(l)$. The leaving node will inform nodes $j(1), ..., j(2), ..., j(l - c - 1)$ to replace its id with $i(1), i(2), ..., i(l - c - 1)$ respectively (wrapping around if $(l - c - 1)$¿l). It will inform nodes $j(l - c), ..., j(l)$ to remove it from their lists without replacing it by any id.

Other Mechanisms

Recovery from isolation A node i becomes isolated when all nodes containing its identifier in their views (all nodes contained in i's $in - view$) have either failed or left. In order to reconnect such nodes, a heartbeat mechanism is used. Each node periodically sends heartbeat messages to the nodes in its $view$. A node that has not received any heartbeat message in a long time re-joins through an arbitrary node in its $view$.

Indirection It is used to select an introducer, with uniform probability among nodes in the overlay. The indirection mechanism works as follows: when a node i receives a join request, it associates a counter with the join request initialized to a value proportional to the size of i's view. It then forwards the request, along with the counter, to a member j of its view. The node receiving the forwarded request decrements the counter and continues forwarding it with probabilities computed as above. When the counter is zero, the node receiving the request is the selected introducer.

Lease It is used to re-balance the graph during the time, considering that old nodes and new nodes may suffer from clustering, i.e. resulting in two different clusters (new nodes have more probability to connect to fresh nodes and less probability to connect to old nodes). The lease mechanism provides that a join request has a finite lifetime. When this lifetime expires for i, every node in i's in-view removes i and the node i joins again without modifying its view.

	Join	**Leave**	**View Exchange**	**Others**
SCAMP	The introducer forwards a join-request to its neighbors which keep or forward this request with a certain probability	The leaving node informs some of its neighbors to replace its id with another that it sends them	None	Recovery from isolation, Lease and Indirection
PRU	First phase: a joining node links to d nodes randomly selected from the server's view and becomes a $d-node$. Second phase: the $d-node$ i establishes a preferred connection with j, when it enters the cache by substituting j. Third phase: when a node in the server's view acquires c connections from new joining nodes, it leaves the server's view and becomes a $c-node$.	If a neighbor j of a node i leaves the network and the connection is not a preferred connection, i links another node in the view of the server's with a certain probability. If the connection is a preferred connection, i links a random node in the view of the server's view and this connection becomes preferred itself.	None	None
Cyclon	The introducer starts several random walks. Reached nodes overwrite one out-going link of theirs with an out-going link to the joining node. The joining nodes inserts these nodes in its view	Not managed	View exchange called Shuffling. A node i starts a view exchange session by picking up the oldest node from its view. Then, i sends to the selected node, *l* entries (which will be replaced with new elements) randomly selected from its view	None
ADH	The joining node copies the view of its introducer	Not managed	View Exchange called Mixing + Reinforcement The new i's view is selected from views of its neighbors and views of nodes that requested i's view	None
NewsCast	Joining node initializes its view with at least one node member of the overlay	Not managed	A peer selects randomly another neighbor and sends it all its view entries having back neighbor's entries. Nodes rewrite oldest entries.	None
PeerSampling RandRand	Not specified	Not specified	Random peer selection from the view and random selection of *l* elements. The node sends its selection to the selected peer.	None
PeerSampling TailHead	Not specified	Not specified	Selection of the last node from the view and selection of the first *l* elements in the view. The node sends its selection to the selected peer.	None
PeerSampling TailRand	Not specified	Not specified	Selection of the last node from the view and random selection of *l* elements. The node sends its selection to the selected peer.	None

Table 6.2: Protocol's operations

PRU

PRU [Pandurangan et al. 2001] is a reactive protocol whose key element is a host server which maintains a $view$ of K nodes of the graph. This server, obviously, is reachable from every node of the overlay and it is supposed not to fail.

Join

When a node joins, it contacts the host server and takes random d (a constant) number of nodes from the server's view. At this point the node is called a *d-node*. It remains a d-node until it subsequently leaves the network or enters the server's view. A node i enters the server's view when a node j in the server's view leaves this view, and i is a neighbor of j selected by a specific rule. The node i stays in the server's view until it gets c (a constant) connections from new joining nodes; at this point it leaves the server's view and is defined as a *c-node*: a *c-node* has always a preferred connection to the node that it replaced to enters the server's view.

Leave

If a node i has a neighbor j that leaves the network there are two ways for its replacement:

- if i has a preferred connection with j then i reconnects to a random node in the server's view and makes the new connection its preferred connection;
- otherwise i connects to a random node in the server's view with a certain probability depending on some parameters.

The preferred connection is useful to avoid little isolated clusters that appear in the network [Pandurangan et al. 2001].

Cyclon

Cyclon [Voulgaris et al. 2005] follows a proactive approach where nodes perform a continuous periodic view exchange with their neighbors in the overlay. The view exchange (named in this case *shuffling*) aims at randomly mixing links between neighbor nodes. Leaves are treated as failures (no leave algorithm is provided) and a simple failure detection mechanism is integrated in the shuffling mechanism to clean views from failed nodes.

Each node i maintains a $view$ with fixed size. Each node j in the view is associated to a local age, indicating the number of shuffles during which the node j was present in the i's view.

Join

When a node i joins the network it chooses one node as introducer among those already present in the overlay. The protocol then starts a set of independent random walks from the introducer. When each random walk terminates, the last visited node, say j, adds i to its view by replacing one link to a node, say k, which is added to i's view.

View Exchange

A shuffling cycle is composed of three phases. In the first phase a node i, after increasing the age of all the nodes in its view, chooses its shuffle target, j, as the node with higher age among those in its view. Then, i sends to j a shuffling message containing $l-1$ nodes randomly chosen in i's view, plus i itself. In the second phase, j, once received the shuffling message from i, replaces $l-1$ nodes in its view (chosen at random) with the l nodes received from i and sends them back to i. In the final phase i replaces the nodes previously sent to j with those received from it. The link previously connecting i to j is also reversed after the shuffle.

ADH

ADH [Allavena et al. 2006] is a proactive protocol which employs a slightly different strategy to maintain views in comparison with Cyclon. Each node periodically substitutes its whole view with a new one, which is built basing on information collected since the last view exchange. Even in this case joins are managed in a reactive manner, through a join procedure, while voluntary departures of nodes are treated as failures. Failure detection techniques are not used because crashed nodes are automatically discarded by view exchanges after some time.

ADH, as Cyclon, employs a single view for each node. The size of the view k is fixed and can be set arbitrarily. Two more parameters are used: the fanout f and the weight of reinforcement w, both detailed in the following.

Join

Nodes joining the overlay network fill their initially empty views with the view of the introducer.

View Echange

During a view exchange each node collects a list L_1 comprising the local views of f nodes chosen at random from its view and a list L_2 comprising those nodes that concurrently requested its view. At the end of this exchange these two lists are used to create the local view. The new view is built by choosing k nodes from both L_1 and L_2. The weight of reinforcement w is used to decide from which list a node must be picked: if $w = 0$ then all nodes are selected in L_1, if $w = 1$ nodes are selected with equal probability in L_1 and L_2, and, finally, if $w = \infty$ then all nodes are selected in L_2. This mechanism is used to clean the overlay from failed nodes (that surely will not appear in L_2), while mixing views. For these reasons the authors of [Allavena et al. 2006] with respect to the value of w, suggest "larger is better and will be either 1 or 1 on a typical implementation".

NewsCast

NewsCast [Jelasity et al. 2003] is a proactive based protocol which assumes loosely synchronization among nodes. In particular, each node-id in a view is associated with a timestamp coming from the skew of two nodes, as detailed below.

Join

A joining node initializes its view with an introducer.

View Exchange

The view exchange is performed as follows: a node i selects a node j to send it the request of new links and sends its view (at most l entries) along with the current local time T_i to j. The node j sends its view to i along with the current local time T_j and adds to the timestamp of each new entry the value $T_j - T_i$. When i receives the new view, it adds to the timestamps of the new entries a value equal to the current local time minus T_j. Both nodes merge their views, obtaining at most $2l + 1$ entries. Then, they remove oldest entries to keep the l freshest entries.

PeerSampling

In [Jelasity et al. 2004] different ways to exchange views are compared. We present here three different view exchanges, calling the related protocols, respectively: **RandRand**, **TailHead** and **TailRand**.

- **RandRand**: a node i selects a random node from its view to begin view exchange. Then it sends a random selection of l elements to the selected node.
- **TailHead**: a node i selects the last node from its view to begin view exchange. Then it the first l elements in its view sends to the selected node.
- **TailRand**: a node i selects the last node from its view to begin view exchange. Then it sends a random selection of l elements to the selected node.

6.3. Protocols Evaluation

In this Section we will show known results about the previous presented protocols under three different scenarios: semi-static, constant churn, variable churn. All these scenarios evaluate different levels of churn where churn indicates the situation of some nodes leaving the system and some others joining the system.

The effects of churn may heavily affect the overlay topology. In order to see these effects the following parameters are considered:

Evaluated Parameters

- *Reachability*: information about how many nodes a peer is able to reach with a multicast message in the network.
- *Links Distribution*: it shows how links are distributed among the nodes of the network.
- *Diameter*: indicates the longest path in the network.
- *Overlay Clustering*: the parameter gives information about how many clusters are in the network and their size.

6.3.1. Semi-Static Environment

In Table 6.3 we can see known results in a semi-static environment. A semi-static environment considers an overlay network obtained by a serialized sequence of join operations and than decreased by removing some percentage of nodes. Typically, the evaluation of parameter is done before and after the removal.

SCAMP

SCAMP [Ganesh et al. 2003] In the semi-static environment SCAMP shows a good behaviour. As for reachability, in a failulre-free scenario, i.e. before any removal, SCAMP offers a reachability of 100%. Then, after a massive removal equals to 70% it still offers a reachability equals to 95%. Links distribution shows that the objective of the protocol is reached: the view size of a node is always equal to $log(n)$ where n is the total number of nodes inside the network. In condition of a fail free network, moreover, no clustering happens by the reachability data.

Semi-Static Environment				
	Reachability	**Links Distribution**	**Diameter**	**Overlay Clustering**
SCAMP	100% in a failure-free network. 95% after the removal of 70% of nodes.	The view size is $O(log(n))$ where n is total number of nodes in the overlay.		No clustering in a failure-free network.
Cyclon	100% in a failure-free network with a view size sublinear w.r.t the total number of nodes.	Size of the in-view nearly equal to the size of the view.		No clustering in a failure-free network.
NewsCast	100% in a failure free network. 40% after the removal of 95% of nodes. These results hold with a view size sublinear w.r.t the total number of nodes.		$O(log(n))$	First cluster (less than 1%) appears if 68% of nodes are removed. If 95% of nodes are removed then the largest cluster has a size of 2% of the initial number of nodes. These results hold with a view size sublinear w.r.t the total number of nodes equals to 100.000
PeerSampling	100% in a failure-free network. 20% after the removal of 95% of nodes.	Average node view size = $O(log(n))$.	Average path length: $O(log(n))$.	Failure-free network, average largest cluster for **RandRand** and **TailRand** 96-99% while 70% for **TailHead**. After removal of 95% of nodes: average largest cluster: 1% of all initial nodes.

Table 6.3: Known Results in semi-static environment

CYCLON

Cyclon [Voulgaris et al. 2005] has been tested in [Voulgaris et al. 2005] without considering removals but just after the serialized sequence of joins. Then, in the fail free context considered in the paper the reachability is 100% when the view size is set to a value equal to 20 as compared to 10,000 nodes, the in-degree distribution is on average with the same value of the view size and there exists no cluster.

NEWSCAST

NewsCast [Jelasity et al. 2003] has been tested in [Jelasity et al. 2003] to evaluate reachability, diameter and clustering. As for reachability, it offers a performance of 100% if the network is failure-free and equals to 20% if 95% of nodes are removed. The second parameter shows a diameter 4.5 in a network of 100,000 nodes and an average view size 20. The clustering study shows that if the view size is 20 then the first cluster appears after removal of 68% of nodes and that after the removal of 95% of nodes the biggest cluster is composed by 2,000 nodes.

PEERSAMPLING

About the static environment we also have some results about protocols tested in PeerSampling [Jelasity et al. 2004] on a network of 10,000 nodes. In this condition selected protocols are tested only on reachability showing an average result of 0 nodes outside the largest cluster in a fail free network and of 20% after removal of 95% of nodes. The average node degree is 55 and the average path length 3. The Overlay Clustering results have to be schematized because of big differences among the three view exchange protocols used:

- RandRand:
  - Average number of clusters 2.27
  - Average largest cluster 9572.18
- TailHead:
  - Average number of clusters 38.19
  - Average largest cluster 7150.52
- TailRand:
  - Average number of clusters 2.00
  - Average largest cluster 9941

ADH and PRU

Have not been tested in semi-static conditions.

6.3.2. Dynamic Scenarios

In Tables 6.4 and 6.5 known results in dynamic environments are presented. The first Table 6.4 points at a network affected by constant churn rate. This means that the dynamicity of nodes is modelled through a constant portion of nodes that join and leave every time units. The second table shows known results in a network affected by a different model for churn rate which consider a variable portion of nodes joining/leaving in different time units (see below for further details).

All protocols but **ADH-synch** and **PRU** (see below) are evaluated in their asynchronous version: at each time unit, C joins and C leaves are injected and their execution requires some time. Then, each node can update its view more than once in a time unit or not at all.

In the Table 6.4 results are given by specifying the number of refreshments: the number of times the overlay has substituted all the nodes.

Constant Churn Rate				
	Reachability	**Links Distribution**	**Diameter**	**Overlay Clustering**
SCAMP	15% after 3 total refreshments and C=2. 5% after 3 total refreshments and C=4. 1% after 3 total refreshments and C=8.	The view size is $log(n)$ where n is the network size.		after 3 total refreshments: C=2 MC=35% SC=35% IN=30%. C=8 MC=0% SC=55% IN=45%.
Cyclon	95% after 3 total refreshments and C=2. 75% after 3 total refreshments and C=4. 27% after 3 total refreshments and C=8.	as for the semi-static case		After 3 network total refreshments and some shuffling cycles: C=2 MC=99% SC=0% IN=1%. C=8 MC=60% SC=15% IN=25%.
ADH	93% after 3 total refreshments and C=2. 68% after 3 total refreshments and C=4. 28% after 3 total refreshments and C=8.	In a network of 100.000 nodes and view size 17, in a loosely synchronized system the maximum in-degree is always below 4.5 times that of a random graph.		After 3 total refreshments: C=8 MC=50% LC=31% IN=19%.
ADH-synch				With C=n/32 (1.6) and $view = log_2(n)$, number of iterations until partitioning are 1000 if n=30 3000 if n=100 7000 if n=1000.

Table 6.4: Results in constant churn rate environment

Parameters used in the tables are:

- **C** is the churn rate. It means that at every time unit C join operations and C leave operations happen.
- **n** is the size of the network.
- **MC** is the size of the main cluster: in a partitioned network it shows the percentage of n belonging to the largest cluster.
- **SC** it shows the percentage of n spread in small clusters.
- **IN** it shows the percentage of isolated nodes.

Variable Churn Rate (arrival: Poisson distribution; duration of time a node connected to the network: exponential distribution)				
	Reachability	**Links Distribution**	**Diameter**	**Overlay Clustering**
PRU	At all times each node is connected to some cache node. If u and v are 2 cache nodes, there is a probability = $1 - (log2(n)/n)$ that there is a path in the network at time t connecting v and u. There is a Single Point of Failure.		$O(log(n))$.	

Table 6.5: Results in variable churn rate environment

The tables shows that:

SCAMP

Tested on a dynamic environment **[Baldoni et al. 2006-07] SCAMP** highlights how the excellent performances on static situations do not appear to be the same on dynamic networks. In example the reachability on a C=2 situation is 15% after three network refresh in a network composed by 1000 nodes. Furthermore if we try to raise the churn rate on value of C=4 and C=8 the reachability is reduced to 5% and 1% respectively.
About the clustering we tested SCAMP on a totally asynchronous environment and after three network refresh and the execution of many runs, average values was: MC=35% LC=35% IN =30% if C=2 and MC=0% LC=55% IN =45% if C=8.

CYCLON

The situation of **Cyclon** is quite different: testing it in the same situations of SCAMP **[Baldoni et al. 2006-07] [Baldoni et al. 2005-12]** we have obtained better results; in fact Cyclon's performances are 95% if C=2, 75% if C=4 and 27% if C=8. Similar improvements are showed on the other parameters too. The links distribution and the diameter, after a little stability period, is almost the same of the static environment. The clustering is also interesting: using a churn of C=2 Cyclon manage the network obtaining excellent results which we summarize in MC=99% LC=0% IN =1%. Finally if C=8 then MC=60% LC=15% IN =25%.

ADH-synch

This version of the protocol has been tested by the authors in [Allavena et al. 2006]. **ADH-synch** is loosely synchronous, which means that nodes are updated sequentially in random order, each node being updated exaclty once per time unit.

Reachability: with C=n/32 (C=1.6) and view=$\log_2 n$ authors calculated the number of iterations until partitioning obtaining these results: 1000 if n=30, 3000 if n=100 and 7000 if n=1000. Links Distribution: in a network of 100,000 nodes with a view size of 17, the maximum size of in-views is always below 4.5 times that of a random graph.

ADH

Reachability: in the same conditions of Cyclon and SCAMP we obtained respectively 93%, 68% and 28% Overlay Clustering: with C=2 there are no particular differences with Cyclon but if C=8 then MC=50% LC=31% IN=19%.

PRU

PRU has been analytically evaluated (no simulation has been done) in a dynamic environment with a specific model for churn. The authors assume a poisson arrival distribution and uptime following an exponential distribution. The parameters evaluated are reachability and diameter.

Reachability results point out that if i and j are two nodes in the server's view, there is a probability = $1 - (log_2 n/n)$ that there exists a path in the network at time t connecting i and j. This protocol is designed to create networks with a little diameter: results shows a diameter of $O(log(n))$.

Conclusions

This chapter pointed out the importance of continuous churn as first class enemy that must be fought in order to maintain a connected overlay network with a shape as close as possible to the ideal one. The chapter presented a comparison among overlay maintenance protocols for unstructured overlay. The comparison has focussed on their ability to cope with churn and to bring back an overlay topology to its ideal shape. While these protocols are effective in avoiding large network breakages, we showed that they suffer from clustering under continuous churn **[Baldoni et al. 2006]**.

Despite this, not all protocols suffer from churn in the same way. The comparison show that proactive protocols based on view-shuffling resist better to the reactive counterpart. In other words they are able to remain connected against longer period of continuous churn.

7 – High assurance voting systems

Introduction

The trustworthiness and dependability of voting systems has received a great deal of media attention in recent years, notably in the wake of the 2000 and 2004 US presidential elections. It is clear that ensuring that the public has a high degree of confidence in the outcome of an election is of paramount importance. It is essential that voting systems are not only trustworthy but also seen to be trustworthy.

For over a century, the US has been using quite sophisticated technologies: the earliest lever machines go back to the end of the 19th century, followed by punchcards, optical scanning and touch screens. The drive to adopt such technologies may have arisen from the level of fraud witnessed with the pen and paper system. In the recent presidential election in the US, approximately 30% of the electorate who voted did so using touch screen machines.The excellent book by Gumbel, "Steal this Vote" [Gumbel 2005], documents an extraordinary litany of instances of corruption and techniques to corrupt every voting system that has ever been deployed in the US. Recent reports, like the Johns Hopkins report [Kohno 2004] and more recently the Princeton report [Felton 2006] on the Diebold touch screen devices have demonstrated how vulnerable a poorly designed electronic voting system can be to virtually undetectable corruption.

There is generally pressure from politicians etc for the adoption of electronic means for various stages in the processing of votes. There appears to be a number of motivations for this: the convenience of remote, e.g., internet, phone voting etc. is thought to encourage higher participation. The use of touch screens to assist vote capture helps with more complex choices presented to voters, as common in many US elections. Electronic counting potentially could help with the speed and accuracy and efficiency. However, it is clear that hasty adoption of poorly designed systems could seriously endanger the integrity of elections and public confidence.

The challenge is to design systems that provide high assurance of the accuracy of the outcome, whilst at the same time guaranteeing the secrecy of ballots. These two requirements are seemingly contradictory. In the absence of the secrecy requirement, accuracy would be trivial to achieve: a simple show of hands would suffice. However, if every voter's choice is to be kept secret, such an approach does not work. Equally, if accuracy is not required, then secrecy is trivial: a constant function achieves this. Similarly, if complete trust is placed in the process or officials who collect and count the votes then again there is no real problem. The real challenge is to provide accuracy and secrecy with minimal, ideally no, trust in the devices, processes and officials that conduct the election. And, as if this were not challenging enough, the systems must be transparent and simple enough to gain a reasonable degree of public understanding and trust. The problem is thus highly socio-technical in nature.

It is important to draw a clear distinction between supervised and remote voting systems. In the former, voting takes place in a controlled environment such as a polling station and the casting of the vote occurs in the enforced isolation of a polling booth. In remote systems, voters are able to cast their vote over a suitable channel, e.g., postal, internet, telephone, interactive TV etc. In such systems, the isolation of the voter whilst

casting their vote cannot be enforced and the threats of vote buying and coercion are consequently much more serious.

A fundamental problem that distinguishes voting systems from conventional dependable systems is that there is no extrinsic way to characterise the correctness of the outcome of a secret ballot. By definition, no party has a god-like view of the intentions of voters that can determine if the result is correct or not. Thus, it is possible for an election system to fail in a way that is not manifest. The cryptographic systems that we outline below can be thought of as striving to make errors manifest, whilst preserving ballot secrecy.

Voting systems can be viewed as a special case of secure distributed computation, but with all the additional socio-technical aspects. Besides the devices and processes that capture, transmit and count votes, there are the surrounding systems that maintain the electoral register, authenticate voters, officials and scrutinisers who supervise and observe the process, etc.

In the past few years, significant progress has been made towards systems that can provide high assurance of accuracy and ballot secrecy whilst being sufficiently simple to gain the understanding and confidence of the various stakeholders: the voters, election officials, politicians, etc. Significant contributions have been made by ReSIST partners, in particular Newcastle University.

In this chapter we will outline the state-of-the-art in high assurance, cryptographically based voting systems. The execution of the system should be as transparent as possible within the constraints of ballot secrecy. These can be thought of as placing voting systems on a scientific, falsifiable basis, in contrast to having to place faith in the high priests of the Diebold or Sequoia Corporations.

7.1. The Requirements

At the most abstract level, we are seeking to ensure “freeness and fairness” of the democratic process. This rather vague statement has to be mapped down to more precise requirements on the various components of the socio-technical system. Key properties, described informally, include:

7.1.1. Accuracy/integrity

All legitimately cast votes should be accurately included in the count.

Legitimate voters should be able to cast at most one vote.

Many of the crypto schemes described below provide unconditional integrity, that is, they provide guarantees of integrity that do not depend on computational assumptions. Integrity is guaranteed even against an adversary with unbounded computational power.

7.1.2. Ballot Privacy/secrecy

It should not be possible for a third party to establish which way a particular voter cast their vote.

7.1.3. Voter anonymity

It is sometimes required that the fact of having voted be kept secret. This requirement is sometimes termed “voter-anonymity”. Clearly, where voting is obligatory, such a requirement is not applicable.

7.1.4. Accessibility

All legitimate voters should be able to cast their votes without let or hindrance during the voting period.

7.1.5. Voter-verifiability

This is a rather novel requirement, not feasible in most conventional systems. Here voters are able to verify that their vote in accurately included in the count and, in the event of their vote not being accurately included, to prove this to a judge. At the same time, the voter is not able to prove to a third party which way they voted. At first glance this seems impossible, but some new cryptographic schemes that we outline below do realise this requirement.

7.1.6. Receipt-freeness

This is a refinement of the notion of voter privacy. Here the requirement is that it should not be possible for the voter to construct any form of receipt, i.e. any proof to a third party of how they cast their vote. Thus, if the protocol requires the voter to use certain keys or passwords, we assume that the voter is willing to reveal these to the coercer. Thus, for the protocol to be receipt-free against this model, it is necessary that the voter be able to lie to the coercer without fear of being detected. Note that most conventional pen and paper systems are receipt-free in a rather trivial sense: the voter does not get any kind of record of how they voted. Note further that the term "receipt-free" can be a little misleading: many of the schemes we describe below do provide the voter with a form of receipt, but one that carried their vote in concealed form. Conversely, in some schemes, for example FOO [Fujioka 1992], the voter does not obtain a receipt, and yet is able to later construct a proof of her vote.

7.1.7. Coercion-resistance

This is an even stronger requirement that receipt-freeness, or more precisely, it is essentially the same property but in the context of an even stronger adversary model (coercer). Here we assume that the coercer is able to observe and influence certain steps of the vote casting protocol. If the protocol involves the voter making certain choices we assume that these choices may be dictated by the coercer. Thus we are assuming a more active attacker than for receipt-freeness, where the attacker was essentially passive, i.e., could only observe but not influence steps of the protocol. For a scheme satisfying this property to be possible, it is necessary to assume at least one step in the interaction that cannot be observed by the coercer, otherwise the voter and coercer would be indistinguishable to the system.

7.1.8. Usability, public trust and acceptance

Besides all the above technical requirements, voting systems must also be easy for voters to use and capable of gaining a sufficient degree of public understanding and trust.

7.2. Cryptographic schemes

In general, assurance of "correct" behaviour of a system can be derived in various ways. At one end of the spectrum, it can be based on prior evaluation, verification and testing of the system. Such analysis seeks to demonstrate all possible system executions will satisfy the requirements. At the other end of the spectrum, assurance may be derived from close monitoring of the system as it executes. In this approach, we seek to detect any departure from correct behaviour and take suitable corrective action in the event of any departure.

Both approaches have a useful role to play. Emphasis must be placed on the former approach for systems for which failure may be hard to recover, e.g., an avionics system. The drawback with the approach is, firstly, that complete and correct analysis is extremely difficult to achieve and, secondly, even if it could be achieved, system degradation or upgrades, changes in the environment, etc. may all serve to invalidate the basis of the analysis. Such assurance is thus very fragile.

Run-time monitoring by contrast, is less susceptible to the vagaries of the implementation, maintenance and environment. Run-time monitoring is far more robust: as long as we have a precise definition of "correct" behaviour and can implement a suitable execution monitor, we will be able to detect and, depending on the property, recover from any erroneous behaviour. We are no longer prey to the problems of incomplete analysis or testing, software updates etc.

In the case of voting systems, we argue that assurance of integrity should be based primarily on run-time monitoring. In a suitably designed voting system, recovery from an erroneous state should be perfectly feasible, if somewhat irritating. Of course, we will still want to verify and test the system in order to ensure that it is reasonably dependable and so satisfies availability requirements, but our assurance of integrity will not be conditioned on this analysis. To borrow Benaloh's phrase: "we should verify the election, not the system".

We should note that secrecy or privacy properties are of quite a different nature to integrity properties. The later are typically trace, also known as safety, properties, i.e., can be formalised as predicates over the system traces (behaviours). Secrecy properties are usually defined in terms of the entire set of behaviours rather than individual traces. As such, it is typically not possible to detect failures with respect to a secrecy property by monitoring an individual execution. It also tends to be much harder to recover from secrecy failures. Consequently, for secrecy, we do need to place more reliance on prior verification than for integrity.

Many cryptographic schemes that seek to provide high levels of assurance of accuracy and secrecy have been proposed. Typically these strive to minimise the need to place trust in the implementations and they seek assurance through maximal transparency. In accordance with the principle of "no security through obscurity", the details of these schemes are laid bare to universal scrutiny, so that any flaws in the design may be detected before any deployment. Furthermore, the integrity of the election rests now on the validity of the mathematics arguments rather than on mutable implementations. This is a key and very subtle point: if the implementation malfunctions or is corrupted, this will be detected at run-time by the auditing procedures. All the computations performed during the auditing phase are against publicly known functions and are independently verifiable.

Notable examples are Chaum [Chaum 2004], VoteHere [Neff 2006] and Prêt à Voter [Ryan 2004, Ryan 2005, Ryan 2006]]. These enable the voters to gain confidence and contribute to the dependability by allowing them to perform certain checks that serve to detect any malfunction or corruption of the devices.

Cryptographic schemes open up novel and surprising possibilities, in particular the notion of voter-verifiability. This enables voters to confirm that their vote is accurately recorded and counted whilst at the same time avoiding any possibility of them proving which way they voted to a third party. The key idea in achieving this is to provide voters with a receipt at the time casting that carries their vote in encrypted form. Voters are later able to confirm, via a secure Web Bulletin Board (WBB), that their receipt has been correctly entered into a robust, anonymising tabulation process. Mechanisms are provided to ensure that:

1. The voter's intent is accurately encoded in the receipt.
2. The tabulation process accurately decrypts all receipts.

Putting the pieces of the argument together, we can conclude that each voter can assure themselves that their vote will be counted as intended.

We next sketch some of the crypto primitives utilised in cryptographic voting schemes. For more detailed descriptions consult any good book on cryptography, e.g., Stinson [Stinson 2005]. We assume familiarity with the basis notions of symmetric and asymmetric (public) cryptography.

7.3. Cryptographic primitives

7.3.1. Threshold cryptography

It is often important not to depend, either for secrecy or availability, on a single entity to perform cryptographic operations, such as encryption, decryption, signing, etc. This prompted the development of techniques and algorithms to distribute the knowledge of crypto variables amongst a set of entities in such a way that only certain subsets of can perform the operation.

7.3.2. Cryptographic commitments

These can be thought of as the cryptographic analogue of Anne writing down a data item on a piece of paper, locking this in a box and passing the box to Bob whilst retaining the key. She has now committed to the data value but Bob does not yet have access to it. To reveal it she hands the key to Bob. Such schemes should provide two properties: commitment, i.e., Anne should not be able to alter the committed value, and secrecy: Bob should not be able to gain any knowledge of the value until Anne releases the key.

7.3.3. Robust anonymising mixes

In decryption mixes, a deterministic encryption algorithm such as RSA is used. The plaintexts are encrypted in a number of layers under the public keys of the mix servers. The batch of receipts is put through a series of robust anonymising mixes that progressively decrypt and shuffle them. At the end of these mixes, the raw, decrypted votes pop out but with any link to the original receipts obliterated by the multiple shuffles. The fact that all the terms passing through the mix go through a decryption at each stage ensures that they cannot be simply traced through the mix by simple pattern matching.

For re-encryption mixes, a randomising algorithm such as ElGamal or Paillier is used. Instead of striping off layers of encryption at each step of the mix, the mix servers re-randomise the encryption. The mix servers do not hold any secret keys: re-randomisation can be done by any entity that knows the public key. A sequence of such mixes can then be followed by a decryption stage by servers that hold the appropriate secret keys.

Various checks, such as randomised partial checking [Jakobsson 2002], can be deployed to ensure that any attempt to corrupt votes by either incorrectly performing the encryption of the receipts or incorrectly decrypting the receipts during the mixes will, with high probability, be detected.

Tabulation using mixes can thus be thought of as analogous to conventional counting of paper ballots.

7.3.4. Homomorphic tabulation

In the homomorphic tabulation, suitable cryptographic primitives are employed that exhibit algebraic homomorphic properties which enable the count to be performed without needing to decrypt individual receipts. For example, the Paillier algorithm has the property that the product of the encryption of a set of values equals the encryption of the sum of the values:

$$\Pi_i \ \mathrm{Enc}(x_i) = \mathrm{Enc}(\Sigma_i \ x_i)$$

This can be exploited to extract the overall count without having to decrypt individual votes. For a simple yes/no referendum it is clear how encode votes in order to extract the result: yes votes are represented are encryption of +1, whilst no votes are 0. Thus, if the overall sum is >n/2, where n is the number of votes cast, the ayes carry it, if the sum is <n/2, the nays. For elections with choices of more than 2 candidates or options, more subtle encodings are required. Homomorphic tabulation can thus be thought of as roughly analogous to the operation of lever machines.

7.3.5. Cut and choose

This is a common device to avoid the need to trust a device performing a cryptographic operation, typically encryption. The device is required to commit to several independent encryptions of the given plaintext. It is then challenged to reveal the keys or randomising factors for all but one, randomly chosen by the requestor. If all of these challenged encryptions prove to be valid then it is a good bet that the unrevealed encryption is valid too.

Many voting schemes which generate encrypted receipts employ a cut-and-choose element as a way to detect malfunction or misbehaviour in the encryption devices. An example is the pre-election audit on a random selection of ballot forms in Prêt à Voter [Ryan, 2005]: the authority commits to the cryptographic primitives used in the construction of the forms before it knows which will be selected for auditing.

7.3.6. Digital signatures

These are the digital analogue of conventional signatures. Typically, Anne would digitally sign a text M by appending a crypto hash of the text encrypted under her private key. Anyone knowing Anne's public key can verify this by applying the public key to the encrypted hash and applying the hash to the plaintext. If these two computations agree one can be confident that the text was indeed signed by Anne and has not been altered. This assumes that Anne's private key has not been compromised.

7.3.7. Blinded digital signatures

Blind digital signatures [Chaum 1992] can be used to authenticate a vote while ensuring secrecy. Suppose that Anne wants Bob to sign some text but does not want to reveal the text to him. This can be achieved using the notion of blinding: she constructs a term that is scrambled (blinded) by a random value she generates. She passes this to Bob for him to sign and when she gets the signed term back she reverses the blinding to yield the original text signed by Bob. For this to work crypto primitives must be used such that the blinding operation commutes with the signing.

7.3.8. Zero-knowledge proofs

An interactive zero-knowledge proof (ZKP) [Goldreich 1991] is a protocol in which one party, the prover P, demonstrates knowledge of a fact to another, the verifier V, without V learning anything other than the truth of the statement. Such protocols typically involve a sequence of random challenges issued by the verifier to the prover. The proof can be non-interactive if the two parties concerned share a pre-determined random string [Blum 1991]. A well-known device, due to Fiat and Shamir, for converting an interactive ZKP into a non-interactive one if for the challenges to be determined by some pre-agreed crypto-hash of the proof script [Fiat 1986]. ZKPs have been used in mix-nets for proving ciphertext equivalence, and hence improving their robustness [Jakobsson 1999].

7.3.9. The Benaloh-Tunistra device

An interactive proof is only convincing if it is clear that the prover could not predict the challenges. This observation can be neatly turned around to allow the construction of "false proofs". If the challenges are known in advance, it is easy to construct a proof script of a false statement. Such a script will be indistinguishable from a genuine proof to someone not involved in the proof interaction. Such constructions can be useful in situations in which we want to provide genuine proofs to individual voters whilst maintaining coercion-resistance.

We now outline the key developments in the field. Full details can be found in the references.

7.4. Voter-verifiable, cryptographic schemes

7.4.1. Benaloh Scheme

In his Yale thesis, Benaloh [Benaloh 1987], introduced the concept of voter-verifiability and proposed a possible implementation. This involved a rather complex protocol between the voter and the device in the booth in order to create the encrypted receipt and provide an ephemeral proof of correctness, existing only within the booth during the process of creating the receipt.

7.4.2. FOO

Fujioka et al [Fujioka 1992] devised a remote voting scheme which makes use of blind signatures. Although coercion-resistance is near impossible to ensure in the remote context, the protocol has other flaws which leave the voter open to coercion [Peacock 2006].

7.4.3. Shoenmakers et al

This is a scheme allowing remote, e.g., internet, voting. It has the drawback of requiring the voters to place trust in a personalised computing device to perform certain cryptographic operations.

7.4.4. Chaum's Visual Crypto System

The next major step was a scheme proposed by Chaum [Chaum 2004], that made ingenious use of the notion of visual cryptography [Naor 1995]. In the booth, the voter would provide their choice to the device. This

would create two layers of patterns of pixels and print these onto two overlaid layers of transparency. Correctly overlaid, these transparent layers create a 2D image of the voter's choice. Viewed separately, they reveal only random array of pixels. Having confirmed that correct ballot image appeared, the voter, still in the booth, would separate the layers and make an arbitrary choice as to which layer to retain as her receipt.

7.4.5. Neff's VoteHere

Around the same time, and apparently independently of Chaum, Neff proposed an ingenious scheme involving paper receipts but, rather like the original Benaloh scheme, involving a fairly complex interaction between the voter and the booth device [Karlof 2005]. Note however, that this may no longer be true of the current implementation of VoteHere [Neff 2006]. Neff also proposed a rather efficient scheme for robust mixes of ElGamal encrypted terms.

7.4.6. Ryan's Prêt à Voter (and its variants)

Taking the Chaum scheme as the staring point, Ryan, of Newcastle [Ryan, 2004], proposed a far simpler way of encoding voters' choices in a receipt. The key innovation is, for each ballot, to create a randomised frame of reference, for example, a randomised order of candidates. Voters indicate their choice in the usual way, for example putting an X against their choice of candidate. The left hand column is now detached and discarded, leaving the receipt.

Obelix	
Asterix	
Panoramix	
Idefix	
	2h78Uj629kM

Figure 1: typical Prêt à Voter ballot form

Obelix	
Asterix	X
Panoramix	
Idefix	
	2h78Uj629kM

Figure 2: ballot form showing a vote for Asterix.

X
2h78Uj629kM

Figure 3: with the left hand column removed to leave the receipt.

Information allowing the tellers to reconstruct the order and so extract the vote is buried in the onion printed at the bottom of the right hand column.

The voter now leaves the booth with their receipt and registers with a polling station official. Their identity is checked and their right to vote confirmed against the electoral role. The vote is now cast in the presence of one or more officials: a digital copy of the information on the receipt is made: an index value indicating the position of the X along with the cryptographic value (the onion) at the bottom. Additionally, a paper copy could be created and verified by the voter and officials to create an encrypted paper audit trail. This has an advantage over the Mercuri VVPAT [Mercuri 2005]. To protect voter privacy, it is usually required that ballots be separated and shuffled.

An encrypted audit trail could, by contrast, be printed on a paper roll like a till receipt. This is easier to implement and a record preserved on a single strip is much harder to manipulate than a batch of ballots. The need for a publicly verifiable record of the votes cast in the case of touch screen devices has been recognised [Dill 2006]. This is essential for auditing purposes and as a back-up should the record of receipts posted to the WBB be contested. It can also serve as the basis for further checks of correspondence between the audit record and what is posted to the WBB to be performed by independent auditors to supplement voter checks.

Clearly, this approach is conceptually and technologically far simpler than the Chaum original. It also has a rather subtle but significant additional advantage: due to the way that votes are encoded, the voter need never communicate their selection to any device. Other schemes involve the vote choice being communicated to a device which produces one or more encryptions. This results in the possibility of hidden channels leaking information about the voter's choice.

Various enhancements to the basic scheme sketched above have been proposed. In [Ryan 2005], Ryan et al propose concealing the onion with a scratch strip until the actual time of casting a vote as a way to counter chain-voting. Clarkson et al [Clarkson 2005] have devised a remote implementation of Prêt à Voter with increased resistance to coercion. Voters are issued with capabilities, and can choose to vote with either a valid capability or a random string, for example, if threatened by a coercer. Valid capabilities, and hence associated votes, are only identified during the tabulation phase. The coercer has no way to distinguish between valid and invalid capabilities.

More recently, Ryan et al have [Ryan 2006] have replaced the original RSA construction of the ballot forms with randomising algorithms, specifically ElGamal and Paillier. This enables the replacement of decryption mixes with re-encryption mixes. Although this introduces several advantages over the original scheme, the key innovation is that ballot form creation is distributed among a number of entities. This mitigates the danger of information leaks that could occur if the ballot form information is held by a single authority.

Furthermore, it allows ballot forms to be stored and distributed in encrypted form, so countering chain of custody and chain voting threats.

In [Ryan 2005], [Peacock 2005], Peacock et al take initial steps towards a full threat analysis of cryptographic election schemes, based on Prêt à Voter, but using other voting schemes to provide a comparison. The analysis described in [Ryan 2005] considers the system in its entirety, while [Peacock 2005] concentrates on and proposes a definition for coercion-resistance. Melding the two, and incorporating other important requirements (see below) of voting systems, is the subject of current research.

In [Lundin 2006], Lundin et al propose re-introducing visual crypto into Prêt à Voter in order to enforce the destruction of the candidate order.

7.4.7. Chaum's PunchScan

Following the introduction of Prêt à Voter, Chaum proposed a new scheme [PunchScan 2006] that incorporates two independent candidate order randomisations per ballot form. A ballot form comprises two layers of paper. The upper layer has holes through which symbols printed on the lower layer show. Voters indicate their choice using a "bingo dauber" that simultaneously marks symbols on the lower layer and the surrounding paper of the upper layer.

7.4.8. Randell/Ryan scratch strip voting

Away from the purely technical aspects, Randell et al [Randell 2005] explore the issue of instilling voter trust in a system: retaining simplicity and familiarity, at the same time maximising security. They propose a variation of the original Prêt à Voter that seeks to provide the same guarantees but without using cryptography. In this scheme, a code relating to the candidate ordering (OCN) on a ballot form is covered with a printed scratch strip. This strip is overprinted with a receipt identification number (RIN). The scratch strips are only removed during vote counting, simultaneously revealing the OCN and destroying the RIN. The process is roughly analogous to the anonymising mix/tabulation of Prêt à Voter. The OCN is used to interpret the value of a vote, while the RIN is used to check that the ballot has been accurately included in the tabulation.

7.4.9. Adida/Rivest Scratch and Vote

In [Adida 2006] Adida et al propose a further enhancement to Prêt à Voter and PunchScan, called Scratch and Vote. Scratch and Vote has a significant advantage in that allows off-line auditing of ballot forms, i.e. all the information needed for this task is contained on each ballot form, but concealed by a scratch strip. Removing the strip reveals the information enabling audit but voiding the ballot form for voting. There is no need to depend on the availability of the tellers to return the seed values to enable the verification of the construction of ballot forms. They utilise homomorphic tabulation, in which election officials only cooperate to decrypt a single tally per race.

7.4.10. Rivest's ThreeBallot scheme

At the time of writing, a remarkable new scheme has been proposed by Rivest, [Rivest 2006]. This achieves voter-verifiability and unconditional privacy without using any cryptography. In essence, voters cast three ballots in such a way as to allocate two votes to their candidate of choice and exactly one vote to all other

candidates. The votes should be randomly assigned to the three ballots and at the time of casting the voter should randomly select one of the ballots to be copied and retained as a receipt. All three ballots are cast, recorded and posted to a secure WBB. At the time of casting, each voter retains one out of their three ballots, chosen arbitrarily. This chosen form they can check against the WBB. As a result, an attempt to corrupt a ballot stands a 1/3 chance of being detected.

Asterix		Asterix	X	Asterix	
Idefix		Idefix	X	Idefix	X
Obelix	X	Obelix		Obelix	
Panoramix	X	Panoramix		Panoramix	
377209628		374811209		980344216	

Figure 5: A ThreeBallot vote for Idefix.

Asterix	
Idefix	X
Obelix	
Panoramix	
980344216	

Figure 6: possible receipt for the above vote.

The tabulation is easy to perform and universally verifiable. Suppose that there are n voters. Votes for each candidate are totalled over all the 3n posted ballots. n is subtracted from the total for each candidate. The remaining values represent the votes cast for each candidate!

7.5. Scalability and Interoperability

For most of the schemes described here, scalability seems not to be a major issue: most work factors scale roughly linearly. Interoperability with surrounding systems, for example the electoral register, may be an issue. This tends not to be addressed in the crypto literature but is addressed by various official standards such as EML, the Election Mark-up language.

Conclusions and prospects

Significant strides have been made in the last few years towards trustworthy voting systems. Indeed, one could argue that such system strive to be trust-free, i.e., they seek to avoid the need to place trust in devices, processes or officials.

There is doubtless scope for further innovations and simplifications of these schemes. Clearly, far more analysis of these schemes is required, and such analysis will have to be extended to a full systems based approach taking full account of the surrounding socio-technical system.

Whilst such schemes appear to provide high levels of trustworthiness, at least in the eyes of experts, it is not clear that the public at large would understand and appreciate the rather subtle arguments and so be prepared to place as much trust in them as say the familiar pen and paper.

This leads us to some fascinating socio-technical questions:

- To what extent could the properties of such cryptographic systems be explained to the general public?
- How easy would the electorate find such schemes to use.
- To what extent is it necessary for the general public to understand in order to have sufficient trust in such systems?
- To what extent are the assurances of independent, impartial experts enough to engender sufficient trust?
- To what extent might it be necessary to compromise on trustworthiness in order to achieve understandability, by, for example, replacing cryptographic mechanisms with simpler technology or processes?

These issues are important to consider along with future technological advancements of voting schemes.

Another area that has seen very little exploration to date is that of effective recovery strategies. Most of the above schemes define precise procedures for detecting errors or corruption, but typically say little about how to respond when errors are detected. Clearly we do not want to abort an election if only a smattering of errors are detected, especially if these are negligible compared to the margin of the count. Where should the threshold be placed at which recovery actions are be triggered?

In the view of the authors, voter-verifiable schemes are reaching a sufficient level of maturity for serious consideration to be given to trails and eventual deployment in real elections and referenda. The low level of public confidence in existing systems, particularly in the US, suggests that the time is ripe for truly high-assurance voting systems.

References

[Adelsbach et al., 2002] Adelsbach, A., Alessandri, D., Cachin, C., Creese, S., Deswarte, Y., Kursawe, K., Laprie, J. C., Powell, D., Randell, B., Riordan, J., Ryan, P., Simmonds, W., Stroud, R., Veríssimo, P., Waidner, M., and Wespi, A. (2002). Conceptual Model and Architecture of MAFTIA. Project MAFTIA deliverable D21.

[Adida 2006] B. Adida and R. L. Rivest, Scratch & Vote: Self-Contained Paper-Based Cryptographic Voting, Proceedings of the Workshop on Privacy in the Electronic Society, October 2006.

[Afek and Merritt 1999] Afek Y. and Merritt M., Fast,Wait-Free (2*k*-1)-Renaming. Proc. 18th ACM Symposium on Principles of Distributed Computing (PODC'99), ACM Press, pp. 105-112, Atlanta (GA), 1999.

[Afek et al. 1993] Afek Y., Attiya H., Dolev D., Gafni E., Merritt M. and Shavit N., Atomic Snapshots of Shared Memory. Journal of the ACM, 40(4):873-890, 1993.

[Agrawal 1988] P. Agrawal, "Fault Tolerance in Multiprocessor Systems without Dedicated Redundancy," IEEE Trans. on Computers, 37(3), pp. 358–362, Mar. 1988.

[Aguilera 2004] Aguilera M.K., A Pleasant Stroll Through the Land of Infinitely Many Creatures. ACM SIGACT News, Distributed Computing Column, 35(2):36-59, 2004.

[Aguilera, 2004] Aguilera, M. (2004). A pleasant stroll through the land of infinitely many creatures. ACM SIGACT News, 35(2):36–59.

[Aho and Ullman 1992] Aho A.V. and Ullman J.D., Foundations of Computer Science. Computer Science press, 765 pages, 1992.

[Allavena et al. 2006] A. Allavena, A. Demers and J.E. Hopcroft, "Correctness of a gossip based membership protocol", Annual ACM Symposium on Principles of Distributed Computing archive Proceedings of the twenty-fourth annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing table of contents., 2006.

[Alon et al., 2005] Alon, N., Merrit, M., Reingold, O., Taubenfeld, G., and Wright, R. (2005). Tight bounds for shared memory systems acessed by Byzantine processes. Distributed Computing, 18(2):99–109.

[Asokan et al. 1997] N. Asokan, M. Schunter, and M. Waidner. Optimistic Protocols for Fair Exchange. In Proceedings of the 4th ACM Conference on Computer and Communications Security, Zurich, April 1997.

[Attie, 2002] Attie, P. C. (2002). Wait-free Byzantine consensus. Information Processing Letters, 83(4):221–227.

[Attiya and Fouren 2000] Attiya H. and Fouren A., Polynomial and Adaptive Long-lived (2*k*-1)-Renaming. Proc. Symposium on Distributed Computing (DISC'00), Springer-Verlag LNCS #1914, pp. 149-163, Toledo (Spain), 2000.

[Attiya and Fouren 2001] Attiya H. and Fouren A., Adaptive and Efficient Algorithms for Lattice Agreement and Renaming. SIAM Journal of Computing, 31(2):642-664, 2001.

[Attiya and Rachman 1998] Attiya H. and Rachman O., Atomic Snapshots in O(*n* log *n*) Operations. SIAM Journal on Computing, 27(2):319-340, 1998.

[Attiya and Welch 1998] Attiya H. and Welch J., Distributed Computing: Fundamentals, Simulations and Advanced Topics, McGraw-Hill, 451 pages, 1998.

[Attiya et al. 1990] Attiya H., Bar-Noy A., Dolev D., Peleg D. and Reischuk R., Renaming in an Asynchronous Environment. Journal of the ACM, 37(3):524-548, 1990.

[Avizienis et al. 2004] A. Avizienis, J.-C. Laprie, B. Randell and C. Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing", IEEE Trans. on Dependable and Secure Computing, 1(1), pp. 11-33, January-March 2004.

[Avizienis et al., 2004] Avizienis, A., Laprie, J.-C., Randell, B., and Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing, 1(1):11–33.

[Babaoglu et al. 2002] O. Babaoglu, H. Meling, and A. Montresor, "Anthill: A framework for the development of agent-based peer-to-peer systems", In Proceedings of the 22th International Conference on Distributed Computing Systems (ICDCS'02), Vienna, Austria, July 2002.

[Baldoni et al. 2005] R. Baldoni, C. Marchetti, A. Virgillito, R. Vitenberg, "Content-Based Publish-Subscribe over Structured Overlay Networks", 25th IEEE International Conference on Distributed Computing Systems (ICDCS), 2005.

[Baldoni et al. 2005-12] R.Baldoni, A. Noor Mian, S. Scipioni and S. Tucci Piergiovanni, "Churn Resilience of Peer-to-Peer Group Membership: a Performance Analysis", InternationalWorkshop on Distributed Computing, Kharagpur, India, December 2005.

[Baldoni et al. 2006] R. Baldoni, S. Bonomi, L. Querzoni, A. Rippa, S. Tucci Piergiovanni, A. Virgillito, "Fighting Erosion in Dynamic Large-Scale Overlay Networks", Technical Report - Midlab 9/06 , Dip. Informatica e Sistemistica "Antonio Ruberti", Universit di Roma "La Sapienza", 2006.

[Baldoni et al. 2006-07] R. Baldoni, S. Bonomi, L. Querzoni, A. Rippa, S. Tucci Piergiovanni and A. Virgillito, "Evaluation of Unstructured Overlay Maintenance Protocols under Churn", IWDDS 2006 co-located with ICDCS2006.

[Baldoni et al. 2006-10] R. Baldoni, M. Malek, A. Milani, S. Tucci Piergiovanni, "Weakly- Persistent Causal Objects In Dynamic Distributed Systems", To appear in proc. of SRDS 2006, october 2006, Leeds (UK).

[Baldoni et al. 2006-11] R. Baldoni, R. Guerraoui, R. Levy, V. Quema, S. Tucci Piergiovanni, "Unconscious Eventual Consistency with Gossips", To appear in Proc. of SSS 2006, November 2006, Dallas (USA).

[Baldoni et al., 2003] Baldoni, R., Helary, J., Raynal, M., and Tanguy, L. (2003). Consensus in Byzantine asynchronous systems. Journal of Discrete Algorithms, 1(2):185–210.

[Banâtre et al. 2004] M. Banâtre, P. Couderc, J. Pauty, and M. Becus. Ubibus: Ubiquitous Computing to Help Blind People in Public Transport. In proceedings of Mobile HCI 2004, pages 310-314, 2004.

[Barborak et al. 1993] M. Barborak, M. Malek and A. Dahbura, "The Consensus Problem in Fault Tolerant Computing," ACM Surveys, vol. 25, pp. 171–220, Jun. 1993.

[Barreto et al. 2002] D. Barreto, Y. Liu, J. Pan, and F. Wang, "Reputation-based participation enforcement for ad hoc networks", 2002.

[Batten et al., 2001] Batten, C., Barr, K., Saraf, A., and Treptin, S. (2001). pStore: a secure peer-to-peer backup system. Technical Report MIT-LCS-TM-632, MIT Laboratory for Computer Science.

[Ben Azzouna and Guillemin 2004] N. Ben Azzouna and F. Guillemin, "Experimental analysis of the impact of peer-to-peer applications on traffic in commercial IP networks", European transactions on Telecommunications: Special issue on P2P networking and P2P services, ETT 15(6), November-December 2004.

[Beneloh 1991] J. Beneloh, Verifiable Secret-Ballot, PhD thesis, Yale University, 1987.

[Ben-Or, 1983] Ben-Or, M. (1983). Another advantage of free choice: Completely asynchronous agreement protocols. In Proceedings of the 2nd ACM Symposium on Principles of Distributed Computing, pages 27–30.

[Ben-Or, 1985] Ben-Or, M. (1985). Fast asynchronous Byzantine agreement. In Proceedings of the 4th ACM Symp. on Principles of Distributed Computing, pages 149–151.

[Bessani et al., 2006] Bessani, A. N., Correia, M., Fraga, J. S., and Lung, L. C. (2006). Sharing memory between Byzantine processes using policy-enforced tuple spaces. In Proceedings of the 26th International Conference on Distributed Computing Systems.

[Blough and Brown 1999] D.M. Blough and H.W. Brown, "The Broadcast Comparison Model for On-Line Fault Diagnosis in Multicomputer Systems: Theory and Implementation," IEEE Trans. on Computers, 48(5), pp. 470–493, May 1999.

[Blount 1977] M. Blount, "Probabilistic treatment of diagnosis in digital systems," Proc. of 7th Int'l IEEE Symp. of Fault-Tolerant Computing (FTCS), pp. 72-77, 1977.

[Blum 1991] M. Blum, A. De Santis, S. Micali, and G. Persiano, Noninteractive Zero-knowledge, SIAM Journal on Computing, 20(6):1084-1118, 1991.

[Bolosky et al., 2000] Bolosky, W. J., Douceur, J. R., Ely, D., and Theimer, M. (2000). Feasibility of a serverless distributed file system deployed on an existing set of desktop pcs. In Proceedings of the International Conference on Measurement and Modeling of Computer Systems, pages 34–43.

[Bolton and Ockenfels 2000] G. E Bolton and A. Ockenfels, "ERC: a theory of equity, reciprocity, and competition", American Economic Review 90(1): 166-193, 2000.

[Bondavalli et al. 1997] A. Bondavalli, S. Chiaradonna, F. Di Giandomenico and F. Grandoni, "Discriminating Fault Rate and Persistency to Improve Fault Treatment," Proc. of FTCS-27, pp. 354–362, 1997.

[Bondavalli et al. 2000] A. Bondavalli, S. Chiaradonna, F. Di Giandomenico and F. Grandoni, "Threshold-Based Mechanisms to Discriminate Transient from Intermittent Faults," IEEE Trans. on Computers, 49(3), pp. 230–245, Mar. 2000.

[Bondavalli et al. 2004] A. Bondavalli, S. Chiaradonna, D. Cotroneo and L. Romano, "Effective Fault Treatment for Improving the Dependability of COTS and Legacy-Based Applications" IEEE Trans. On Dependable and Secure Computing, 1(4), pp. 223-237, Dec. 2004

[Borowsky and Gafni 1993-1] Borowsky E. and Gafni E., Immediate Atomic Snapshots and Fast Renaming. Proc. 12th ACM Symposium on Principles of Distributed Computing (PODC'93), ACM Press, pp. 41-51, Ithaca (NY), 1993.

[Borowsky and Gafni 1993-2] Borowsky E. and Gafni E., Generalized FLP Impossibility Results for . -Resilient Asynchronous Computations. Proc. 25th ACM Symposium on Theory of Computation (STOC'93), San Diego (CA), pp. 91-100, 1993.

[Bracha, 1984] Bracha, G. (1984). An asynchronous b(n−1)/3c-resilient consensus protocol. In Proceedings of the 3rd ACM Symposium on Principles of Distributed Computing, pages 154–162.

[Brown and Kolberg 2006] A. Brown, M. Kolberg, "Tools for Peer-to-Peer Network Simulation", March 3rd, 2006, http://www.ietf.org/internet-drafts/draft-irtf-p2prg-core-simulators-00.txt

[Buchegger and Le Boudec 2002] S. Buchegger, and J. Y. Le Boudec, "Performance Analysis of the CONFIDANT Protocol: Cooperation Of Nodes — Fairness In Distributed Ad-hoc NeTworks", In Proceedings of IEEE/ACM Workshop on Mobile Ad Hoc Networking and Computing (MobiHOC), Lausanne, CH, IEEE (2002) 226–236, 2002.

[Bussard and Molva 2004] L. Bussard, R. Molva. One-Time Capabilities for Authorizations without Trust. In Proceedings of the second IEEE International Conference on Pervasive Computing and Communications (PerCom 2004), Orlando, Florida, March 13-17, 2004, pages 351-355.

[Bussard et al. 2004] L. Bussard, Y. Roudier, R. Molva. Untraceable Secret Credentials: Trust Establishment with Privacy. In Proceedings of the First IEEE International Workshop on Pervasive Computing and Communication Security (PerSec 2004), Orlando, Florida, March 14, 2004.

[Buttyán and Hubaux 2000] L. Buttyan and J.-P. Hubaux, Toward a formal model of fair exchange -- a game theoretic approach, 2000, http://citeseer.ist.psu.edu/article/buttyan00toward.html, updated version of the SSC Technical Report No. SSC/1999/039

[Buttyán and Hubaux 2001] L. Buttyán and J. Hubaux, "Nuglets: a virtual currency to stimulate cooperation in self-organized ad hoc networks", Technical report, EPFL, 2001.

[Buttyán and Hubaux 2003] L. Buttyán and J.-P Hubaux, "Stimulating Cooperation in Self-Organizing Mobile Ad Hoc Networks", ACM/Kluwer Mobile Networks and Applications, 8(5), October 2003.

[Byers et al., 1998] Byers, J. W., Luby, M., Mitzenmacher, M., and Rege, A. (1998). A digital fountain approach to reliable distribution of bulk data. In SIGCOMM, pages 56–67.

[Byers et al., 1999] Byers, J. W., Luby, M., and Mitzenmacher, M. (1999). Accessing multiple mirror sites in parallel: Using Tornado codes to speed up downloads. In INFOCOM (1), pages 275–283.

[Cachin et al., 2000] Cachin, C., Kursawe, K., and Shoup, V. (2000). Random oracles in Contanstinople: Practical asynchronous Byzantine agreement using cryptography. In Proceedings of the 19th ACM Symposium on Principles of Distributed Computing, pages 123–132.

[Cachin et al., 2001] Cachin, C., Kursawe, K., Petzold, F., and Shoup, V. (2001). Secure and efficient asynchronous broadcast protocols (extended abstract). In Kilian, J., editor, Advances in Cryptology: CRYPTO 2001, volume 2139 of Lecture Notes in Computer Science, pages 524–541. Springer-Verlag.

[Canetti and Rabin, 1993] Canetti, R. and Rabin, T. (1993). Fast asynchronous Byzantine agreement with optimal resilience. In Proceedings of the 25th Annual ACM Symposium on Theory of Computing, pages 42–51.

[Carbone et al. 2003] M. Carbone, M. Nielsen, and V. Sassone, "A Formal Model for Trust in Dynamic Networks", BRICS tech. report RS-03-4, Univ. Aarhus, 2003.

[Castro and Liskov, 2002] Castro, M. and Liskov, B. (2002). Practical Byzantine fault tolerance and proactive recovery. ACM Transactions on Computer Systems, 20(4):398–461.

[Chandra and Toueg, 1996] Chandra, T. and Toueg, S. (1996). Unreliable failure detectors for reliable distributed systems. Journal of the ACM, 43(2):225–267.

[Chandra et al., 1996] Chandra, T., Hadzilacos, V., and Toueg, S. (1996). The weakest failure detector for solving consensus. Journal of the ACM, 43(4):685–722.

[Chaudhuri, 1993] Chaudhuri, S. (1993). More choices allow more faults: Set consensus problems in totally asynchronous systems. Information and Computation, 105(1):132–158.

[Chaum 1992] D. Chaum, Blind Signature for Untraceable Payments, Advances in Cryptology: Crypto'82, pp 199-203, 1992.

[Chaum 2004] D. Chaum, Secret-Ballot Receipts: True Voter-Verifiable Elections, IEE Security & Privacy, Vol. 2, No. 1, pp. 38-47, January-February 2004.

[Chervenak et al., 1998] Chervenak, A., Vellanki, V., and Kurmas, Z. (1998). Protecting file systems: A survey of backup techniques. In Proceedings Joint NASA and IEEE Mass Storage Conference.

[Chockler et al., 2001] Chockler, G., Keidar, I., and Vitenberg, R. (2001). Group communication specifications: A comprehensive study. ACM Computing Surveys, 33(4):427–469.

[Clarke et al. 2001] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, "Freenet: A distributed anonymous information storage and retrieval system", In Designing Privacy Enhancing Technologies: International Workshop on Design Issues inAnonymity and Unobservability, LNCS 2009, New York, 2001.

[Clarkson 2005] M. Clarkson and A. Myers, Coercion-resistant Remote Voting using Decryption Mixes, Workshop on Frontiers of Electronic Elections, 2005.

[Cooley et al., 2004] Cooley, J., Taylor, C., and Peacock, A. (2004). ABS: the apportioned backup system. Technical report, MIT Laboratory for Computer Science.

[Cooper and Garcia-Molina, 2002] Cooper, B. F. and Garcia-Molina, H. (2002). Bidding for storage space in a peer-topeer data preservation system. In ICDCS, pages 372–.

[Correia et al., 2002] Correia, M., Veríssimo, P., and Neves, N. F. (2002). The design of a COTS real-time distributed security kernel. In Proceedings of the Fourth European Dependable Computing Conference, pages 234–252.

[Correia et al., 2005] Correia, M., Neves, N. F., Lung, L. C., and Veríssimo, P. (2005). Low complexity Byzantineresilient consensus. Distributed Computing, 17(3):237–249.

[Correia et al., 2006a] Correia, M., Bessani, A. N., Neves, N. F., Lung, L. C., and Veríssimo, P. (2006a). Improving byzantine protocols with secure computational components. In the report.

[Correia et al., 2006b] Correia, M., Neves, N. F., Lung, L. C., and Veríssimo, P. (2006b). Worm-IT – a wormhole-based intrusion-tolerant group communication system. Journal of Systems and Software. to appear.

[Correia et al., 2006c] Correia, M., Neves, N. F., and Veríssimo, P. (2006c). From consensus to atomic broadcast: Timefree Byzantine-resistant protocols without signatures. Computer Journal, 41(1):82–96.

[Courtés et al., 2006] Courtés, L., Killijian, M.-O., and Powell, D. (2006). Storage tradeoffs in a collaborative backup service for mobile devices. In Proceedings of the 6th European Dependable Computing Conference (EDCC-6), number LAAS Report #05673, pages 129–138, Coimbra, Portugal.

[Cox and Noble 2002] L. P. Cox and B. D. Noble, "Pastiche: making backup cheap and easy", in Proceedings of the Fifth USENIX Symposium on Operating Systems Design and Implementation, Boston, MA, December 2002.

[Cox and Noble, 2002] Cox, L. P. and Noble, B. D. (2002). Pastiche: making backup cheap and easy. In Fifth USENIX Symposium on Operating Systems Design and Implementation, pages 285–298, Boston, MA, USA.

[Cox and Noble, 2003] Cox, L. P. and Noble, B. D. (2003). Samsara: honor among thieves in peer-to-peer storage. In Proceedings 19th ACM Symposium on Operating Systems Principles, pages 120–132, Bolton Landing, NY, USA.

[Cristian 1991] F. Cristian, "Reaching agreement on processor-group membrship in synchronous distributed systems," Distributed Computing, 4(4), pp. 175–187, Dec. 1991.

[Cristian and Fetzer, 1998] Cristian, F. and Fetzer, C. (1998). The timed asynchronous system model. In Proceedings of the 28th IEEE International Symposium on Fault-Tolerant Computing, pages 140–149.

[Dabek et al., 2001] Dabek, F., Kaashoek, M. F., Karger, D., Morris, R., and Stoica, I. (2001). Wide-area cooperative storage with CFS. In Proceedings 18th ACM Symposium on Operating Systems Principles, pages 202–215.

[de Prisco et al., 1999] de Prisco, R., Malki, D., and Reiter, M. (1999). On k-set consensus problems in asynchronous systems. In Proceedings of the 18th ACM Symposium on Principles of Distributed Computing, pages 257–265.

[Deswarte et al., 1998] Deswarte, Y., Kanoun, K., and Laprie, J. C. (1998). Diversity against accidental and deliberate faults. In Computer Security, Dependability, & Assurance: From Needs to Solutions. IEEE Press.

[Dill 2006] D. Dill, http://www.verifiedvoting.org/

[Dingledine 2000] R. Dingledine, "The Free Haven project: Design and deployment of an anonymous secure data haven", Master's thesis, MIT, June 2000.

[Dolev et al., 1987] Dolev, D., Dwork, C., and Stockmeyer, L. (1987). On the minimal synchronism needed for distributed consensus. Journal of the ACM, 34(1):77–97.

[Doudou and Schiper, 1997] Doudou, A. and Schiper, A. (1997). Muteness detectors for consensus with Byzantine processes. Technical Report 97/30, EPFL.

[Doudou et al., 2002] Doudou, A., Garbinato, B., and Guerraoui, R. (2002). Encapsulating failure detection: From crashstop to Byzantine failures. In International Conference on Reliable Software Technologies, pages 24–50.

[Druschel and Rowstron 2000] P. Druschel and A. Rowstron, "PAST: A large-scale, persistent peer-to-peer storage utility", in Proceedings of HotOS VIII, May 2001.

[Dutta et al., 2005] Dutta, P., Guerraoui, R., and Vukolic, M. (2005). Best-case complexity of asynchronous Byzantine consensus. Technical Report 200499, ´ Ecole Polytechnique F´ed´erale de Lausanne, Lausanne, Switzerland.

[Dwork et al., 1988] Dwork, C., Lynch, N., and Stockmeyer, L. (1988). Consensus in the presence of partial synchrony. Journal of the ACM, 35(2):288–323.

[Elnikety et al., 2002] Elnikety, S., Lillibridge, M., and Burrows, M. (2002). Peer-to-peer cooperative backup system. In The USENIX Conference on File and Storage Technologies, Monterey, California, USA.

[Felton 206] E. Felton et al, http://itpolicy.princeton.edu/voting/

[Fetzer and Cristian, 1995] Fetzer, C. and Cristian, F. (1995). On the possibility of consensus in asynchronous systems. In Proceedings of the Pacific Rim International Symposium on Fault-Tolerant Systems.

[Fiat 1986] A. Fiat, and A. Shamir, How to prove yourself", Proceedings of CRYPTO '86, Lecture Notes in Computer Science, Vol. 263, pp 186-194, Springer-Verlag, 1986.

[Fischer et al., 1985] Fischer, M. J., Lynch, N. A., and Paterson, M. S. (1985). Impossibility of distributed consensus with one faulty process. Journal of the ACM, 32(2):374–382.

[Fisher and Merritt 2003] Fischer M.J. and Merritt M., Appraising Two Decades of Distributed Computing Results. Distributed Computing, 16(4):239-247, 2003.

[Fraga and Powell, 1985] Fraga, J. S. and Powell, D. (1985). A fault- and intrusion-tolerant file system. In Proceedings of the 3rd International Conference on Computer Security, pages 203–218.

[Friedman et al., 2002] Friedman, R., Mostefaoui, A., Rajsbaum, S., and Raynal, M. (2002). Distributed agreement and its relation with error-correcting codes. In Proceedings of the 16th International Conference on Distributed Computing, pages 63–87.

[Friedman et al., 2005] Friedman, R., Mostefaoui, A., and Raynal, M. (2005). Simple and efficient oracle-based consensus protocols for asynchronous byzantine systems. IEEE Transactions on Dependable and Secure Computing, 2(1):46–56.

[Fujioka 1992] A. Fujioka, T. Okamoto and K. Ohta, A Practical Secret Voting Scheme for Large Scale Elections, Auscrypt '92, LNCS 718, pp 244-251, 1992

[Gafni 2004] Gafni E., Read/Write Reductions. DISC/GODEL presentation given as introduction to the 18th Int'l Symposium on Distributed Computing (DISC'04), 2004. http://www.cs.ucla.edu/˜ eli/eli/godel.ppt.

[Gafni and Rajsbazum 2005] Gafni E. and Rajsbaum S., Musical Benches. Proc. 19th Int'l Symposium on Distributed Computing (DISC'05), Springer Verlag LNCS #3724, pp. 63–77, 2005.

[Gafni et al. 2006] Gafni E., Rajsbaum R., Raynal M. and Travers C., The Committee Decision Problem. Proc. 8th Latin-American Theoretical INformatics Symposium (LATIN'06), Springer Verlag LNCS #3887, pp. 502-514, 2006.

[Ganesh et al. 2003] A.J. Ganesh, A.M. Kermarrec and L. Massouli´e, "Peer-to-Peer Membership Management for Gossip-Based Protocols", IEEE Trans. Computers 52(2): pp. 139-149, 2003

[Goldberg and Yianilos, 1998] Goldberg, A. V. and Yianilos, P. N. (1998). Towards an archival intermemory. In Proceedings IEEE International Forum on Research and Technology Advances in Digital Libraries (ADL'98), pages 147–156. IEEE Society.

[Goldreich 1991] O. Goldreich, S Micali, A. Wigderson, Proofs That Yield Nothing But Their Validity, Journal of the ACM, volume 38, issue 3, pp 690-728. July 1991.

[Grandoni et al. 1998] F. Grandoni, S. Chiaradonna, and A. Bondavalli. A new heuristic to discriminate transient from intermittent faults. In 3rd IEEE High Assurance System Engineering Symposium (HASE'98), pages 224-231, Bethesda, MD, USA, 1998.

[Grandoni et al. 2001] F. Grandoni, S. Chiaradonna, F. Di Giandomenico and A. Bondavalli, "Evaluation of Fault-Tolerant Multiprocessor Systems for High Assurance Applications", The Computer Journal, Vol 44, N. 6, 2001, pp.544-556.

[Greco and Floridi, 2003] Greco, G. M. and Floridi, L. (2003). The tragedy of the digital commons. Ethics and Information Technology, 6(2):73.

[Grothoff, 2003] Grothoff, C. (2003). An excess-based economic model for resource allocation in peer-to-peer networks. Wirtschaftsinformatik.

[Growchowski, 1998] Growchowski, E. (1998). Emerging trends in data storage on magnetic hard disk drives. In Datatech, pages 11–16. ICG Publishing.

[Guerraoui and Raynal, 2004] Guerraoui, R. and Raynal, M. (2004). The information structure of indulgent consensus. IEEE Transactions on Computers, 53(4):453.

[Guerraoui and Schiper, 2001] Guerraoui, R. and Schiper, A. (2001). The generic consensus service. IEEE Transactions on Software Engineering, 27(1):29–41.

[Guerraoui et al., 2000] Guerraoui, R., Hurfin, M., Mostefaoui, A., Oliveira, R., Raynal, M., and Schiper, A. (2000). Consensus in asynchronous distributed systems: A concise guided tour. In Krakowiak, S. and

Shrivastava, S., editors, Advances in Distributed Systems, number 1752 in Lecture Notes in Computer Science, pages 33–47. Springer-Verlag.

[Guerraoui, 2000] Guerraoui, R. (2000). Indulgent algorithms. In Proceedings of the 19th ACMSymposium on Principles of Distributed Computing, pages 289–298.

[Gumbel 2005] A. Gumbel, Steal This Vote, Nation Books, July 10, 2005

[Hadzilacos and Toueg, 1994] Hadzilacos, V. and Toueg, S. (1994). A modular approach to fault-tolerant broadcasts and related problems. Technical Report TR94-1425, Cornell University, Department of Computer Science.

[Hardin, 1968] Hardin, G. (1968). The tragedy of the commons. Science, 162:1243–1248.

[Heckmann and Bock, 2002] Heckmann, O. and Bock, A. (2002). The eDonkey 2000 Protocol. Technical Report KOMTR- 08-2002, Multimedia Communications Lab, Darmstadt University of Technology.

[Herlihy 1991] Herlihy M.P., Wait-Free Synchronization. ACM Transactions on Programming Languages and Systems, 13(1):124-149, 1991.

[Herlihy andWing 1990] Herlihy M.P. and Wing J.M, Linearizability: a Correctness Condition for Concurrent Objects. ACM Transactions on Programming Languages and Systems, 12(3):463-492, 1990.

[Hsu et al., 2004] Hsu, E., Mellen, J., and Naresh, P. (2004). DIBS: distributed backup for local area networks. Technical report, Parallel & Distributed Operating Systems Group, MIT, USA.

[Ioannidis et al., 2002] Ioannidis, J., Ioannidis, S., Keromytis, A. D., and Prevelakis, V. (2002). Fileteller: Paying and getting paid for file storage. In Sixth Annual Conference on Financial Cryptography, page 282299, Bermuda.

[Iyer et al. 1990] R. Iyer, L.T. Young and P.V.K. Iyer "Automatic Recognition of Intermittent Failures: An Experimental Study of Field Data," IEEE Trans. on Computers, 39(3), pp. 525-537, Apr. 1990.

[Jakobsson 1999] M. Jakobsson and A. Juels, Millimix: Mixing in small batches. Technical Report 99-33, DIMACS, 1999

[Jakobsson 2002] M. Jakobsson, A. Juels, and R. Rivest. Making Mix Nets Robust for Electronic Voting By Randomized Partial Checking, Proceedings of USENIX Security Symposium 2002, pp 339-353, 2002

[Jakobsson et al. 2003] M. Jakobsson, J.-P. Hubaux, and L. Buttyan, "A Micro-Payment Scheme Encouraging Collaboration in Multi-Hop Cellular Networks", In Proceedings of Financial Crypto, La Guadeloupe, Jan. 2003.

[Jelasity et al. 2003] M. Jelasity,W. Kowalczyk and M. van Steen, "Newscast computing", Technical Report IR-CS-006, Vrije Universiteit Amsterdam, Department of Computer Science, Amsterdam, The Netherlands, November 2003.

[Jelasity et al. 2004] M. Jelasity, R. Guerraoui, A.M. Kermarrec and M. van Steen, "The peer sampling service: Experimental evaluation of unstructured gossip-based implementations", In Hans-Arno Jacobsen, editor, Middleware 2004, volume 3231 of Lecture Notes in Computer Science, pages 7998. Springer-Verlag, 2004.

[Jøsang and Ismail 2002] A. Jøsang and R. Ismail. The Beta Reputation System. In Proceedings of the 15th, Bled Electronic Commerce Conference, Bled, Slovenia, June 2002.

[Jøsang et al. 2005] A. Jøsang, R. Ismail, and C. Boyd, "A Survey of Trust and Reputation Systems for Online Service Provision", In Proceedings of Decision Support Systems, 2005.

[Junqueira et al., 2003] Junqueira, F., Bhagwan, R., Marzullo, K., Savage, S., and Voelker, G. M. (2003). The Phoenix recovery system: rebuilding from the ashes of an internet catastrophe. In Ninth Workshop on Hot Topics in Operating Systems (HotOS IX).

[Kamvar et al. 2003] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, "The EigenTrust Algorithm for Reputation Management in P2P Networks", In Proceedings of the Twelfth International World Wide Web Conference, Budapest, May 2003.

[Karlof 2005] C. Karlof and N. Sastry and D. Wagner, Cryptographic Voting Protocols: A Systems Perspective, USENIX Security Symposium, 2005.

[Keidar, 2002] Keidar, I. (2002). Challenges in evaluating distributed algorithms. In Proceedings of the International Workshop on Future Directions in Distributed Computing.

[Kihlstrom et al., 2001] Kihlstrom, K. P., Moser, L. E., and Melliar-Smith, P. M. (2001). The SecureRing group communication system. ACM Transactions on Information and System Security, 4(4):371–406.

[Kihlstrom et al., 2003] Kihlstrom, K. P., Moser, L. E., and Melliar-Smith, P. M. (2003). Byzantine fault detectors for solving consensus. The Computer Journal, 46(1):16–35.

[Killijian et al. 2004] M.O. Killijian, D. Powell, M. Banâtre, P. Couderc, Y. Roudier, Collaborative Backup for Dependable Mobile Applications. In proceedings of the 2nd International Workshop on Middleware for Pervasive and Ad-Hoc Computing, Middleware 2004, Toronto, Ontario, Canada, October 18th - 22nd, 2004, ACM.

[Killijian et al. 2006] M.-O. Killijian, M. Banâtre, C. Bryce, L. Blain, P. Couderc, L. Courtès, Y. Deswarte, D. Martin-Guillerez, R. Molva, N. Oualha, D. Powell, Y. Roudier, I. Sylvain, "MoSAIC: Mobile System Availability Integrity and Confidentiality", Progress Report, June 2006.

[Killijian et al., 2004] Killijian, M.-O., Powell, D., Banˆatre, M., Couderc, P., and Roudier, Y. (2004). Collaborative backup for dependable mobile applications. In Proceedings of 2nd International Workshop on Middleware for Pervasive and Ad-Hoc Computing (Middleware 2004), pages 146–149, Toronto, Ontario, Canada. ACM.

[Klemm et al. 2003] A. Klemm, C. Lindemann, and O. Waldhorst, "A Special-Purpose Peer-to-Peer File Sharing System for Mobile Ad Hoc Networks", Proc. IEEE Semiannual Vehicular Technology Conference (VTC2003-Fall), Orlando, FL, October 2003.

[Kohno 2004] T. Kohno, A. Stubblefield, A. D. Rubin and D. S. Wallach, Analysis of an Electronic Voting System, Proceedings of the IEEE Symposium on Security and Privacy, May, 2004.

[Kopetz and Grunsteidl 1994] H. Kopetz and G. Grunsteidl, "TTP - A Protocol for Fault-Tolerant Real-Time Systems," IEEE Computer, 27(1), pp. 14–23, Jan. 1994.

[Kubiatowicz et al. 2000] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, " OceanStore: An architecture for

globalscale persistent storage", in Proceedings of the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000), Nov. 2000.

[Kubiatowicz et al., 2000] Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Weimer, W., Wells, C., and Zhao, B. (2000). OceanStore: an architecture for global-scale persistent storage. In Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000), pages 190–201.

[Kuhl and Reddy 1981] J. Kuhl, S. Reddy, "Fault Diagnosis in Fully Distributed Systems," Proc. of FTCS-11, pp. 100–105, 1981.

[Lai et al., 2003] Lai, K., Feldman, M., Chuang, J., and Stoica, I. (2003). Incentives for cooperation in peer-to-peer networks. In Workshop on Economics of Peer-to-Peer Systems.

[Lala and Alger 1988] J. Lala and L. Alger, "Hardware and Software Fault Tolerance: A Unified Architectural Approach", Proc. of FTCS-18, pp. 240–245, 1988.

[Lala, 2003] Lala, J. H., editor (2003). Foundations of Intrusion Tolerant Systems. IEEE Computer Society Press.

[Lamport 1977] Lamport L., Concurrent Reading and Writing. Communications of the ACM, 20(11):806-811, 1977.

[Lamport 1978] Lamport L., Time, Clocks, and the Ordering of Events in Distributed Systems, Communications of the ACM, 21(7):558-565, 1978.

[Lamport 1987] Lamport L., A Fast Mutual Exclusion Algorithm. ACM Transactions on Computer Systems, 5(1):1- 11, 1987.

[Lamport 1996] Lynch N.A., Distributed Algorithms. Morgan Kaufmann Pub., San Francisco (CA), 872 pages, 1996.

[Lamport and Raynal 2004] Lamport L. and Raynal M., Private discussion, 2004.

[Lamport et al., 1982] Lamport, L., Shostak, R., and Pease, M. (1982). The Byzantine generals problem. ACM Transactions on Programming Languages and Systems, 4(3):382–401.

[Lamport, 1998] Lamport, L. (1998). The part-time parliament. ACM Transactions Computer Systems, 16(2):133–169.

[Lamport, 2001] Lamport, L. (2001). Paxos made simple. SIGACT News, 32(4):51–58.

[Lampson, 2001] Lampson, B. (2001). The abcd's of paxos. In Proceedings of the 20th annual ACM Symposium on Principles of Distributed Computing, page 13.

[Landers et al., 2004] Landers, M., Zhang, H., and Tan, K.-L. (2004). PeerStore: better performance by relaxing in peer-to-peer backup. In Proceedings of the Fourth International Conference on Peer-to-Peer Computing, pages 72–79, Zurich, Switzerland.

[Latronico et al. 2004] E. Latronico, P. Miner and P. Koopman, "Quantifying the Reliability of Proven SPIDER Group Membership Service Guarantees," Proc. of DSN-04, pp. 275-284, 2004.

[Lee et al. 2003] S. Lee, R. Sherwood, B. Bhattacharjee. "Cooperative peer groups in NICE". In INFOCOM'03, April 2003.

[Liang et al. 2006] J. Liang, R. Kumar, and K.W. Ross, "The FastTrack overlay: A measurement study", Computer Networks, 50, 842-858, 2006.

[Lillibridge et al. 2003] M. Lillibridge, S. Elnikety, A. Birrell, M.Burrows, and M. Isard, "A Cooperative Internet Backup Scheme", In Proceedings of the 2003 Usenix Annual Technical Conference (General Track), pp. 29-41, San Antonio, Texas, June 2003.

[Lin and Siewiorek 1990] T. Lin and D. Siewiorek, "Error Log Analysis: Statistical Modeling and Heuristic Trend Analysis," IEEE Trans. on Computers, 39(4), pp. 419-432, Oct. 1990.

[Lincoln and Rushby 1993] P. Lincoln and J. Rushby, "A Formally Verified Algorithm for Interactive Consistency under a Hybrid Fault Model," Proc. of FTCS-23, pp. 402–411, 1993.

[Littlewood and Strigini, 2004] Littlewood, B. and Strigini, L. (2004). Redundancy and diversity in security. In Samarati, P., Rian, P., Gollmann, D., and Molva, R., editors, Computer Security – ESORICS 2004, 9th European Symposium on Research Computer Security, LNCS 3193, pages 423–438. Springer.

[Loo et al. 2002] B. T. Loo, A. LaMarca, and G. Borriello, "Peer-To-Peer Backup for Personal Area Networks", Intel Research Technical Report IRS-TR-02-15, October 2002.

[Loo et al., 2003] Loo, B. T., LaMarca, A., and Borriello, G. (2003). Peer-to-peer backup for personal area networks. Technical Report IRS-TR-02-015, UC Berkeley; Intel Seattle Research (USA).

[Lord, 2005] Lord, T. (2005). The GNU arch distributed revision control system.

[Lua et al., 2005] Lua, K., Crowcroft, J., Pias, M., Sharma, R., and Lim, S. (2005). A survey and comparison of peer-topeer overlay network schemes. Communications Surveys & Tutorials, IEEE, pages 72–93.

[Lundin 2006] D. Lundin, H. Treharne, P. Y. A. Ryan, S. Schneider, J. Heather, Z. Xia, Tear and Destroy: Chain voting and destruction problems shared by Pret `a Voter and Punchscan and a solution using Visual Encryption, Workshop on Frontiers of Electronic Elections, 2005.

[Lynch, 1989] Lynch, N. (1989). A hundred impossiblility proofs for distributed computing. In Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing.

[Lynch, 1996] Lynch, N. (1996). Distributed Algorithms. Morgan Kaufmann, San Mateo, CA.

[Malek 1980] M. Malek, "A Comparison Connection Assignment for Diagnosis of Multiprocessor Systems," Proc. of the 7th Annual Symp. on Computer Architecture, pp. 31–36, 1980.

[Malkhi and Reiter, 1997] Malkhi, D. and Reiter, M. (1997). Unreliable intrusion detection in distributed computations. In Proceedings of the 10th Computer Security Foundations Workshop, pages 116–124.

[Malkhi et al., 2003] Malkhi, D., Merrit, M., Reiter, M., and Taubenfeld, G. (2003). Objects shared by Byzantine processes. Distributed Computing, 16(1):37–48.

[Mallela and Masson 1980] S. Mallela, G. Masson, "Diagnosis without Repair for Hybrid Fault Situations," IEEE Trans. On Computers, 29, pp. 461–470, Jun. 1980.

[Marti et al. 2000] S. Marti, T.J. Giuli, K. Lai, and M. Baker, "Mitigating routing misbehavior in mobile ad hoc networks", Mobile Computing and Networking 255–265, 2000.

[Martin and Alvisi, 2005] Martin, J. P. and Alvisi, L. (2005). Fast Byzantine consensus. In Proceedings of the IEEE International Conference on Dependable Systems and Networks, pages 402–411.

[Maymounkov and Mazi`eres, 2002] Maymounkov, P. and Mazi`eres, D. (2002). Kademlia: A peer-to-peer information system based on the XOR metric. Lecture Notes in Computer Science, 2429:53–??

[McCoy 2001] J. McCoy. "Mojo Nation Responds". January 2001. http://www.openp2p.com/pub/a/p2p/2001/01/11/mojo.html.

[Mercuri 2005] R. Mercuri, http://www.notablesoftware.com/evote.html

[Merritt and Taubenfeld 2000] Merritt M. and Taubenfeld G., Computing Using Infinitely Many Processes. Proc. 14th Int'l Symposium on Distributed Computing (DISC'00), Springer-Verlag LNCS #1914, pp. 164-178, 2000.

[Meyer et al. 1985] J.F. Meyer, A. Movaghar and W.H. Sanders, "Stochastic Activity Networks: Structure, Behavior, and Application," Int'l Workshop onTimed Petri Nets, pp. 106-115, 1985.

[Michiardi 2004] P. Michiardi, "Cooperation enforcement and network security mechanisms for mobile ad hoc networks", PhD Thesis, December 14th, 2004.

[Michiardi and Molva 2002] P. Michiardi, and R. Molva, "CORE: a collaborative reputation mechanism to enforce node cooperation in mobile ad hoc networks", CMS'2002, Communication and Multimedia Security 2002 Conference, Portorosz, Slovenia, September 26-27, 2002.

[Moir 1998] Moir M., Fast, Long-Lived Renaming Improved and Simplified. Science of Computer Programming, 30:287-308, 1998.

[Moir and Anderson 1995] Moir M. and Anderson J.H., Wait-Free Algorithms for Fast, Long-Lived Renaming. Science of Computer Programming, 25:1-39, 1995.

[Mongardi 1993] G. Mongardi, "Dependable Computing for Railway Control Systems," Proc. DCCA-3 - Dependable Computing for Critical Applications, pp. 255-277, 1993.

[Moniz et al., 2006] Moniz, H., Correia, M., Neves, N. F., and Verssimo, P. (2006). Randomized intrusion-tolerant asynchronous services. In Proceedings of the International Conference on Dependable Systems and Networks.

[Montresor et al. 2004] A. Montresor, G. Di Caro, P. E. Heegaard, "Architecture of the Simulation Environment", BISON IST-2001-38923, January 29, 2004.

[Mostefaoui et al. 2006] Most´efaoui A., Raynal M. and Travers C., Exploring Gafni's reduction land: from Ω^K to wait-free adaptive $(2p - \left\lceil \frac{p}{k} \right\rceil)$-renaming via k-set agreement. Proc. 20th Symposium on Distributed Computing (DISC'06), Springer Verlag LNCS #4167, pp. 1-15, Stockholm (Sweden), 2006.

[Mostefaoui et al., 2000] Mostefaoui, A., Raynal, M., and Tronel, F. (2000). From binary consensus to multivalued consensus in asynchronous message-passing systems. Information Processing Letters, (73):207–212.

[Mostefaoui et al., 2003a] Mostefaoui, A., Mourgaya, E., and Raynal, M. (2003a). Asynchronous implementation of failure detectors. In Proceedings of the IEEE International Conference on Dependable Systems and Networks, pages 351–360.

[Mostefaoui et al., 2003b] Mostefaoui, A., Rajsbaum, S., and Raynal, M. (2003b). Conditions on input vectors for consensus solvability in asynchronous distributed systems. Journal of the ACM, 50(6):922–954.

[Mostefaoui et al., 2004] Mostefaoui, A., Rajsbaum, S., Raynal, M., and Roy, M. (2004). Condition based consensus solvability: A hierarchy of conditions and efficient protocols. Distributed Computing, 17:1–20.

[Mostefaoui et al., 2005] Mostefaoui, A., Raynal, M., Travers, C., Patterson, S., Agrawal, D., and Abbadi, A. E. (2005). From static distributed systems to dynamic systems. In Proceedings of the 24th IEEE Symposium on Reliable Distributed Systems, pages 109–118.

[Mui et al. 2001] L. Mui, M. Mohtashemi, C. Ang, P. Szolovits, and A. Halberstadt. Ratings in Distributed Systems: A Bayesian Approach. In Proceedings of the Workshop on Information Technologies and Systems (WITS), 2001.

[Muthitacharoen et al., 2001] Muthitacharoen, A., Chen, B., and Mazi`eres, D. (2001). A low-bandwidth network file system. In Proceedings of the 18th ACM Symposium on Operating Systems Principles, pages 174–187.

[Muthitacharoen et al., 2002] Muthitacharoen, A., Morris, R., Gil, T. M., and Chen, B. (2002). Ivy: a read/write peer-topeer file system. SIGOPS Oper. Syst. Rev., 36(SI):31–44.

[Naor 1995] M. Naor and A. Shamir, Visual Cryptography, Lecture Notes in Computer Science, Vol. 951, pp 1-12, Springer-Verlag, 1995.

[Napster,] Napster. Site internet napster : http://www.napster.com.

[Neff 2006] A. Neff, http://www.votehere.com

[Neves et al., 2004] Neves, N. F., Correia, M., and Veríssimo, P. (2004). Wormhole-aware Byzantine protocols. In 2nd Bertinoro Workshop on Future Directions in Distributed Computing: Survivability - Obstacles and Solutions.

[Neves et al., 2005] Neves, N. F., Correia, M., and Veríssimo, P. (2005). Solving vector consensus with a wormhole. IEEE Transactions on Parallel and Distributed Systems, 16(12):1120–1131.

[Oberender et al. 2005] J. Oberender, F. –U. Andersen, H. de Meer, I. Dedinski, T. Hoßfeld, C. Kappler, A. Mäder, and K. Tutschku, “Enabling Mobile Peer-to-Peer Networking. Mobile and Wireless Systems”, In Proceedings of Mobile and Wireless Systems, LNCS 3427, Dagstuhl, Germany, January 2005.

[Obreiter and Nimis 2003] P. Obreiter & J. Nimis, “A Taxonomy of Incentive Patterns - the Design Space of Incentives for Cooperation”, Technical Report, Universität Karlsruhe, Faculty of Informatics, 2003.

[Page et al. 1998] L. Page, S. Brin, R. Motwani, and T. Winograd, “The PageRank Citation Ranking: Bringing Order to the Web”, Technical report, Stanford Digital Library Technologies Project, 1998.

[Pandurangan et al. 2001] G. Pandurangan, P. Raghavan and E. Upfal. ”Building Low-Diameter P2P Networks”, IEEE Symposium on Foundations of Computer Science pag. 492-499, 2001.

[Papadopouli and Schulzrinne 2001] M. Papadopouli and H. Schulzrinne, “A Performance Analysis of 7DS, A Peer-to-Peer Data Dissemination and Prefetching Tool for Mobile Users”, In Advances in wired and wireless communications, IEEE Sarnoff Symposium Digest, March 2001.

[Peacock 2005] T. Peacock, P. Y. A. Ryan, Coercion-resistance as Opacity in Voting Systems, School of Computing Science Technical Report CS-TR: 959, Newcastle University, Apr 2006

[Pease et al., 1980] Pease, M., Shostak, R., and Lamport, L. (1980). Reaching agreement in the presence of faults. Journal of the ACM, 27(2):228–234.

[Pedone et al., 2002] Pedone, F., Schiper, A., Urb´an, P., and Cavin, D. (2002). Solving agreement problems with weak ordering oracles. In Proceedings of the Fourth European Dependable Computing Conference, pages 44–61.

[Peteron 1983] Peterson G.L., Concurrent Reading while Writing. ACM Transactions on Programming Languages and Systems, 5(1):46-55, 1983.

[Pizza et al. 1988] M. Pizza, L. Strigini, A. Bondavalli, and F. Di Giandomenico. Optimal discrimination between transient and permanent faults. In 3rd IEEE High Assurance System Engineering Symposium, pages 214-223, Bethesda, MD, USA, 1998.

[Powell et al. 1999] D. Powell, J. Arlat, L. Beus-Dukic, A. Bondavalli, P. Coppola, A. Fantechi, E. Jenn, C. Rabjac and A. Wellings, “GUARDS: A Generic Upgradable Architecture for Real-Time Dependable Systems,” IEEE Trans. on Parallel and Distributed Systems, 10(6), pp. 580–599, Jun. 1999.

[Powell, 1992] Powell, D. (1992). Fault assumptions and assumption coverage. In Proceedings of the 22nd IEEE International Symposium of Fault-Tolerant Computing.

[Preparata et al. 1967] F.P. Preparata, G. Metze and R.T. Chien, “On the Connection Assignment Problem of Diagnosable Systems,” IEEE Trans. on ElectronicComputers, 16(12), pp. 848-854, Dec. 1967.

[Punchscan 2006] Punchscan, http://punchscan.org/

[Quinlan and Dorward, 2002] Quinlan, S. and Dorward, S. (2002). Venti: a new approach to archival storage. In Proceedings of the First USENIX Conference on File and Storage Technologies, pages 89–101, Monterey,CA.

[Rabin, 1983] Rabin, M. O. (1983). Randomized Byzantine generals. In Proceedings of the 24th Annual IEEE Symposium on Foundations of Computer Science, pages 403–409.

[Randell 2005] B. Randell and P.Y.A. Ryan, Voting Technologies and Trust, School of Computing Science, Technical Report CS-TR: 911, Newcastle University, 2005.

[Randriamaro et al., 2006] Randriamaro, C., Soyez, O., Utard, G., and Wlazinski, F. (2006). Data distribution in a peer to peer storage system. Journal of Grid Computing (JoGC), Special issue on Global and Peer-to-Peer Computing, Springer,Lecture Notes in Computer Science.

[Raynal 2002] Raynal M., Wait-Free Objects for Real-Time Systems. Proc. 5th Int’l IEEE Symposium on Object- Oriented Real-Time Distributed Computing (ISORC’02), pp. 413-420,Washington DC, 2002.

[Raynal 2004] Raynal M., The Renaming Problem as an Introduction to Structures forWait-free Computing. Proc. 7th Int'l Conference on Parallel Computing Technologies (PaCT'03), Springer Verlag LNCS #2763, pp. 151-164, 2003.

[Raynal and Travers 2006] Raynal M. and Travers C., In search of the holy grail: Looking for the weakest failure detector for wait-free set agreement. Invited paper. Proc. 12th Int'l Conference on Principles of Distributed Systems, (OPODIS'06), To appear in Springer Verlag LNCS, 2006.

[Reiter, 1996] Reiter, M. K. (1996). A secure group membership protocol. IEEE Transactions on Software Engineering, 22(1):31–42.

[Rivest 2006] http://theory.csail.mit.edu/~rivest/Rivest-TheThreeBallotVotingSystem.pdf

[Romano et al. 2002] L. Romano, S. Chiaradonna, A. Bondavalli, and D. Cotroneo, "Implementation of Threshold-Based Diagnostic Mechanisms for COTS-Based Applications," Proc. 21st IEEE Symp. Reliable Distributed Systems (SRDS 2002), 2002.

[Rowstron and Druschel, 2001] Rowstron, A. and Druschel, P. (2001). Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility.

[Rubel, 2005] Rubel, M. (2005). Rsnapshot: a remote filesystem snapshot utility based on rsync.

[Ryan 2005] P. Y. A. Ryan and T. Peacock, Prêt à Voter: a Systems Perspective, School of Computing Science Technical Report CS-TR:929, Newcastle University, 2005

[Ryan and Schneider 2006] P.Y.A. Ryan and S. A. Schneider, Prêt à Voter with Re-encryption Mixes, School of Computing Science Technical Report CS-TR: 956, Newcastle University, 2006.

[Ryan, to appear] P. Y A Ryan, The Prêt à Voter Scheme for Voter-verifiable Elections, School of Computing Science Technical Report, Newcastle University, to appear.

[Schlosser and Kamvar 2002] M. Schlosser and S. Kamvar, "Simulating a file-sharing p2p network", In Proceedings of the First Workshop on Semantics in P2P and Grid Computing, 2002.

[Schneider 1993] Schneider F.B., What Good are Models and what Models are Good? Chapter 2, Distributed Systems (2nd edition), Addison-Wesley and ACM Press, pp. 15-26, 1993.

[Schneider, 1990] Schneider, F. B. (1990). Implementing faul-tolerant services using the state machine approach: A tutorial. ACM Computing Surveys, 22(4):299–319.

[Sengupta and Dahbura 1992] A. Sengupta, A. Dahbura, "On Self-Diagnosable Multiprocessor Systems: Diagnosis by the Comparison Approach," IEEE Trans. OnComputers, 41(11), pp. 1386–1396, Nov 1992.

[Shin and Ramanathan 1987] K. Shin, P. Ramanathan, "Diagnosis of Processors with Byzantine Faults in a Distributed Computing System," Proc. of FTCS-17, pp. 55–60, 1987.

[Shokrollahi, 2003] Shokrollahi, A. (2003). Raptor codes.

[Siewiorek and Swarz 1998] D. P. Siewiorek and R. S. Swarz, "Reliable Computer Systems – Design and Evaluation", A. K. Peters, Ltd.

[Sit et al., 2003] Sit, E., Cates, J., and Cox, R. (2003). A DHT-based backup system. Technical report, MIT Laboratory for Computer Science.

[Spainhower et al. 1992] L. Spainhower, J. Isenberg, R. Chillarege, and J. Berding, ªDesign for Fault-Tolerance in System ES/9000 Model 900,º Proc. 22nd IEEE FTCS - Int'l Symp. Fault-Tolerant Computing, pp. 38-47, 1992.

[Stinson 2005] D. Stinson, Cryptography Theory and Practice, Chapman & Hall, 2005.

[Terada et al. 2004] M. Terada, M. Iguchi, M. Hanadate, and K. Fujimura. An Optimistic Fair Exchange Protocol for Trading Electronic Rights. In 6th Smart Card Research and Advanced Application conference (CARDIS'2004), 2004.

[Ting and Deters 2003] N. S. Ting, R. Deters, "3LS - A Peer-to-Peer Network Simulator", IEEE International Conference on Peer-to-Peer Computing, 2003.

[Toueg, 1984] Toueg, S. (1984). Randomized Byzantine agreements. In Proceedings of the 3rd ACM Symposium on Principles of Distributed Computing, pages 163–178.

[Turpin and Coan, 1984] Turpin, R. and Coan, B. A. (1984). Extending binary Byzantine agreement to multivalued Byzantine agreement. Information Processing Letters, 18(2):73–76.

[Veríssimo et al., 2003] Veríssimo, P., Neves, N. F., and Correia, M. (2003). Intrusion-tolerant architectures: Concepts and design. In Lemos, R., Gacek, C., and Romanovsky, A., editors, Architecting Dependable Systems, volume 2677 of Lecture Notes in Computer Science, pages 3–36. Springer-Verlag.

[Veríssimo, 2003] Veríssimo, P. (2003). Uncertainty and predictability: Can they be reconciled? In Future Directions in Distributed Computing, volume 2584 of Lecture Notes in Computer Science, pages 108–113. Springer-Verlag.

[Veríssimo, 2006] Veríssimo, P. (2006). Travelling through wormholes: A new look at distributed systems models. SIGACT News, 37(1):66–81.

[Vogt et al. 2001] H. Vogt, H. Pagnia, and F. C. Gärtner. Using Smart cards for Fair-Exchange. WELCOM 2001, LNCS 2232, Springer, pp. 101-113, 2001. http://citeseer.ist.psu.edu/vogt01using.html

[Voulgaris et al. 2005] S. Voulgaris, D. Gavidia and M. van Steen, "CYCLON: Inexpensive Membership Management for Unstructured P2P Overlays", Journal of Network and Systems Management, Vol. 13, No. 2, June 2005

[Walter et al. 1994] C. Walter, M.M. Hugue and N. Suri, "Continual On-line Diagnosis of Hybrid Faults," Proc. of DCCA-4, pp. 150-166, 1994.

[Walter et al. 1997] C. Walter, P. Lincoln, N. Suri, "Formally Verified On-line Diagnosis," IEEE Trans. On Software Engineering, 23(11), pp. 684–721, Nov. 1997.

[Weatherspoon and Kubiatowicz, 2002] Weatherspoon, H. and Kubiatowicz, J. (2002). Erasure coding vs. replication: A quantitative comparison. In Proceedings of the First International Workshop on Peer-to-Peer Systems.

[You et al., 2005] You, L. L., Pollack, K. T., , and Long, D. D. E. (2005). Deep Store: an archival storage system architecture. In Proceedings of the 21st International Conference on Data Engineering (ICDE 2005), pages 804–815, Tokyo, Japan. IEEE.

[Zhong et al. 2003] S. Zhong, J. Chen, Y. R. Yang, "Sprite: A Simple, Cheat-Proof, Credit-Based System for Mobile Ad-Hoc Networks", INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies.

[Zielinski, 2004] Zielinski, P. (2004). Paxos at war. Technical Report UCAM-CL-TR-593, University of Cambridge Computer Laboratory, Cambridge, UK.

Part Socio – Resilient Socio-Technical Systems

Co-ordinator: Michael Harrison

Contributors: Eugenio Alberdi[1], Sandra Basnyat[4], Roberto Bonato[2], Remi Bastide[4], Giorgio Faconti[5], Jeremie Guiochet[6], Michael Harrison[7], Philippe Palanque[4], Alberto Pasquini[2], Birgit Pfitzmann[3], Lorenzo Strigini[1] , Mark Sujan[7], Marco Winckler[4]

[1]City University, London; [2]Deep Blue; [3]IBM Zurich, [4]IRIT University Toulouse; [5]ISTI Pisa; [6]LAAS Toulouse; [7]University of Newcastle

Definition of a socio-technical system

A socio-technical system is a system comprising humans interacting with technology. The term is used in the context of an approach to complex work design and assessment that accounts for the interaction between people and technology: the design of the technical component of a system is informed by human/social related characteristics that contribute to identify choices and that influence the technical development. Devices are simply the technical components of the socio-technical system, for example, the flight management systems or the medical advisory systems that are conventionally the subject of software or hardware development. These devices must be designed to operate resiliently within the broader socio-technical system and these systems typically consist of many and varied such devices. Within Part Socio, the term "system" will be used to refer to socio-technical system.

Introduction

This section identifies general topics relevant to the modelling and analysis of resilience in systems. It focuses specifically on topics that have been the concern of the ResIST network of excellence. We bring these topics together here to emphasise some of the distinctive human and organisational issues that are at stake here. The document discusses different phases of a putative process for the analysis, design and implementation of systems. Inevitably there are omissions and biases in its content – our aim is to provide a context for understanding the knowledge about the variety of techniques that are available within the network of excellence rather than providing a general framework. Topics from human factors or ergonomics such as display design and technical concerns such as the effectiveness of different kinds of pointing device in different task contexts will be omitted from this discussion. The importance of these activities is recognised, however focus is given to topics that are based on the expertise of ResIST and in some sense go beyond these topics. Section 2 briefly discusses a number of techniques that have been and are being developed for the study of systems, providing material that enables designers and implementers to establish the requirements for a new system. It focuses on the techniques for representing the tasks, devices and systems to make them tractable to analysis. Section 3 discusses systematic, structured but in some cases informal techniques for the analysis of aspects of systems to enable the formation of more resilient designs. This section then summarises quantitative techniques for the analysis of an installed system and for assessment of the ongoing evolution of the design of the socio-technical system.

1 – Understanding the structure and organisation of socio-technical systems: representation and modelling

1.1. Elicitation and observation

Many architectural and organisational issues have an impact on the resilience of systems. This section focuses on methods of elicitation techniques used to comprehend systems based on observation, techniques that can be used to gain an understanding of the system prior to modelling it. An essential element during the design, analysis and evaluation of systems is a clear understanding of the roles and tasks of the humans who are embedded within it (users, operators, maintainers, etc.). Many techniques have been proposed to understand these roles and tasks through the observation and analysis of humans at work [Jonassen et al. 1999]. Observation may be difficult when designing new systems. However, the most common case when such techniques are used is when design is for up-grading and enhancing of an existing system. In this case analysis will regard the tasks performed within the existing system. When the system must be designed from scratch, this is done usually to automate the manual activity of an existing process. In this case the observation is conducted on the "non-automated" tasks that still represent a valuable source of information. A variety of techniques are available including those inspired by ethnographic techniques, see for example "Contextual Inquiry" [Beyer and Holtzblatt 1998] as well as the use of techniques such as activity theory and other holistic approaches [Pasquini et al. 2000]. Many of these techniques involve the analyst working alongside or acting as an apprentice to someone involved in the work being observed, others involve techniques using video data or interviewing people involved in their activities.

Once an understanding of the system has been gained it becomes possible to represent elements of the system for the purposes of implementation and analysis.

Task analysis techniques also represent a systematic method for this observational and analysis work, enabling a rigorous, structured characterization of user activity. Hierarchical Task Analysis (HTA) is the most popular and flexible of the task analysis techniques [Diaper and Stanton 2003]. Here work is represented in terms of the goals, sub-goals, actions and plans that are observed of the activity. These task components are then graphically represented using a structure chart. HTA entails identifying tasks, categorizing them, identifying the subtasks, and checking the overall accuracy of the model. [Vicente 1999] contains an excellent critique of task analysis techniques. He proposes as a complement to task descriptions a variety of techniques (Cognitive Work Analysis) for identifying the structure of the organisation in terms of two hierarchies (a means-ends hierarchy and an object orientated hierarchy). These describe and analyse what is to be achieved and the means by which it is achieved. He also proposes the identification of strategies adopted by users and typical scenarios of use, and a less constrained and prescriptive approach to

capturing the work that people do – instead of sequences of actions, the user's activity is understood in terms of a number of constraints that govern the way that the work is performed. Less complicated and more practical techniques used in Human Computer Interaction include scenario analysis (see, for example, [Rosson and Carroll 2002]). This technique captures activity in context as stories or narratives as gathered from existing users of the device. Here narrative descriptions of typical or extreme system behaviours are employed as a basis for analysis.

Cognitive techniques aim to overcome the overly restricted scope of HTA especially when tasks are more intricate, knowledge-intensive, and subject to increasingly integrated forms of technological support, as in modern, complex, socio technical systems [Chipman and Shalin 2000]. Cognitive techniques demand a strong interaction with experts of the domain investigated, to understand the knowledge embedded in the tasks that are carried out. They are supported by a range of techniques such as structured interviews [Hoffman et al. 1995], naturalistic observation [Gordon and Gill 1997], and ethnography [Roth and Patterson 2000]. The scope of cognitive techniques is to define a coherent knowledge representation for the domain being studied, expressing it in terms, for example, of a semantic network or a goal/method graph [Gordon and Gill 1997]. Activity theory takes a more holistic view of the human work and models people as agents, rather than as collections of cognitive attributes [Bannon 1991]. In contrast to traditional task analysis it uses a higher level of analysis, considering "activity," rather than tasks.

From a different perspective, explorations of rules and procedures and their effect in organisations is a broad topic of study. This is represented within the network by IBM Research [Giblin et al. 2005]. Recent years have seen a number of high-profile incidents of corporate accounting fraud, security violations, terrorist acts, and disruptions of major financial markets. This has led to a proliferation of new regulations that directly impact businesses. As a result, businesses, in particular publicly traded companies, face the daunting task of complying with an increasing number of intricate and constantly evolving regulations. Together with the growing complexity of today's enterprises this requires a holistic compliance management approach with the goal of continually increasing automation. IBM Research has introduced REALM (Regulations Expressed as Logical Models), a meta-model and method for modelling regulations and managing them in a systematic lifecycle within an enterprise [Giblin et al. 2005]. Regulatory requirements are formalized as sets of compliance rules in a novel real-time temporal object logic over concept models in UML, together with metadata for traceability. REALM provides the basis for subsequent model transformations, deployment, and continuous monitoring and enforcement of compliance in real business processes and IT systems.

Formal notations and processes for software representation have been employed by members of ResIST to capture some elements of systems and in some cases this has involved some extension to these techniques.

1.2. Modelling the task

ISTI have developed a notation, CTT (ConcurTaskTrees) [Paterno 1999], that has been used extensively by a number of groups including IRIT. This notation, based on LOTOS (Language of Temporal Ordering Specification), permits the representation of concurrent tasks and identification of agency with action. [Navarre et al. 2001] have integrated CTT with their ICO (Interactive Cooperative Objects) formalism. These techniques are useful for decomposing complex tasks, but have a narrow view of task. These representations tend to focus on what people do, rather than why they do it, and are only useful when there is a clear, well understood goal for the human activity.

1.3. Modelling the device

The purpose here has been to produce models that have value from a usability point of view as elaborated in the agendas established in [Harrison et al. 1989], [Harrison et al. 1993] and to check whether these device models respect properties that ensure usability of the device. All of the approaches therefore make use of existing formal techniques. The innovation in the context of socio-technical systems is the way in which the specification or models are structured, the concepts that are modelled and the properties that are articulated using the notational frameworks. The initial perspective was that, with the help of psychologists, it would be possible to explore the role that these properties played as general characteristics of interactive systems. In practice an understanding of context is an important prerequisite to identifying the particular characteristics and purpose of the system in which the device is embedded. Both ISTI and Newcastle nodes have been concerned with the specification of building "bricks" in devices (referred to as *interactors).* The aim of interactors is to structure specifications to emphasise those aspects of the device of value from the point of view of the whole system – how the device is used within the system. This work was initially done by Faconti and Paterno using LOTOS. Duke and Harrison explored a variety of other specification techniques (Z, MAL and VDM). The relationship between these different techniques is described in [Duke et al. 1994]. An alternative strand of research with similar goals has been the development of interactor like structures for statechart based specifications [Degani 1996]. Alternative approaches to modelling devices have included work within ISTI [Faconti and Massink 1998] in particular using LOTOS [Faconti and Massink 1997] and Petri nets linked to objects [Bastide and Palanque 1990]. Further exploration of the scope of modelling techniques has been carried out using a hybrid notation of Petri nets and a simple notation for describing continuous flow to represent the hybrid aspects of interaction techniques within virtual environments [Willans and Harrison 2001]. Other work has involved the modelling of time, beginning with [Bowman et al. 1998] and concerned with dynamic function scheduling in [Loer et al. 2004]. The latter activity has been concerned with modelling the way that a system is automated deciding the design of the devices that are to be used within the system. IRIT in particular has been concerned with resilience-explicit aspects of both system and devices through explicit modelling of barriers against undesired events, to aid the safety assessment of systems **[Basnyat et al. Submitted]; [Schupp et al. 2006]; [Basnyat and Palanque 2006]** (see also section 3 on safety assessment).

Recent research has been concerned with supporting the requirements of a particular branch of the airspace industry by adopting Statecharts. In this work a particular focus has been the development of a tool that eases the formulation of usability requirements in Computational Tree Logic (CTL) [Loer and Harrison, 2006] in the context of the development and analysis of the moding of a mobile device for process control as well as specifications of interface standards for cockpit devices **[Barboni et al. 2006a],** [Campos and Harrison 2001].

Application Domains	Requirements	Tasks Modeling	Devices Modeling	User Modeling	Location/context Modeling
Air Traffic Control	X	X			
Military Cockpits		X			
Civil Cockpits			X		
Cockpits DCSC	X				X
Health/hospital					
Built Environment			X	X	X
Mining Case Study		X			
Satellite Control Rooms		X	X		

Development Phases

Table 1. Knowledge gathered by Working Group Socio members related to the application domain and development phases

Table 1 illustrates many of the phases of the development process (not evaluation and assessment) and some of the domains of application that have been addressed by ResIST participants. A number of domains have been covered at some level of detail. The table indicates that the work represented has been less concerned with the whole process of development, identifying requirements and modelling system elements including tasks and context that capture requirements before modelling the process than describing how different stages of the process can be dealt with using different approaches and notations. Two projects involving members of ResIST have explored reference frameworks designed to make it easier for implementers of interactive systems to understand the properties and characteristics of their systems and thus deal with the whole development process. The TACIT network produced a taxonomy concerned with identifying the special properties of "continuous interaction" [Massink and Faconti 2002]. This taxonomy was concerned with interaction that involves both discrete and continuous elements using mouse clicks and keystrokes but also using haptic properties of the interaction and using characteristics and shapes of gestures. Systems have also been modelled as layers, understanding interaction at a number of levels of detail, much like the protocol layers familiar in other branches of computer science [Barnard et al. 2000]. The aim of structuring the interaction in several layers of abstraction is to reduce analysis complexity.

Since the requirement for experiment and exploration of the device in use is so important ResIST members have inevitably been concerned with prototyping device models. The integration of formal device modelling (using ICOs) and implementation and rapid prototyping has been explored through a number of concrete examples, for example [Winckler and Palanque 2003] in relation to web applications and [Bastide et al. 2002]. in relation to air traffic control en-route workstations. Such connection between prototyping and modelling provides a way of addressing, within the same framework, both usability and reliability concerns that are key design issues of safety critical socio technical systems [Navarre et al. 2002]. Similar techniques have been used to model and prototype interaction techniques combining continuous and discrete components [Willans and Harrison 2001].

1.4. Modelling the user (syndetic modelling)

The term 'syndetic modelling' denotes an approach to reasoning about interaction developed within the interdisciplinary project AMODEUS [Duke et al. 1998]. A cognitive model, using cognitive theory, representing features of the user is combined with a device model. The models are described using a common (formal) representation to allow description of and reasoning about the interaction between human and computer agents. Syndetic modelling reconciles two different views by developing two complementary models using the same specification technique, using a notation that is familiar to formal software engineers. In [Duke et al. 1998] the cognitive theory is based on an approach called Integrated Cognitive Subsystems (ICS). The device model focuses on user interactive properties. This representational framework can accommodate both the information carried and presented by the system, and that perceived and understood by the user. It makes the simplifying assumption that a system can be characterised simply as a dyad: a device and a user.

The model is as correct as the theory that is captured by the user specification. Interaction techniques can be exported across environments as long as one stays with the same theory. This is shown for example in [Duke 1995], [Faconti and Duke 1996], [Faconti 1996], and [Bowman and Faconti 1996].

In [Palanque and Bastide 1995] formal description techniques are used to check data and behavioural compatibility between task models and device models. [Palanque et al. 1997] relate requirements models expressed in temporal logics to tasks and device models.

1.5. Open issues

There are a number of open issues relating to resilience in socio-technical systems. These issues have an impact on how we evaluate these systems and the kinds of models that are appropriate to their analysis and implementation.

1. How do we identify, through observation, the “resilience” behaviour of the system to reflect these characteristics through appropriate models. While there are many useful references to types of errors, see for example individual errors [Reason 1990] and also organisational errors [Reason 1997], the propensities towards these behaviours is more difficult to assess. This may be a process of understanding the “disturbances” or “mismatches” that typically occur within human processes but are usually recovered from.
2. How do we understand the redundancy and diversity mechanisms that make systems “high reliability organisations”, where is there potential for propagation of failure throughout the organisation and how can these failures be recovered from and barriers used to prevent these failures from propagating? There is an extensive literature about the advantages and limits of redundancy and diversity in social and socio-technical systems. For example, much attention has been given by social scientists to “High Reliability Organizations”, with descriptions that often emphasise the usefulness of checks and balances, the availability of people to take on tasks when required to correct for failures, etc. However, this mostly descriptive literature falls short of answering the important questions, such as how to decide before operation whether a certain form of added redundancy will actually be effective or cost-effective in improving the overall resilience and dependability of the socio-technical system. A more extensive critique with a short review of such literature is in [Marais et al. 2004]. [Sagan 2004] lists reasons why adding redundancy to an organisation may not improve its resilience, which he summarises in a non-exhaustive list as “common-mode

failures", "social shirking" (the fact that knowledge of sharing a responsibility with others makes people less likely to take action to fulfil it) and "overcompensation" (the fact that if people attempt to exploit the presence of risk reduction mechanisms to improve performance without decreasing risk, they may make mistakes and actually increase risk compared to what they accepted without the risk reduction mechanisms). But this paper, too, is a valuable invitation to caution against trusting intuition when dealing with subtle design issues, not a technical contribution to methods for analysing these issues. Evaluation of diversity in general is discussed in Part Eval of this document. It is worth recalling here some of the special difficulties that affect it (and thus the rational design of redundancy and diversity) in socio-technical systems. These are linked to the common problem of people exhibiting more variation of behaviour (between different people, and different instances of behaviour of the same person) than many technical components, and to the specific fact that the characteristics of their behaviour are affected by their perception of the system of which they are parts.

3. How do we understand the difference between work as envisaged and work as practised. [Dekker 2006] highlights the difference (which may be substantial) between operations as management imagines them and how these operations are actually carried out. Often leadership may not be tuned in to the challenges and risks encountered in real operations. Supervisors may be convinced that safety and success result from workers following procedures. But workers may encounter problems for which the right tools are not at hand and face severe time constraints. In such cases workers need to adapt, invent compromise and improvise to get the job done in spite of resource limitations, organizational dilemmas, and pressures. From the outside such informal work systems can be characterised as routine violations of procedures (perceived as 'unsafe'); but, from the inside such violations are a mark of expertise. Informal work systems thrive because procedures are inadequate to cope with local challenges and surprises. However resilience thus built, invisible to management, is vulnerable to unusual events that go beyond the knowledge and experience of the workers, and make any organisational changes more difficult to accomplish and failure prone.

4. How do we monitor organisations in order to understand their ongoing performance? [Wreathall 2006] notes that organisations constantly struggle to understand how they are performing with regard to safety and try to measure their safety performance. Some of the most common approaches involve recording and measuring trends associated with safety outcomes and using some kinds of performance models and data-based evaluations of safety performance. But these are very limited sources of operational management information. On the one hand, historic data will always be out of date as a measure of *today's* performance since data are from relatively rare events and almost always aggregated over long periods of time. On the other hand the current practice of Probabilistic Safety Assessment and other current modelling techniques are static interpretations of how accidents occur. They neglect the complexity and interactions seen in complex accidents.

What are required are data that allow the organization's management to know the current 'state of play' of the safety performance within the organization. Work has started to develop indicators of organisational performance that can be of value in this respect. This approach looks for data both at the working level and in organizational behaviours that can have set them up to have vulnerabilities. This requires multidisciplinary efforts to integrate activities such as:

- data analysis related to safety culture and climate and understanding how they relate to performance
- observations of how work is carried out in the real world
- the timing and extent of resources that are necessary for "harm absorption"
- how work processes and human behaviour act to make safety better through individual and small team activities

- improved understanding of decision making when it relates to sacrificing production goals to safety goals

5. How do we understand the impact of a system's evolution? There is little relevant research that concerns the analysis and prediction of behaviour of socio-technical systems in the face of system evolution. This issue is discussed by **[Sujan et al. 2006a]** in the context of healthcare systems. Here two examples are given, one concerns the introduction of new technology, the other an organisational change. The ability to predict the resilience of a system in the face of system evolution is an open issue that we see as a gap for further exploration.

2 – Evaluation and verification issues in resilience in socio-technical systems

Introduction

Evaluating the resilience, dependability or quality of service of a system, including a socio-technical system, implies:

- taking a whole-system viewpoint, i.e., evaluating not how well individual components or algorithms inside the system work, but how well the whole system works as an effect of their co-operation, interference, redundancy, etc. (this forms the basis for the discussion of this section).
- giving a quantitative estimation: not necessarily precise numbers ("the expected reduction in the number of lives saved by this medical tool because of failures is 2.5 per year per 1000 patients"), but intervals or orderings ("introducing this medical tool can be trusted with high confidence to reduce the number of deaths in the category of patients affected").

This section focuses on systematic analyses, considering how a number of types of models can be used as a basis for exploring a system ranging from descriptive models using a variety of levels of formality and rigour to quantitative and probabilistic models. It discusses a variety of topics in the evaluation and analysis of systems. The techniques described vary between the informal and structured approaches, function allocation and human error identification and more systematic and mathematically based analyses of models of interactive systems. Section 2.2 discusses function allocation techniques in the context of automation. We discuss the problem of automation and methods that support decision about how a system should be automated. In section 2.3 we give consideration to issues of system evaluation. *Usability evaluation* has evolved in response to awareness of deficiencies in human computer interaction, i.e., of the possibility of substantially reducing operator errors by more careful design of the user interface of automated components. Additionally, improving performance and users' comfort are also targets of this kind of evaluation. Consequently usability evaluation has typically focused on identifying problems and possibilities of improvement; it is thus a useful part of the modelling towards quantitative evaluation. A limitation from the ResIST viewpoint has been its focus on the device. The device model must be complemented with models of aspects of the system, that enable an analysis or assessment of the effects of the user or the physical context. *Safety assessment* (section 2.4) has been a required engineering activity in many industrial sectors for a long time. Therefore, several different, well-established forms of process exist to accomplish it in different industrial cultures, which deal in different ways (often unsatisfactory) with the difficulties presented by the complexity of the task and specifically by the human factors involved. Section 2.5 focuses on issues with the

formal verification of interactive systems, checking that device and system models of the proposed design incorporate requirements relating to its use.

A further subsection (2.6) deals with issues in the quantitative evaluation of socio-technical systems. The simplest case of evaluation is retrospective, how a system *has behaved*, e.g., counting service failures, or measuring their cost, at the end of a period of operation of the socio-technical system. Such evaluation may for instance resolve litigation as to how good a service a company has provided. But most often, useful evaluation work is a matter of predicting the future, or hypothetical futures, to support decision making by answering questions like:

- is this system good enough to deploy?
- if I deploy this system, what amount of loss, of preventive maintenance, of ... do I need to plan for?
- of these two systems that I could deploy, which one will be better?

As for other systems, these prediction tasks generally require knowledge integration and analyses of different kinds. Two structures of argument are common in reliability engineering in general:

- clear-box models combining descriptions of how the system components may fail, how each component's failure would affect the rest of the system, and what kind of redundancy exists to tolerate them, together with estimates of the probabilities of the various combinations of such events;
- black-box models that consider retrospective evaluation of a system together with an assessment of the extent to which its future behaviour may differ from the past, because of changes in either the system or its environment (a special category of these is so called "reliability growth modelling": trend extrapolation). This latter form, even if not used at whole-system level, is usually the basis for the component-level evaluations needed as inputs to the clear-box approach.

It is commonly acknowledged that both forms of evaluation present special difficulties for socio-technical systems. The current ability to evaluate such systems is widely seen as unsatisfactory. It is difficult to turn available evidence about a system into a cogent argument about foreseeable levels for measures of its resilience, dependability or quality of service. In part, these deficiencies call for better ability to manage complex "cases", handling properly all the evidence pertinent to the problem (compare Part Eval of this deliverable); in part, for better techniques, or more complete empirical knowledge to fit into them. The goal is to describe the uncertainty about the future of a system in terms that are as clear and correct as feasible, although this uncertainty cannot be eliminated and may well remain substantial.

Finally this section briefly discusses various general issues and difficulties with system evaluation.

2.1. Automation and function allocation

Function Allocation is driven by several, partly conflicting, motivations. Automation promises to extend or support human performance, to compensate for human performance deficits, to relieve the human of routine tasks, or to replace the human altogether. At the same time, it transforms the role of the human from a hands-on controller to that of a strategic decision-maker and supervisory controller of an ever-increasing multitude of functions. The operator may have to switch between functions, react to unforeseen events or demands, and compensate for automation failures. To fulfil this role, the human needs to be kept informed of, and involved in, the operation of the system. Consistent and transparent automation behaviour can help to maintain the compatibility of the operation strategies of the human and the automation, whereas mismatches between these strategies can lead to failure. The challenge for automation designers lies in providing automation that

can act rapidly and dependably under hard deadlines, while allowing the human to control, configure or intervene in the operation of the system.

Function allocation is about trade-offs between technology, the human performance of tasks in a system and the assignment of functions to one or more generic system elements [Pressman 1992]. It takes place early during the design phase of a system while the different design options are being synthesised and evaluated. In many system engineering methods the allocation of function is glossed over, and when it is mentioned it is primarily concerned with the allocation between software and hardware and not between humans and machines. As a result of this, allocation decisions tend to be based solely upon engineering criteria (such as 'Is this feasible?' or 'How much will it cost?') and not on human factors issues (such as 'What will be the impact upon the operator's ability to maintain situational awareness if this function is automated?').

Inappropriate use of automation and inappropriate division of functionality between humans and machines can result in undesirable system characteristics. It is now widely recognised that simply automating everything possible results in systems that are difficult for their operators to use. Operators can find themselves asking questions such as 'What is it doing now?' and 'Why is it doing that?' Inappropriate automation can also lead to degradation of operators' skills due to their lack of use, and it can even decrease the ability of the operators to respond to an unexpected event. This happens because inappropriate automation prevents the operators from actively participating in the running of the system, and therefore they have little awareness of the current system state when the unexpected event occurs. Many accidents have been documented where inappropriate use of automation has been a primary contributory factor. Even though the allocation of functions is not an explicit step in most system engineering methods, there has to be a point during the development of every system where the decision about the allocation of functions and the various consequential trade-offs are made. The assumption most system engineering methods seem to make is that the designers simply know what tasks are best performed by the machine and what are best performed by the operator, but experience shows that for complex interactive systems this assumption is not a valid one. The rapid advance in computer technology makes it possible for designers to provide levels of automation that totally change the relationship between the operator and the machine, and they are under pressure to ensure that they take full advantage of the benefits that this can bring. The development of a complex interactive system requires the designer to carefully consider the possible implications of the proposed automation. One approach to doing this is to use a function allocation method, designed to aid in the production of systems with an appropriate use of automation.

Function allocation methods give the designer guidance, based upon a chosen set of decision criteria, as to what level of automation should be used. The traditional question of function allocation is "To which agent, human or machine, should we allocate this function?". The early work of Fitts and allocation approaches using MABA-MABA (Men are better at … Machines are better at …) lists are typically restricted to considerations of performance [Fitts 1951]. It became apparent that many functions in a complex system require the sharing of the function between the human and machine. Such sharing may take place in many different ways. Billings identifies 5 different levels of automation between totally manually operated by the human and fully autonomous operation by the machine [Billings 1996]. Sheridan & Verplanck offer a set of 8 intermediate levels that can occur in supervisory control systems [Sheridan and Verplanck 1978]. The IDA-S framework offers a potential 43,046,721 levels of automation [Dearden et al. 2000]. Figure 1 presents the flow of the method behind such a framework. The process starts by listing system functions and then exploits usage scenarios as a structuring means for identifying potential candidates to automation. Cost-benefit trade-offs are used to assess the potential impact of such automation. One important aspect is the identification of emergent tasks/scenarios that emerge rather than being designed as a result of the systems

modification. Automation trade-offs can be understood by exploring the role and the user by investigating user activity described in terms of scenarios that have been understood as activation of user tasks. System allocation decisions may themselves influence user tasks. The method therefore concludes by requiring a more global (at least over all the scenarios) assessment of the impact of the given function allocation decisions.

There has been a steady increase in the number of criteria that may be applied in making function allocation decisions, see [Johnson et al. 2001] for a review of current methods, and [Harrison et al. 2003] for considerations about how to integrate function allocation into systems / software engineering techniques in the context of a ship based navigation system. The process involves a decision procedure in which trade-offs involve notions such as workload, situation awareness and human performance against cost of implementing the technology required for the automation and how it should match to roles designed for the human operators. Although this is not an explicit concern of these papers, it would make sense to consider further issues such as issues of redundancy and diversity of alternative levels of automation.

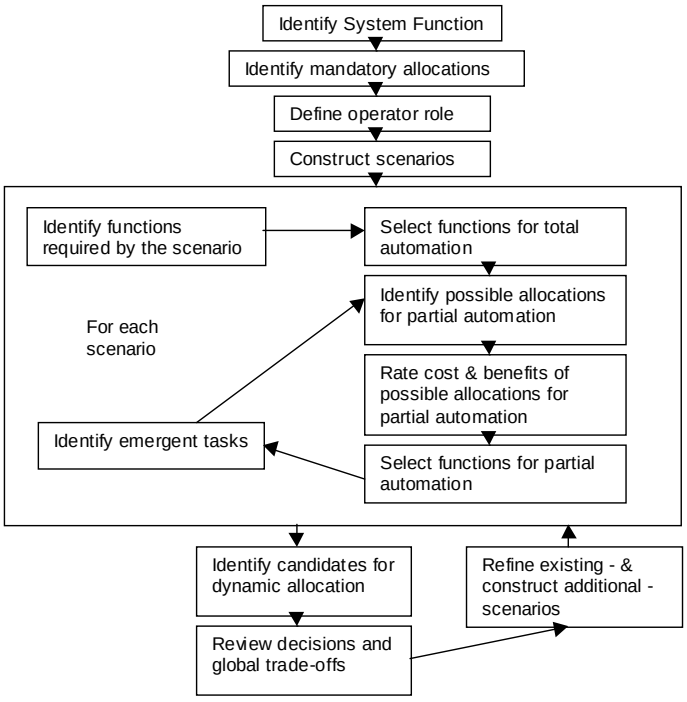

Figure 1: Structure of the [Dearden et al. 2000] method.

Dynamic Function Allocation (DFA) addresses the problems of static allocation by providing multiple levels of automation, and decision rules (potentially also automated) to switch between them at runtime. This creates a workload balancing mechanism that makes the system more adaptive to a wide range of operational parameters, as the agents – human(s) and automation – can supply mutual back-up in case of performance degradation or changing demand characteristics. Changes in automation levels may occur as part of a planned operation strategy (e.g., engage autopilot during cruise phase, disengage during take-off / landing),

but most empirical studies in this area have been concerned with allocation switches as a reaction to unforeseen events or workload changes, e.g., [Endsley and Kaber 1999]. It is important to notice that these allocation decisions are dynamic in time, but in a form that is best described as "snapshot allocation". At any instant, the algorithm that chooses what automation is required (relying, for instance, on triggers from critical-event monitoring, performance or physiological measurement) assesses the need for re-distribution of functions and suggests or implements the required changes by moving up or down the automation scale. Dynamic allocation for example recognises human operator workload to require the human operator to do more or less work with the existing function. This relying on snapshot information helps to keep the allocation decision computationally simple, but it also disconnects automation design considerations from another important class of workload management strategies: for example the use of scheduling of activities. These issues are discussed with examples in [Hildebrandt and Harrison 2002a]; [Hildebrandt and Harrison 2002b];[Hildebrandt and Harrison 2003];[Hildebrandt and Harrison 2004]. As the dynamic decision algorithm implied in DFA becomes more complex it becomes more important that it also should be considered to be a further candidate for automation. A careful analysis of the role of the operator is required to support the most effective automation of the algorithm.

An important issue in the consideration of system automation is how the automation – the shifting of functions from the user to the automation – may leads to unanticipated biases, i.e., systematic deviation of the user's behaviour from the desired behaviour. This topic is discussed in more detail in the review **[Alberdi et al. 2006].**

2.2. Considering the user and usability evaluation

Usability problems might appear in different forms in the user interface, reducing the user performance with the device, increasing the number of errors or having users being reluctant to use the device due to uncomfortable/unpleasant interaction. In the last decades, the Human-Computer Interaction (HCI) community has developed several methods to support sound and rigorous identification of usability problems. Many approaches have been proposed to cope with the diversity of contexts in which evaluation has to take place. Such methods may involve observation of users' activity, inspection by a usability specialist, simulation and/or prediction of usability problems based on models describing users' expected activity like, for instance, task models. One class of such methods, those that are model-based provide better support for the design of interactive systems, for example by identifying usability problems in the early phases of the development process thus reducing the time and development costs.

Quite often, usability evaluation methods must be adapted also to the idiosyncrasies of the system and the device such as multimodal[1] user interfaces **[Palanque et al. 2006]**, Web applications [Scapin et al. 2000], [Winckler et al. 2004], information visualization techniques [Winckler et al. 2004], and so on.

In the context of ResIST, usability evaluation methods can be used to assess the resilience of socio-technical systems. Indeed, generic expertise within the group of partners that can be fruitfully used includes:

- measurement of user performance while interacting with the system/components/devices,
- measurement of user cognitive load during task execution (which can have an impact on the occurrence of human errors),
- gathering information as well as predicting user behaviour when confronted with a critical situation (e.g., system error or malfunction)

[1] Multimodal interfaces require a combination of modalities, for example voice, gesture and keyboard interaction.

- gathering information about the subjective perception of systems (for example, trust, comfort, …).

Usability evaluation methods can be used for assessing many aspects of systems such as the user interaction with devices (e.g., user performance with devices or combinations of devices, for example assessing the use of modalities in such devices), with interaction components (for example pie menus, gesture interaction, standard menus, and so on), and with the device itself (e.g., complex dialogue techniques used in 3D environments).

The research conducted by IRIT is mainly focused on the development of model-based usability evaluation methods with a particular emphasis on devices and has been applied to different application domains. A recent study **[Palanque et al. 2006]** presented the use of a formal description technique for describing multimodal interactive applications using the ICO formal description technique for a "space ground segment system" for satellite control. While these model-based techniques provide several benefits related to the software engineering of such systems, they can also support the usability evaluation activities that are usually considered external to the development process. This specific contribution provides a way of integrating, within a single development framework, competing requirements like reliability and usability.

Connecting knowledge and practices in the field of human-computer interaction to the ones available in software engineering is critical if usability and reliability aspects of the interactive systems design both have to be addressed. Previous work [Palanque et al. 1999] proposed a way of integrating ergonomic rules with a model-based development process. Other work, such as [Palanque and Paterno 2001], proposed a tool-supported framework for integrating task models and device software models.

Another thread of work at IRIT includes methods that can be applied in the early phases of the development process, thus reducing the costs of problems correction after the application has been implemented **[Xiong et al. 2006].**

2.3. Safety assessment

In terms of a systematic approach to integrating human factors into system hazard analysis, there are a number of techniques that can be applied (e.g., [Hollnagel 1998] [Kirwan 1994]). In doing so it is important to address a number of dimensions which impact on usability, task performance and the behaviour of complex heterogeneous systems. To assess the system safety it is first necessary to perform a background analysis using the processes discussed in section 1, namely:

- **Work and interface design**—including user interfaces, procedural aspects, working materials. One important aspect is whether the user's mental model is supported by the device interface or whether there are important divergences. Under stressful or unusual situations users will act according to how the device might be expected to behave.

- **Social aspects** – including collaboration and communication models for operators and supervisors. There have been cases, for example the redesign of control rooms, where the new system has failed to address the social communication aspects of how the day-to-day work was done. The way people communicate and act in a control room is often complex and subtle.

- **Organisational context** – including an assessment of the communication channels within an organisation, safety culture, degree of interconnectivity between the different processes and procedures, focus of safety management and, how tasks are delegated, initiated, interrupted and terminated.

- **Impact of change** – there is typically a gap between what is documented as official process and actual process followed. The impact of change should address the actual work environment in preference to the documented process.

Safety or dependability assessment is a process that involves the following steps:

- Hazard identification: in which the immediate causes of failure are identified through a process involving a team systematically exploring the different elements of the domain.

- Cause-consequence analysis: in which a systematic analysis of the design is performed either using a representation of the architecture of the system or a representation of the intended processes that the system is expected to perform. Typically this process involves an "imaginative anticipation" of possible errors associated with the design. Often the purpose of this design is to check that all potential vulnerabilities in the design are "mitigated" against. This process involves providing an argument why the system will prevent the specified hazard from occurring. In some cases, this is the essence of one leg of a dependability argument. Cause consequence analysis may also be seen as part of the design process. In this mode the method is iterative and should be performed as early as possible in order that iterations and "barrier" designs may be produced at minimum cost to reduce the system's vulnerability to failure.

- As a result of the hazard analysis events or faults may be identified and through analysing these events and faults vulnerabilities in the design of the system can be rectified. This identification may be achieved either by performing a cause consequence analysis as described above or by identifying scenarios that are particularly problematic because of the control mode or circumstances in which the scenario occurs. Once these events have been identified probabilistic safety assessment may be performed on these events or faults by considering the set of possible (logical, causal or temporal) paths leading to them and thereby establishing the risk that they may occur. This will be discussed in more detail in the next section.

Human error hazard identification

Hazard analysis and error identification are systematic processes applied to an emerging or completed design to gather information about the design's dependability. There are a number of such methods and they have features in common. For example, in all cases some form of design representation is used as a starting point. This representation could be a process architecture of the system or a procedural description of how the system behaves for example. The second common feature is some kind of procedure for asking structured questions systematically about the designed system. These questions are used to identify significant parts of the system where dependability is problematic. None of the methods involves quantification. They involve a variety of possible techniques including for example HAZOP, Technique for Human Error Assessment (THEA), TRACEr as examples. The HAZOP method is a structured process [ICI (Imperial Chemical Industries Ltd.) 1974] systematically performed by a team on some representation of the system. There are a number of human orientated versions of HAZOP that apply the analysis to task representations [Leathley 1997] and have been applied to a variety of safety critical human operator interfaces. Like human HAZOP, THEA [Pocock et al. 2001] also begins with an explicit description of the work that is under analysis in this case scenarios because they provide a richer cultural context for the analysis of the system. The identification and explanation of human error is based on Norman's cyclic model of human processing and a series of questions are asked based on this model. This model is presented in Figure 2. It shows that a user's behaviour while interacting with a system is iterative starting by the identification of a goal, then defining a

plan for actions to be performed within the system and then performing these actions and thereby considering the user interface of the system. The users then observe changes on the user interfaces and evaluate whether those changes correspond to their initial goals. If not, a new plan is defined and the cycle performed again. Using this model, errors can be regarded as failures in cognitive processing [Norman 1988].

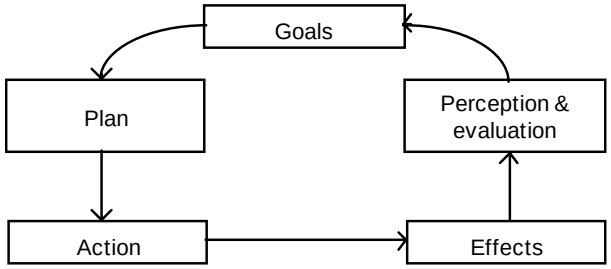

Figure 2 Norman's cyclic model of human information processing

TRACEr [Shorrock and Kirwan 2002] is a method of human error identification that has been developed for a particular domain - air traffic control. It involves several layers of analysis and is therefore relatively difficult to apply. Whether these layers are justified in the sense that they lead to more accurate analyses remains to be seen. The method is intended to deal with three issues: the context within which the error occurred, the mechanisms that lead to the production of the error and the recovery mechanisms from potential error. The method is intended to have a modular structure and therefore allows the analyst to describe the error at a level for which there is information about the system and its use. TRACEr also proposes retrospective analysis – analysing an incident using a similar technique. Analysis of an incident is carried out in terms of error chains, using a technique akin to Why Because Analysis [Ladkin and Loer 1998].

Very little credible validation has been performed on these techniques. Shorrock as part of an MSc thesis in 1997 [Shorrock 1997] analysed inter-analyst reliability (that is consistency between analysts) on a prototype version of TRACEr. Nine human factors specialists individually classified 23 different events highlighted in four controller-reported Airprox reports. For each event on average 5 out of 9 analysts agreed on the same error type category. Note that in this study the method was used retrospectively. A further study [Kirwan 1992] identified the analyst effort involved as the major weakness of the method. Newcastle has been concerned with the nature of the argument, its structure and, in particular, the way in which barriers or defences are used in mitigation **[Sujan and Harrison 2006;** Sujan et al. 2006b].

It is important to understand that all these aspects have to be taken into account in order to design and produce more resilient systems. In **[Basnyat and Palanque 2006]** it has been shown how software barrier modelling can be integrated into ICO device models (see comments about ICO above). In this work a software barrier is an artefact of a software based design aimed at preventing or protecting against failure. Hence a software interlock preventing an action from being carried out unless a prior action has occurred would be such a barrier. In [Basnyat PhD] a generic framework for integrating barrier modelling with task and device modelling is proposed. Additionally, [Basnyat et al. 2006] show how incident and accident investigation and modelling techniques can be integrated.

Probabilistic safety analysis and generic human reliability data

Probabilistic safety assessment of human error is usually based on databases of probabilities to estimate probability of failure for action. There are a number of issues associated with the use of generic human reliability data. Much of it has been collected in laboratory or simulator. Many of the techniques, for example THERP [Swain and Guttman 1983] are "corrected for" nuclear power plants. There is therefore an element of domain specificity and the translation to a new domain may not be obvious. There is wide variability in the data, for example [Villemeur 1992] comments that the range of probability of failure for action for automatic acts is 5×10^{-5} to 5×10^{-3}, while for rule based acts it is 5×10^{-4} to 5×10^{-2} and for knowledge based acts 5×10^{-3} to 5×10^{-1}. It is clear that circumstances will affect these values enormously, for example time pressure can have a significant effect on the probability of success. THERP addresses time explicitly in their assessment of likelihood.

Analysis is applied to some unit which varies between an action ("unlock restraining latch") in the case of an approach like THERP and generic task (for example "Restore or shift a system to original or new state following procedures, with some checking") in the case of HEART (Human Error Reduction Technique) [Kirwan 1992]. In the case of CREAM (Cognitive Reliability and Error Analysis Method) [Hollnagel 1998], the unit of analysis is even smaller and is associated with a so-called cognitive function (e.g. "act") which appears as one of the stages in the Norman cognitive loop [Norman 1988]. It should be said that while the CREAM technique is described to this level of detail, it is only proposed that it be used to provide broad assessments of error likelihood based on combinations of performance shaping factors relating to the whole organisation.

These units can either be part of a fault tree, where the actions of THERP might correspond to a fault in the fault tree or may be a generic task characterised by a cut set in the fault tree. Alternatively, or additionally, the part of the system that is subjected to analysis may have been uncovered through a process of error identification as described above. These units are associated with a database of human failure probabilities that are derived from data gathered through experience of some specific domain, either through experience or through simulations. In the case of THERP and some other techniques, for example HCR [Kirwan 1992] other more complex calculations are involved in assessing the probability including for example the time available. In the case of THERP time is also used as a measure in deciding what the value should be. Expert and therefore subjective judgement is involved in these calculations both at the level of deciding how the action or task corresponds to those generic items described in the database and, in the case of the more sophisticated techniques, deciding how time effects the action in the proposed situation.

The typical means by which context, though not domain, is corrected for is to take account of so-called performance shaping factors (there are a variety of other terms, for example "error producing conditions", "common performance conditions" for example "little or no intrinsic meaning in a task"). These performance shaping factors are used in a variety of ways to modify a nominal probability extracted from a database. The performance shaping factors are combined based on the team of analysts' assessment of the proportion attached to a particular performance shaping factor. The weighted product is then used as the modification factor. Expert judgement is again crucial, both in determining which performance shaping factors are relevant and in deciding what weightings should be attached to these performance shaping factors. Typically, performance shaping factors are assumed to be independent, though CREAM [Hollnagel 1998] bases "control modes" on common performance conditions. These common performance conditions are considered to be mutually interactive and a simple decision tree is proposed for ensuring that relevant mutual dependencies are taken account of. One major distinction between these methods is whether the method only provides the means to identify errors in a qualitative analysis or whether it also provides an estimate of

probabilities of error. Because human performance is sensitive to numerous contextual factors as well as individual differences, methods are often based on assumptions which are specific to a particular domain such as nuclear power, air traffic control and so on. Where analysis stops at error identification, this is less of a problem particularly if generic techniques such as human HAZOPS, THEA of CREAM are used. But it becomes particularly problematic where the quantification of error probabilities is undertaken. Many researchers have expressed serious doubts about the relatively common practice of using probabilities based on the nuclear industry in other domains such as aviation and air traffic control. Where this is done, error bounds on fixed point estimates are particularly important and should reflect the uncertainty introduced by such a transfer across domains.

The essentially engineering techniques reflected in this section are widely criticised. While these techniques depend on the ability to analyse and then combine components of a system, the complexity and non-linear nature of these systems leads to the unpredictable emergence of resilience characteristics that were not predicted. These issues are discussed comprehensively in [Hollnagel et al. 2006]. A problem is discussed and agenda is hinted at in this book. There are clear opportunities for solutions.

2.4. Formal verification of interactive systems

Three ResIST nodes are concerned with verification issues in relation to interactive systems by which are meant systematic techniques applied to models of interactive systems.

Although Harrison at Newcastle (formerly York) has done some work in the past using theorem provers, the focus of all current ResIST activity is on the use of model checking techniques to analyse interactive systems. While these techniques make it impractical to analyse representation type properties of a device, for example that a display reflects the relevant content of the state, model checking techniques have value in analysing an important class of properties of interactive systems that can be analysed using path type properties. Newcastle [Loer and Harrison 2006] have explored a class of properties inspired by so called usability heuristics [Nielsen 1992]. Their paper describes a front end to the Symbolic Model Verifier (SMV) (for general introduction see [Clarke Jr et al. 1999]) that supports the development of statechart based specification and the construction of properties using usability patterns similar to those discussed in [Dwyer et al. 1999]. The example given is a system for controlling a process plant making a comparative analysis between a control room interface and a hand held device that can be used in the plant to access and control processes (for example valves, thermostats, pumps) that are close by and also has limited capacity to “save” the controls for a particular process for future use even though the device is no longer in the neighbourhood of the process. Most of this group’s work to date revolves around the use of SMV and is concerned with making these tools accessible realistically to system designers. Further work is concerned with the use of Modal Action Logic as an alternative notation to analyse a mode confusion problem relating to a flight management system described in [Heymann and Degani 2007] and also analysed by [Rushby 2002]. In this case the mode confusion arises because pilot action has an unexpected effected when the flight management system is at the final stage of reaching a desire altitude when the aircraft is levelling out. The Newcastle group has also been concerned with the analysis of timing issues in interactive systems using the Uppaal tool to analyse appropriate scheduling decisions for systems involving human control [Loer et al. 2004]. In this case the example was a control task involving the manipulation and painting of boxes arriving on a conveyor belt. Current research at Newcastle involves the modelling of ambient and mobile systems, capturing the properties of systems that combine public displays with hand held devices in a built environment. Of particular interest here are properties of such models that capture the experience a traveller, visitor, etc.

would have within the built environment **[Harrison et al. 2006]; [Harrison and Loer 2006]** and a particular emphasis on the time properties of such systems. Here the motivating example is an airport system.

IRIT is also concerned with verification techniques rooted in the particular methods they use for the formal specification of interactive systems. Verification has been addressed for the following classes of systems and properties:

- Verification of functional properties expressed in temporal logic over an Interactive Cooperative Object (Petri nets-based) specification of the behaviour of interactive systems [Palanque and Bastide 1997] as well as verification of ergonomic criteria [Palanque et al. 1999;Accot et al. 1997].
- Verification of behavioural compatibility between different models involved in the description of interactive systems like a task model and a device model through scenarios [Palanque and Bastide 1995], [Palanque and Bastide 1997], verification of compatibilities between input devices, transducers and system [Accot et al. 1997].
- ISTI have used PEPA [Gilmore and Hillston 2003] and Markov chains to analyse user performance issues in a groupware system, namely thinkteam [Ter Beek et al. 2006].

This work should be seen in the more general context of the state of the art review provided by [Heymann and Degani 2007], and the more systematic, scaleable approach developed by [Berstel et al. 2005].

2.5. Issues of scale

Two studies by ResIST members exemplify the problems posed by human and social aspects in ubiquitous systems, although neither study goes all the way to attempting quantitative evaluation.

Newcastle researchers have been engaged in assessing pilot applications of an electronic transmission of prescriptions in the UK National Health Service. The focus in [Sugden and Wilson 2004] is on the probability of success of adoption of a new system. The important factors include resilience and dependability aspects like reliability of communication and usability, as well as “work around” strategies employed by users to obviate perceived problems of the automated systems or of the recommended procedure. There are concerns of availability, performance, reliability and data integrity (with possible safety implications) and privacy; both accidental faults (of the automation or of the users) and intentional attacks may threaten these properties. The authors emphasise the needs for multi-disciplinary evaluation.

Following Newcastle's interest in E-voting systems, where cryptography is applied to ensure both confidentiality and low probability of undetected tampering [Ryan 2005] (compare Part Eval of this deliverable), the question arises of evaluating such systems **[Bryans et al. 2006]**. It is striking that other authors in the E-voting area dedicate great care to a rigorous analysis of the algorithms themselves; yet, they only talk in informal terms about other essential components of the E-voting systems: the human voters and administrators of the election procedure with its ITC support. We have thus a huge number of components, most of them loosely coupled, yet subject to common and reciprocal influences acting through the election system as well from outside it, for example the media. Not only does proper evaluation of the system require an analysis of the behaviour of the human components, but also a prediction of how their patterns of behaviour may change as a result of the introduction of automation. Assessment must take into account the effects of both intentional attacks and unintentional failures, and their interactions. Some preliminary work took place in the DIRC project [Ryan and Peacock 2005]; **[Bryans et al. 2006]**. It is not difficult to recognise the E-voting system as a fault-tolerant system. In this system, although the automated algorithm

detects failures (violation of vote integrity, by accident or malice) with high probability, recovery actions are the task of the human components of the system; issues of trust in automation, perception of risk and threat will be important in the probability of initiation and success of these actions. The emphasis on error detection also increases the likelihood of "safe" failures, e.g., votes not being counted rather than risking an erroneous count. The proposed designs prudently restrict the need for high-confidence verification of hardware and software to as small as possible a set of crucial functions (the original proposal by [Chaum et al. 2004] suggested that the highly trustworthy error detection features would allow the use of ordinary PCs in public schools as voting machines); but strong error detection naturally increases the probability of failure: the design to avoid flawed election results needs to be complemented with design features to guarantee high enough probability of elections actually completing. These characteristics may change the strategies of would-be attackers – aiming, e.g., at denial of service attacks, either to discredit the election system or to cause a relaxation of the technical and social defences in the system. Discrediting the election system may become itself a direct means for defeating it, e.g., by weakening the voters' belief in its guarantees of secrecy to the point where widespread intimidation becomes possible.

In terms of large-scale modelling IRIT has previously been concerned with addressing the relationship of modelling techniques and frameworks to large scale systems within the specific context of interactive systems. [Jacomi et al. 1997] proposed the use of metaphors for structuring code in air traffic management interactive applications like airspace sectors definition and performance evaluation. [Palanque et al. 1997] presented scalability issues in modelling interactive applications for air traffic control while [Navarre et al. 2000] proposed structuring solutions for large models. In another domain, interactive cockpit applications for civil aircrafts, **[Barboni et al. 2006b], [Barboni et al. 2006b]** present how interaction techniques in modelling tools can support management of both numerous and large models of devices.

2.6. System evaluation

Various ResIST members are interested in the problem of inadequate or incomplete descriptive models of socio-technical systems, which in turn may lead to quantitative models embodying false assumptions about the system, or parameters that are too difficult to estimate in practice. Deep Blue [Pasquini et al. 2000] and Newcastle [Sugden and Wilson 2004] are among these. [Pasquini et al. 2000] discuss the shortcomings of common approaches to quantitative evaluation of human reliability: models that do not map on the actual psychological mechanisms and the way human behaviour depends on and is affected by knowledge distributed in the human's physical and social environment. They present a process developed to obviate these defects and applied to a case study in railway traffic management, which relies on multiple observational techniques, interviewing and historical data to create a more complete model of the socio-technical system and its possible behaviours.

Among ResIST partners, an example of results in the direction of system evaluation in the quantitative sense is the study by City University (with partners from the UK DIRC project) [Alberdi et al. 2004] of Computer Aided Detection (CAD) of cancer.

Here, the system to be modelled is composed of a physician examining X-ray images with a computer, which highlights areas of probable cancer. The quantitative evaluation question is to what extent this system is better (from the two viewpoints of detecting as many cancers as possible and recognising as healthy as many healthy patients as possible) than a doctor without the machine, or two doctors (co-operating according to one of the protocols in current use).

Remarkably, the estimates in the literature vary widely. In most of the medical literature, evaluation of such a system is via a controlled trial, a "black box" measurement of its effectiveness, to be extrapolated to future behaviour. But to make the measurement feasible, it has to be performed in conditions that are very different from those in actual operation (in particular, the frequency of cancer among the cases examined is artificially raised from a few per thousand to more than 10%). It is then difficult to decide to what extent any improvement or reduction in service quality observed in these measurements would be reproduced in actual medical use. The DIRC interdisciplinary approach involved probabilistic modelling, direct observation to challenge modelling assumptions and produce hypotheses about the underlying psychological mechanisms determining the behaviour of the human in this system, and controlled experiment to explore these conjectures and estimate ranges of model parameters [Alberdi et al. 2004]. A probabilistic model proposed by the City researchers in [Strigini et al. 2003] can be seen as a unifying structure for the multiple facets of evaluation: the differences between experimental environment and environments of practical use are captured by changes in the model parameters. In turn, focused controlled experiments can clarify which effects the probabilistic modelling must take into account and offer cues about the possible mechanisms underlying changes in human performance, and thus parameters of the probabilistic modelling, when using automated support [Alberdi et al. 2004], [Alberdi et al. 2005].

This study dealt with a small system with only two or three main components, yet its approach is somewhat innovative and experimental: scaling it up to systems of many machines and people is an open challenge.

Stochastic user models

A technique that combines statistical analysis with model checking has been used by the ISTI group. In [Doherty et al. 2001] both the device and aspects of user behaviour are modelled by using stochastic modelling techniques. These specifications are a means of making explicit the assumptions made about the capabilities of both user and device, and of exploring the behaviour of the combination of device and user on the basis of these assumptions. Predicted results from the stochastic model can thereby be related to data gathered by observing user behaviour. From the point of view of the overall development process, introducing performance data at an early stage is an attractive proposition since it encourages consideration of problems which might otherwise only emerge during testing, since neither prototypes nor high-specification development platforms are constructed with such issues in mind. Hence, we see such analysis as allowing an interactive development and validation loop to occur much earlier in the process.

The advantage of this approach is that it offers the capability of designing devices and adapting the design more effectively to a user's performance. However, interaction techniques and device structures cannot be ported across environments assuming that they will be equally acceptable in the new environment but must be validated again in the new hosting environment.

Failure modelling and quantitative arguments

As a result of hazard identification and analysis, certain events will be identified as critical to the safety of the system. On the basis of these events a number of fault or event trees can be constructed. These trees capture the possible logical or causal connections between events that can lead to these events. The trees are used to describe the conjunctions or disjunctions of faults/events that can lead to the identified event and can be used to capture alternative scenarios that might lead to it. Any given fault or event might be initiated either by the devices or by the humans in the system and might be seen as items already analysed in the

context of the hazard assessment method. The different hazard assessments may identify a number of events that are of interest. These fault/event trees can then be used as a basis for analysis either as a source for scenarios that can provide a basis for further descriptive assessment or as a means of providing estimates of the probability of failure.

In the figure below (from [Shorrock and Kirwan 2002] the concern is that a collision occurs in unregulated airspace whilst the aircraft is under the radar advisory service (RAS). In this diagram a triangle indicates that the event above the triangle must occur as a conjunction of the lower events while a circle indicates that any one of the events below may cause the event. Rounded boxes are events for which further causes are not considered. As can be seen a collision occurs if a conflict occurs and the controller fails to resolve the conflict, and the pilot fails to resolve the conflict and the aircraft are on the same level and converging tracks. Hence causes and consequences are embedded in the tree. It is therefore possible to identify sequences through the tree that take as given the fact that certain failures have occurred. This information could be used as the basis of a probabilistic assessment that the situation has arisen or for a further qualitative assessment.

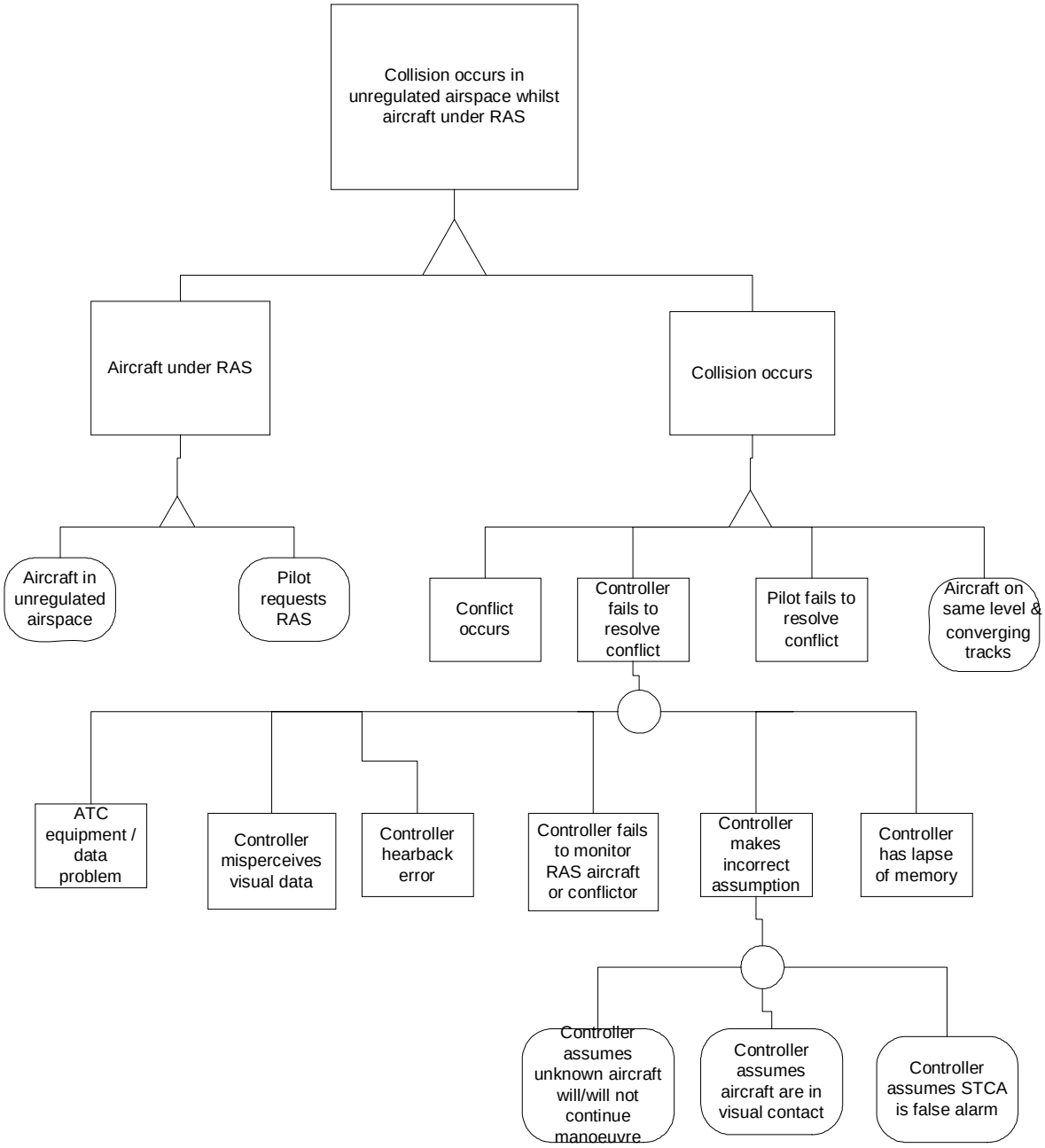

Figure 3: Fault tree for mid-air collision

Quantitative arguments depend in common practice on databases of known frequencies of failures for actions, events or goals. If the trees are to be used as a basis for assessing the probability of failure of the top event then there are a number of concerns.

Finding appropriate figures that are relevant to the current design, particularly when the design is new or modified in ways that might not have been anticipated, is non trivial.

Basic probabilistic data must be conditioned in the case of hardware by factors such as wear and tear and in the context of human factors the context in which the scenario proceeds. Although so-called performance

shaping factors or control modes have been designed to try to deal with these effects, their application is a matter of subjective judgement and may be highly sensitive to the team that is applying them.

The typical signature of a failure in which human error takes place is that instead of dealing with unrelated probabilities, the effect of failure may be to increase workload which will increase the likelihood of a future failure in the tree. The tree therefore needs to be conditioned not just by contextually determined performance shaping factors but also by the knock on effect of an earlier failure.

These difficulties are extremely significant and seriously jeopardise the possibility of deriving any meaningful value for the probability of failure. However these trees may have value in putting events from the different analyses together.

Conclusions

ResIST interest in resilience in socio-technical systems is focussed narrowly on a subset of topics concerned with the observation of such systems, the characterisation and modelling of devices and the analysis of various properties of the device and the system. In some respects the focus of interest differs from many of the prevailing themes because it focuses on a formal perspective on systems and their analysis. It is not concerned with the ergonomics of devices with mechanisms and conventions for communication for example. The ResIST network's focus in this area of socio-technical systems provides novel opportunities for the exploration of new and distinctive avenues in the analysis of resilience.

The predominant discipline represented within the socio group is computer science rather than behavioural science. This limitation though is overcome because all members of the group have strong links with psychologists and sociologists and long term research collaborations. The unusual focus on formal techniques fits well with the industrial scale formal methods used in other topic areas within ResIST. The use of formal notations brings precision and clarity to some of the discussion within the human factors community and provides more effective links to software and system engineering methods. However there is the risk that the essentially reductionist culture of formalism can miss some of the important ideas of socio-technical systems. This is a risk that the group is aware of and will take steps to avoid.

The particular focus of the working group provides strong opportunities for exploring issues in ubiquitous systems: problems associated with scaling resilience in socio-technical systems particularly in relation to the themes: evolvability; assessibility; usability and diversity.

References

[Accot et al. 1997] Accot, J, Chatty, S, Maury, S, and Palanque, P. "Formal Transducers: Models of Devices and Building Bricks for Highly Interactive Systems". 4th EUROGRAPHICS workshop on "Design, specification and verification of Interactive systems". 1997. Springer Verlag.

[Alberdi et al. 2005] Alberdi, E, Ayton, P, Povyakalo, A. A, and Strigini, L. Automation bias and system design: a case study in a medical application". Proc. IEE People & Systems Symposium. 2005.

[Alberdi et al. 2006] Alberdi, E, Ayton, P, Povyakalo, A. A, and Strigini, L. "Automation Bias in Computer Aided Decision Making in Cancer Detection: Implications for System Design". Technical Report, CSR, City University, 2006. 2006.

[Alberdi et al. 2004] Alberdi, E Povyakalo A. A, Strigini, L, and Ayton, P. "Effects of incorrect CAD output on human decision making in mammography". Academic Radiology 11(8): 909-918. 2004.

[Bannon 1991] Bannon, L. In J. Greenbaum & M. Kyng (eds.). "From Human Factors to Human Actors: The Role of Psychology and Human-Computer Interaction Studies in Systems Design". 1991. Design at Work: Cooperative Design of Computer Systems. Lawrence Erlbaum Associates.

[Barboni et al. 2006a] Barboni, E, Conversy, S, Navarre, D, and Palanque, P. "Model-Based Engineering of Widgets, User Applications and Servers Compliant with ARINC 661 Specification". Proceedings of the 13th conference on Design Specification and Verification of Interactive Systems (DSVIS 2006). 2006. Lecture Notes in Computer Science, Springer Verlag.

[Barboni et al. 2006b] Barboni, E, Navarre, D, Palanque, P, and Basnyat, S. "Exploitation of Formal Specification Techniques for ARINC 661 Interactive Cockpit Applications". Proceedings of HCI aero conference, (HCI Aero 2006). 2006.

[Barnard et al. 2000] Barnard, P. J, May.J, Duke, D, and Duce, D. "Systems, Interactions and macrotheory". ACM Trans Comput-Human Interact 7(2): 222262. 2000. Wesley, Reading, MA.

[Basnyat PhD] Basnyat, S. "A Generic Integrated Modelling Framework for the Analysis, Design and Validation of Interactive Safety-critical Error-tolerant Systems". PhD Thesis - University Paul Sabatier, Toulouse, France. To be defended December 2006. Submitted.

[Basnyat et al. 2006] Basnyat, S., Chozos, N., & Palanque, P. (2006) "Multidisciplinary perspective on accident investigation". Special Edition of Elsevier's Reliability Engineering and System Safety Journal.

[Basnyat and Palanque 2006] Basnyat, S and Palanque, P. "A Barrier-based Approach for the Design of Safety Critical Interactive Application". ESREL 2006 Safety and Reliability for Managing Risk. Safety and Reliability Conference. 2006. Balkema (Taylor & Francis).

[Basnyat et al. Submitted] Basnyat, S, Schupp, B, Palanque, P, and Wright, P. "Formal Socio-Technical Barrier Modelling for Safety-Critical Interactive Systems Design". Special Issue of Safety Science Journal. Submitted.

[Bastide et al. 2002] Bastide, R, Navarre, D, and Palanque, P. "A Model-Based Tool for Interactive Prototyping of Highly Interactive Applications. Full demonstration". ACM CHI 2002 conference on Human Factors for Computing Systems. 2002.

[Bastide and Palanque 1990] Bastide, R and Palanque, P. "Petri Net Objects for the design, validation and prototyping of user driven interfaces.". Proceedings Interact 90. IFIP TC13 Third International Conference. 1990.

[Berstel et al. 2005] Berstel, J, Reghizzi, S C, Roussel, G, and Pietro, P. S. "A scalable formal method for the design and automatic checking of user interfaces.". ACM Transactions on Software Engineering and Methodology 14(2): 124-167. 2005.

[Beyer and Holtzblatt 1998] Beyer, H and Holtzblatt, K. "Contextual Design: defining customer-centred systems". Morgan Kaufmann. 1998.

[Billings 1996] Billings, C. "Aviation Automation: the search for a human-centred approach". 1996. Lawrence Erlbaum Hillsdale, NJ.

[Bowman and Faconti 1996] Bowman, H and Faconti, G. "Analysing Cognitive Behaviour using LOTOS and Mexitl". Formal Aspects of Computing, Vol. 11:132-159. 1996. Springer Verlag.

[Bowman et al. 1998] Bowman, H, Faconti, G, and Massink, M. "Specification and Verification of Media Constraints Using UPPAAL". Proceedings of DSV-IS'98. 1998.

[Bryans et al. 2006] Bryans, J. W, Ryan, P. Y. A, Littlewood, B, and Strigini, L. "E-voting: dependability requirements and design for dependability". First International Conference on Availability, eliability and Security (ARES'06). 988-995. 2006.

[Campos and Harrison 2001] Campos, J. C and Harrison, M. D. "Model Checking Interactor Specifications. Automated Software Engineering" . 8(3). 275-310. 2001.

[Chaum et al. 2004] Chaum, D, Ryan, P, and Schneider, S. A practical, voter-veriable election scheme. Technical Report CS-TR-880, University of Newcastle upon Tyne, 2004. http://www.cs.ncl.ac.uk/research/pubs/trs/papers/880.pdf. 2004.

[Chipman and Shalin 2000] Chipman, S. F and Shalin, V. L. "Cognitive Task Analysis". 2000. Lawrence Erlbaum Associates.

[Clarke Jr et al. 1999] Clarke Jr, E. M, Grumberg, O, and Peled, D. A. "Model Checking". 1999. MIT Press.

[Dearden et al. 2000] Dearden, A, Harrison, M, and Wright, P. "Allocation of function: scenarios, context and the economics of effort". International Journal of Human-Computer Studies, Vol. 52, No. 2, pp. 289-318 . 2000.

[Degani 1996] Degani, A. "Modeling Human-Machine Systems: On Modes, Error, and Patterns of Interaction." PhD thesis, Georgia Institute of Technology. 1996.

[Dekker 2006] Dekker, S. "Resilience engineering: Chronicling the emergence of confused consensus". In Hollnagel, E., Woods, D. D., Leveson, N. (Eds.) Resilience engineering. Concepts and Precepts. Ashgate, Aldershot, England. 2006.

[Diaper and Stanton 2003] Diaper, D and Stanton, N. "The handbook of task analysis for human computer interaction". . Lawrence Erlbaum Associates. 2003.

[Doherty et al. 2001] Doherty, G, Faconti, G, and Massink, M. "Reasoning about interactive systems with stochastic models". LNCS 2220, Lecture Notes in Computer Science 144-163, Springer-Verlag. 2001.

[Duke et al. 1994] Duke, D, Faconti, G, Harrison, M, and Paterno, F. Unifying Views of Interactors. In Proceedings of the Workshop on Advanced Visual Interfaces pages 143-152, ACM Press. 1994.

[Duke 1995] Duke, D. J. "Reasoning About Gestural Interaction". Computer Graphics Forum, Vol 14(3) NCC/Blackwell. Conference Issue: Proc. Eurographics'95. 1995.

[Duke et al. 1998] Duke, D. J, Barnard, P. J, Duce, D. A, and May, J. "Syndetic Modelling". Human Computer Interaction, Vol. 13, No. 4, 337-393. 1998.

[Dwyer et al. 1999] Dwyer, M, Avrunin, G, and Corbett, J. "Patterns in property specifications for finite state verification.". In 21st International Conference on Software Engineering. 1999 .

[Endsley and Kaber 1999] Endsley, M. R and Kaber, D. B. "Level of automation effects on performance, situation awareness and workload in a dynamic control task". Ergonomics, 42, 462-492. 1999.

[Faconti and Massink 1998] Faconti, G and Massink, M. Modelling and Verification of PREMO Synchronizable Objects. Formal Aspects of Computing, Vol. 10: 405-434. 1998. Springer Verlag.

[Faconti 1996] Faconti, G. P. "Reasoning on Gestural Interfaces through Syndetic Modelling". ACM SIGCHI Bulletin, V 28(3), July. 1996.

[Faconti and Duke 1996] Faconti, G. P and Duke, D. J. "Device Models". F. Bodart, and J. Vanderdonckt. Design, Specification and Verification of Interactive Systems - DSV-IS'96, pp.73-91. 1996. Springer-Verlag.

[Faconti and Massink 1997] Faconti, G. P and Massink, M. Using LOTOS for the Evaluation of Design Options in the PREMO Standard. BCS-FACS Northern Formal Methods Workshop, Electronic Workshops in Computing. 1997. Springer-Verlag.

[Fitts 1951] Fitts, P. "Human engineering for an effective air navigation and traffic control system". (National Research Council, Washington D.C.) Reprinted as Appendix 1 in Beevis, D., Essens, P. and Schuffel, H. (Eds) 1996 Improving Function Allocation for Integrated Systems Design. Technical report CSERIAC SOAR 96-01. 1951.

[Giblin et al. 2005] Giblin, C, Liu, A. Y, Müller, S, Pfitzmann, B, and Zhou, X. Regulations Expressed As Logical Models (REALM). 18th Annual Conference on Legal Knowledge and Information Systems (JURIX 2005). 37-48. 2005. , IOS Press, Amsterdam.

[Gilmore and Hillston 2003] Gilmore, S and Hillston, J. A survey of the PEPA tools. In Proceedings of the Second Workshop on Process Algebra and Stochastically Timed Activities (PASTA Secondi Piatti). pages 40-49. 2003.

[Gordon and Gill 1997] Gordon, S. E and Gill, R. T. "Cognitive Task Analysis". In C. Zsambok and G. Klein (Eds.). Naturalistic Decision Makin. 1997. Lawrence Erlbaum Associates.

[Harrison et al. 2003] Harrison, M, Johnson, P, and Wright, P. "Relating the automation of functions in multi-agent control systems to a system engineering representation.". E. Hollnagel. Handbook of Cognitive Task Design. pp.503-524. 2003. Lawrence Erlbaum Associates.

[Harrison et al. 1993] Harrison, M. D, Abowd, G. D, and Dix, A. J. Analysing Display Oriented Interaction by means of Systems Models. Byerley, Barnard and May. Computers, Communication and Usability: Design Issues, Research and Methods for Integrated Services. pp147-163. 1993. Elsevier .

[Harrison et al. 2006] Harrison, M. D, Campos, J. C, Dohery, G, and Loer, K. "Connecting rigorous system analysis to experience centred design". Workshop on Software Engineering Challenges for Ubiquitous Computing. 2006.

[Harrison and Loer 2006] Harrison, M. D and Loer, K. "Time as a dimension in the design and analysis of interactive systems". (in preparation).

[Harrison et al. 1989] Harrison, M. D, Roast, C. R, and Wright. P.C. "Complementary methods for the iterative design of interactive systems". G. Salvendy and M.J.Smith. Designing and Using Human-Computer Interfaces and Knowledge Based Systems. 1989. Elsevier Scientific, pp. 651-658.

[Heymann and Degani 2007] Heymann, M and Degani, A. "Formal Analysis and Automatic Generation of User Interfaces: approach,methodology, and an algorithm." . Human Factors journal, to appear. 2007.

[Hildebrandt and Harrison 2002a] Hildebrandt, M and Harrison, M. "The temporal dimension of dynamic function allocation". Proceedings 11th European Conference on Cognitive Ergonomics (ECCE 11). pp. 283-292. 2002.

[Hildebrandt and Harrison 2002b] Hildebrandt, M and Harrison, M. "Time-related trade-offs in dynamic function scheduling". Johnson, C. 21st European Annual Conference on Human Decision Making and Control GIST Technical Report G2002-1, Department of Computing Science, University of Glasgow, Scotland. pages 89--95. 2002.

[Hildebrandt and Harrison 2003] Hildebrandt, M and Harrison, M. D. "Putting time (back) into Dynamic Function Allocation". Proceedings of the Human Factors and Ergonomics Society 47th Annual Meeting . pp. 488-492. 2003.

[Hildebrandt and Harrison 2004] Hildebrandt, M and Harrison, M. D. "PaintShop: A Microworld Experiment Investigating Temporal Decisions in a Supervisory Control Task.". Proceedings of the Human Factors and Ergonomics Society 48th Annual Meeting. pp. 300-304. 2004.

[Hoffman et al. 1995] Hoffman, R. R, Shadbolt, N. R, Burton, A. M, and Klein, G. "Eliciting knowledge from experts: A methodological analysis". Organizational Behavior and Human Decision Processes 62(2), 1995. 1995.

[Hollnagel 1998] Hollnagel, E. Cognitive Reliability and Error Assessment Method (CREAM). 1998. Elsevier.

[Hollnagel et al. 2006] Hollnagel, E, Woods, D. D, and Leveson, N. " Resilience engineering. Concepts and Precepts.". Ashgate. 2006.

[ICI (Imperial Chemical Industries Ltd.) 1974] ICI (Imperial Chemical Industries Ltd.). Hazard and Operability Studies. Process Safety Report 2. 1974.

[Jacomi et al. 1997] Jacomi, M, Chatty, S, and Palanque, P. A Making-Movies Metaphor for Structuring Software Components in Highly Interactive Application. Proceedings of the 12th BCS Human-Computer Interaction conference HCI'97. 1997.

[Jonassen et al. 1999] Jonassen, D. H, Hannum, W. H, and Tessmer, M. "Handbook of Task Analysis Procedures". 1999. Greenwood.

[Kirwan 1992] Kirwan, B. "Human error identification in human reliability assessment". Part 1: overview of approaches. Appl. Ergon. 23(5) 299-318. 1992.

[Kirwan 1994] Kirwan, B. A guide to practical human reliability assessment. 1994. Taylor & Francis.

[Ladkin and Loer 1998] Ladkin, P and Loer, K. Why-Because Analysis: Formal Reasoning About Incidents. RVS-Bk-98-01, 1. 1998.

[Leathley 1997] Leathley, B. A. "HAZOP approach to allocation of function in safety critical systems. ". ALLFN'97 Revisiting the Allocation of Function Issue. pp. 331-343. 1997. .IEA Press.

[Loer and Harrison 2006] Loer, K and Harrison, M. D. "An Integrated Framework for the Analysis of Dependable Interactive Systems (IFADIS): its tool support and evaluation". Automated Software Engineering. 13:4 pp 469-496. 2006.

[Loer et al. 2004] Loer, K, Hildebrandt, M, and Harrison, M. "Analysing dynamic function scheduling decisions". C.Johnson and P.Palanque. Human Error, Safety and Systems Development. pp. 45-60. 2004. Kluwer Academic.

[Marais et al. 2004] Marais, K, Dulac, N, and Leveson, N. "Beyond Normal Accidents and High Reliability Organizations: The Need for an Alternative Approach to Safety in Complex Systems. " . Engineering Systems Division Symposium, MIT, Cambridge, MA. http://sunnyday.mit.edu/papers/hro.pdf. 2004.

[Massink and Faconti 2002] Massink, M and Faconti, G. "A reference framework for continuous interaction". International Journal Universal Access in the Information Society. Electronic journal Vol. 1 n. 4. 237-251. 2002. Springer.

[Navarre et al. 2002] Navarre, D, Palanque, P, and Bastide, R. "Reconciling Safety and Usability Concerns through Formal Specification-based Development Process". Proceedings of HCI aero conference (HCI-Aero'02). 2002.

[Navarre et al. 2000] Navarre, D, Palanque, P, Bastide, R, and Ousmane, S. "Structuring interactive systems specifications for executability and prototypability". 7th Eurographics workshop on Design, Specification and Verification of Interactive Systems, DSV-IS'2000. 2000. Springer Verlag Lecture notes in computer science. n° 1946.

[Navarre et al. 2001] Navarre, D, Palanque, P, Bastide, R, Paternó, F, and Santoro, C. "A tool suite for integrating task and system models through scenarios". In 8th Eurographics workshop on Design, Specification and Verification of Interactive Systems, DSV-IS'2001. 2001. Lecture notes in computer science, no. 2220. Springer.

[Nielsen 1992] Nielsen, J. "Finding usability problems through heuristic evaluation". Proc. of ACM CHI'92 Conference on Human Factors in Computing Systems. pp. 249 256. 1992. New York., ACM.

[Norman 1988] Norman, D. A. The Psychology of Everyday Things. Basic Books. . 1988.

[Palanque and Bastide 1995] Palanque, P and Bastide, R. "Verification of an Interactive Software by analysis of its formal specification". Proceedings of the IFIP Human-Computer Interaction conference (Interact'95). p. 191-197 . 1995.

[Palanque and Bastide 1997] Palanque, P and Bastide, R. "Synergistic modelling of tasks, system and users using formal specification techniques". Interacting With Computers, 9, 12. pp. 129-153. 1997. Academic Press.

[Palanque et al. 1997] Palanque, P, Bastide, R, and Paterno, F. "Formal Specification as a Tool for Objective Assessment of Safety-Critical Interactive Systems". Interact'97 Conference. 1997. Chapman et Hall.

[Palanque et al. 2006] Palanque, P, Bernhaupt, R, Navarre.D, Ould, M, and Winckler, M. "Supporting Usability Evaluation of Multimodal Man-Machine Interfaces for Space Ground Segment Applications Using Petri net Based Formal Specification". Ninth International Conference on Space Operations, Rome, Italy, June 18-22, 2006.

[Palanque et al. 1999] Palanque, P, Farenc, C, and Bastide, R. "Embedding ergonomic rules as generic requirements in the development process of interactive software". A. Sasse and C. Johnson. Proceedings of IFIP TC13 Seventh International Conference on Human-Computer Interaction . 1999. Amsterdam, The Netherlands: IOS Press.

[Palanque and Paterno 2001] Palanque, P and Paterno, F. " Design, specification and verification of interactive systems". ACM SIGSOFT Software Engineering Notes. Vol 26 No 1. 2001.

[Pasquini et al. 2000] Pasquini, A, Rizzo, P, Scrivani, M, Sujan, A, and Wimmer, M. "Human, Hardware and Software Components in Control System Specification". Procs of the Sixth International Workshop on Requirements Engineering: Foundations of Software Quality REFSQ'00. 2000.

[Paterno 1999] Paterno, F. "Model-based Design and Evaluation of Interactive Applications". 1999. Springer Verlag.

[Pocock et al. 2001] Pocock, S, Harrison, M. D, Wright, P. C, and Johnson, P. D. THEA: a technique for human error assessment early in design. M. Hirose . IFIP TC 13 International Conference on Human-Computer Interaction. pp. 247-254. 2001. IOS Press.

[Pressman 1992] Pressman, R. "Software Engineering", Third European edition, McGraw-Hill Book Company. 1992.

[Reason 1990] Reason, J. T. Human error. 1990. Cambridge University Press.

[Reason 1997] Reason, J. T. Managing the risks of organisational accidents. 1997. Ashgate.

[Rosson and Carroll 2002] Rosson, M. B and Carroll, J. M. "Usability Engineering, scenario based development of human computer interaction". The Morgan Kaufmann Series in Interactive Technologies. 2002.

[Roth and Patterson 2000] Roth, E and Patterson, E. S. "Using Observational Study as a Tool for Discovery: Uncovering Cognitive and Collaborative Demands and Adaptive Strategies". Proceedings of the 5th Conference on Naturalistic Decision Making. 2000. IEEE Press .

[Rushby 2002] Rushby, J. "Using model checking to help discover mode confusions and other automation surprises. ". Reliability Engineering and System Safety, 75(2):167.177. 2002.

[Ryan and Peacock 2005] Ryan, P and Peacock, T. "Pret a voter: A system perspective". CS-TR: 929 School of Computing Science, University of Newcastle. 2005.

[Ryan 2005] Ryan, P. Y. "A variant of the Chaum voter-verifiable scheme.". Proceedings of the 2005 Workshop on Issues in the theory of Security (WITS'05. 81-88. 2005. New York, NY, ACM Press.

[Sagan 2004] Sagan, S. D. "The Problem of Redundancy Problem: Why More Nuclear Security Forces May Produce Less Nuclear Security". Risk Analysis, vol. 24, no. 4, pp.935-946. 2004.

[Scapin et al. 2000] Scapin, D, Vanderdonckt, J, Farenc. C, Bastide, R, Bastien, C, Leulier, C, Mariage, C, and Palanque, P. "Transferring knowledge of user interfaces guidelines to the web.". TFWWG '2000. 2000.

[Schupp et al. 2006] Schupp, B, S.Basnyat, S, Palanque, P, and Wright, P. A Barrier-Approach to Inform Model-Based Design of Safety-Critical Interactive Systems. 9th International Symposium of the ISSA Research Section Design process and human factors integration: Optimising company performances. 2006.

[Sheridan and Verplanck 1978] Sheridan, T and Verplanck, W. "Human and computer control of undersea teleoperators". Technical report. Man-machine systems lab, Dept of Mechanical Engineering, MIT, Cambridge, MA. 1978.

[Shorrock 1997] Shorrock, S. T. "The development and evaluation of TRACEr: a technique for the retrospective analysis of cognitive errors in air traffic control". MSc (Eng) Thesis. The University of Birmingham. September 1997. 1997.

[Shorrock 1997] Shorrock, S. T and Kirwan, B. "Development and application of a human error identification tool for air traffic control.". Applied Ergonomics 33: 319-336. 2002.

[Strigini et al. 2003] Strigini, L, Povyakalo, A, and Alberdi, E. "Human-machine diversity in the use of computerised advisory systems: A case study". DSN 2003, International Confernece on Dependable Systems and Networks. 249-258. 2003.

[Sugden and Wilson 2004] Sugden, B and Wilson, R. "Electronic transmission of prescriptions: an evaluation of technical models used in English ETP Pilots 2002". IFIP 2004 (WCC2004-I3E) Conference . 2004. Kluwer.

[Sujan and Harrison 2006] Sujan, M and Harrison, M. D. "Investigation of structural properties of hazard mitigation arguments". Analysis of the structure of mitigation arguments and the role of

barriers or defences with particular reference to the EUROCONTROL Reduced Vertical Separation Minima Functional Hazard Analysis. 2006.

[Sujan et al. 2006a] Sujan, M, Harrison, M. D, Steven, A, Pearson, P. H, and Vernon, S. J. "Demonstration of Safety in Healthcare Organisations". Proceedings SAFECOMP. Springer LNCS. 2006.

[Sujan et al. 2006b] Sujan, M, Smith, S, and Harrison, M. D. "Qualitative analysis of dependability argument structure". D. Besnard, C. Gacek, and C.B. Jones. Structure for Dependability: Computer Based Systems from an Interdisciplinary Perspective Springer. 2006.

[Swain and Guttman 1983] Swain, A and Guttman, H. "Handbook of Human Reliability Analysis with Emphasis on Nuclear Power Plant Applications". Technical Report NUREG/CR-1278 SAND80-0200 RX, AN, U.S. Nuclear Regulatory Commission. Final Report. 1983.

[Ter Beek et al. 2006] Ter Beek, M. H, Massink, M, and Latella, D. "Towards model checking stochastic aspects of the thinkteam user interface". Interactive Systems: Design, Specification and Verification, DSVIS 2005. 2006. Springer LNCS 3941.

[Vicente 1999] Vicente, K. J. "Cognitive Work Analysis". 1999. Lawrence Erlbaum

[Villemeur 1992] Villemeur, A. "Reliability, availability, maintainability and safety assessment. ". 1992. Wiley.

[Willans and Harrison 2001] Willans, J. C and Harrison, M. D. "A toolset supported approach for designing and testing virtual environment interaction techniques". International Journal of Human Computer Studies. 55(2) pp. 145-166. 2001.

[Winckler et al. 2004] Winckler, M, Barboni, E, Farenc, C, and Palanque, P. "SWCEditor: A Model-based Tool for Interactive Modelling of Web Navigation". R. Jacob, Q. Limbourg J. Vanderdonckt. Proceedings of 5th International Conference on Computer-Aided Design of User Interfaces CADUI'04 . pp. 55-66. 2004. Kluwer Academics Pub. Dordrecht.

[Winckler and Palanque 2003] Winckler, M and Palanque, P. "StateWebCharts: a Formal Description Technique Dedicated to Navigation Modelling of Web Applications". International Workshop on Design, Specification and Verification of Interactive Systems - (DSVIS'2003) . 2003. Lecture Notes in Computer Science n° 2669.

[Winckler et al. 2004] Winckler, M, Palanque, P, and Freitas, C. M. D. S. " Tasks and scenario based evaluation of information visualization techniques.". Proceedings of the ACM Conference on Task Models and Diagrams, page 165. 2004.

[Wreathall 2006] Wreathall, J. "Properties of resilient organizations: An initial view.". E.Hollnagel, D.D. Woods, and N.Leveson. Resilience engineering. Concepts and Precepts. 2006. Ashgate, Aldershot, England.

[Xiong et al. 2006] Xiong, J, Diouf, M, Farenc, C, and Winckler, M. "Automating Guidelines Inspection From Web site Specification to Deployment. " . 6th International Conference on Computer-Aided Design of User Interfaces CADUI´2006 jointly with 2nd International Workshop on Design and Engineering of Mixed Reality Systems MIXER´2006 . 2006.

Part Eval – Methods and Tools for Resilience Evaluation

Co-ordinator: Mohamed Kaâniche

Contributors: Jean Arlat[4], Andrea Bondavalli[6], Marc Dacier[2], Hervé Debar[3], Mohamed Kaâniche[4], Karama Kanoun[4], Bev Littlewood[1], Paolo Lollini[6], Nick Moffat[5], Aad van Moorsel[7], Lorenzo Strigini[1]

[1]City University, [2]Eurecom, [3]France Telecom, [4]LAAS-CNRS, [5]QinetiQ, [6]University of Florence, [7]University of Newcastle

Chapter co-ordinators:

1- Compositional modelling for large and evolving systems : Andrea Bondavalli

2- Evaluation with respect to malicious threats : Marc Dacier

3- Dependability benchmarking : Karama Kanoun

4- Diversity : Bev Littlewood

5- Dependability cases: Bev Littlewood

Introduction

The main objectives of resilience evaluation activities are to support design decision-making and to assess the level of confidence that can be placed on the target systems with respect to their ability to fulfil their missions at the desired dependability and security level. Resilience evaluation can be qualitative or quantitative, and the evaluation techniques can be based on analytical modelling, simulation, experimental evaluation or judgements. Models can be used at the early development stages to describe various alternative architectures of the system at different abstraction levels, and to analyze the impact on the system behaviour of different threat assumptions, error detection and recovery strategies, maintenance policies, etc. The assumptions considered in the models should be derived and validated based on measurements carried out through controlled experiments or direct observation of operational components and systems. Another critical aspect of resilience assessment is how to build valid arguments to support the dependability and security claims. Such arguments should take into account disparate evidence concerning the product characteristics as well the quality of the processes, the techniques and the tools used during its development, operation and maintenance.

The background of the ReSIST partners on resilience evaluation and assessment covers a large spectrum of topics and evaluation techniques, taking into account both accidental and malicious threats. This part summarizes the state of knowledge and ongoing research by ReSIST partners on methods and techniques for resilience evaluation, taking into account the resilience-scaling challenges and properties related to the ubiquitous computerised systems considered in the project. We mainly focus on quantitative evaluation approaches. Of course, for each of the topics addressed in this part, although the state of knowledge mainly focuses on the work done by ReSIST partners, the research covered is situated with respect to the corresponding state of the art in the field. It is noteworthy that the assessment of the impact of human interaction faults is not addressed in this chapter. This topic is covered in Part Socio- Section 3.3.

This part is structured into five chapters, each addresses an important challenge in the context of ReSIST and discusses the corresponding state of knowledge.

Chapter 1 deals with model-based evaluation techniques that are commonly used to evaluate and compare, from the dependability point of view, different architecture alternatives at the design stage. It mainly summarizes the state of knowledge related to compositional modelling techniques aimed at mastering the largeness of analytical dependability models at the construction and solution levels. Addressing the model largeness problem is important with respect to the investigation of the scalability of current techniques to meet the complexity challenges of ubiquitous systems.

The modelling techniques discussed in Chapter 1 have been traditionally used to assess the behaviour of computer-based systems in the presence of accidental faults. Recently, the extension of these techniques for the evaluation of security has been investigated. A discussion of the challenges related to resilience evaluation with respect to malicious threats, and of the various existing approaches to evaluate security is presented in Chapter 2. This Chapter includes in particular a summary of recent work carried out by ReSIST

partners on the collection and analysis of data characterizing real attacks and the development of stochastic models that can be used to assess the capacity of systems to resist to attacks.

More comprehensive approaches for resilience evaluation combining analytical modelling and controlled experiments are discussed in Chapter 3 dealing with dependability benchmarking. The objective of such benchmarks is to provide a uniform, repeatable and cost-effective way to evaluate dependability and security attributes either as a standalone assessment or, more often for comparative evaluation of candidate components or systems for integration into a given architecture. This chapter mainly presents the relevant properties and the main activities to be considered in dependability benchmarking and summarizes the challenges and recent studies carried out on dependability benchmarking, with respect to accidental faults and malicious faults.

Diversity is among the resilience scaling technologies identified in the ReSIST project to cope with the challenges raised by ubiquitous systems. Chapter 4 is dedicated to the evaluation of the efficacy of diversity taking into account various facets of diversity: diverse technical systems, human-computer diversity, process diversity as well as diversity in the arguments used to justify dependability claims. A more general discussion of how to build convincing arguments to justify the level of confidence to be placed in the dependability and resilience of systems is addressed in Chapter 5 dealing with dependability cases. In particular, this chapter focuses on safety cases and security cases.

It is important to note that the topics addressed in the five chapters of this part are not independent and various relationships and overlaps between the contents of the chapters can be highlighted. For example, in addition to Chapter 2, malicious faults are discussed in Chapter 3 dealing with dependability benchmarking approaches and in Chapter 5 in the context of security cases. They are also briefly mentioned in Chapter 4 dealing with diversity, as the problem of modelling diversity in the context of security is also open. Another example concerns the model-based evaluation techniques presented in Chapter 1, although these techniques have been traditionally used to address accidental faults, their application to obtain quantitative evaluation measures to characterize the security of operational systems has been also explored in the last decade as discussed in Chapter 2.

In the following, we summarize in more details the contents of the five chapters introduced above.

Compositional modelling for large and evolving systems

This chapter addresses model-based dependability evaluation approaches focussing on the techniques proposed in the literature, including by ReSIST partners, for mastering models largeness and complexity. Such techniques are needed to enable the evaluation of dependability measures at the early stages of the development process and to support the selection of architectural solutions that are best suited to satisfy the dependability requirements.

The complexity of models depends on the dependability measures to be evaluated, the modelling level of detail, and the stochastic dependencies existing between the components. State-space models, in particular homogeneous Markov chains, are commonly used for dependability modelling of computing systems, as they are able to capture various functional and stochastic dependencies among components and allow evaluation of various measures related to dependability and performance based on the same model, when a reward structure is associated to them. One of the major difficulties for the use of state-space models in dependability evaluation of real systems is the well-known state explosion problem. The question of how to cope with large state spaces has received much attention from the modelling community over the last two

decades. The various approaches that have been proposed to deal with this problem either seek to tolerate large state spaces, or seek to reduce large spaces into smaller ones. Generally, the proposed solutions combine both approaches.

First, this chapter reviews various approaches, focusing on the work developed by ReSIST partners, aimed at mastering complexity at the model construction level using model composition, decomposition/aggregation or the derivation of dependability models from high-level description formalism such as UML (Unified Modeling Language) or AADL (Architecture Analysis and Design Language) used in the context of model driven engineering development processes. The end of this chapter focuses on model-based evaluation approaches that have been developed recently to evaluate the dependability of large and evolving emerging systems such as web, grid, and mobile-based applications and systems. Such systems exhibit several scalability challenges that are representative of the types of systems targeted by the ReSIST project.

Evaluation with respect to malicious threats

Very recently, several studies have been launched to gain understanding of the malicious threats that computing systems are facing. They monitor malicious activities while remaining invisible to the attacker. The concept, by itself, is not new but its application to a large scale offers to the analysts substantial datasets that can be used to understand the *modus operandi* of the attackers. As a result, we observe a significant change in the research aiming at evaluating the resilience of systems with respect to malicious threats. This chapter presents the contributions made by the ReSIST partners in this area but also includes, a brief summary of the historical, classical way of addressing this issue. The chapter begins with an overview of traditional security evaluation criteria such as the TCSEC, the ITSEC, or the Common Criteria. Such criteria are mainly qualitative. They are based on the assumption that a proper design would suffice to prevent the introduction or activation of malicious faults in the system under consideration. While recognizing the necessity of prevention approaches to avoid vulnerabilities and to provide the systems with the functionalities needed to address malicious threats, such approaches need to be complemented with quantitative evaluation techniques to assess the ability of systems to resist to potential attacks. The section of this chapter dealing with model-based evaluation addresses this point by focusing on the seminal work carried out by several ReSIST partners, for introducing the use of probabilistic models in such evaluation. A more general discussion of the various approaches that exist to assess dependability of systems can be found in Chapter 3.

Model-based evaluations rely on assumptions that should faithfully reflect reality as the validity of the results depends on the validity of these assumptions. Several experimental studies aimed at the collection and analysis of data characterizing malicious activities on the Internet have been carried out during the last decade. Such studies are highly relevant for elaborating sound assumptions on malicious threats and attackers behaviours. A review of such studies, focusing on some ongoing work by ReSIST partners is presented at the end of this chapter, including a discussion of open issues in particular with respect to the exploitation of the experimental data to support the development of analysis and modelling methodologies allowing security evaluation to be used for driving design decisions.

Dependability benchmarking

Benchmarking the dependability of a system consists in evaluating dependability or performance-related measures, experimentally or based on experimentation and modelling, in a well-structured and standardized way. The development and conduct of benchmarks are still at an early stage in spite of major efforts and

progress made in recent years. The objective of a dependability benchmark is to provide a uniform, repeatable, and cost-effective way to evaluate dependability attributes, either as a stand-alone assessment or, more often, for comparative evaluation across systems and components. To be meaningful, a benchmark should satisfy a set of properties (representativeness, repeatability, portability, cost-effectiveness, etc.). The development of a practical benchmark instance is a complex task as it needs to mitigate all these properties. Indeed, in practice, these properties are not independent; some are correlated (e.g., portability and non-intrusiveness) while others are conflicting (representativeness and cost-effectiveness). With a few exceptions, most of the advances made to date in the development of dependability benchmarking have concentrated on moderate size target systems, taking into account accidental faults, mainly. On the other hand, research aimed at developing dependability benchmarks covering malicious threats is at an early stage. In particular, a summary of related work on benchmarks in the context of intrusion detection is provided at the end of this chapter.

Diversity

This chapter presents an overview of the research focussed on the modelling of diversity and the assessment of its effectiveness for improving dependability. The word “diversity” indicates those forms of difference between components that provide redundancy for one another in a system, in order to reduce the likelihood of common failures of these components defeating redundancy and causing system failures. In computing, the study of diversity has meant mostly diversity of design. The intuitive rationale behind the use of design diversity is simply the age-old human belief that ‘two heads are better than one’. Much of the interest in using diversity in recent years has concerned protection against software failure; however, similar issues arise in other areas, for example when any kinds of design fault may be present; or when a computer monitors a human’s performance (or vice versa). Besides bringing benefits when applied at the process and system levels, diversity can also bring benefits when applied to the arguments used to support claims for dependability, as part of a dependability case.

The importance of diversity for the “ubiquitous” systems that are the focus of ReSIST lies in at least the following factors:

- in large-scale systems, common failures can defeat the natural redundancy typical of these systems and may cause large-scale system failures;
- the limited dependability, and/or evidence thereof, of many off-the-shelf components on which these systems are built makes diversity a necessary remedy for many services to achieve their required dependability levels;
- “natural” diversity is often available through the interconnection of diverse large-scale systems (e.g. multiple communication networks), and evaluating the degree of protection - of resiliency - this offers to the services these ubiquitous systems provide, is essential.

Current knowledge about modelling diversity is mostly about modelling systems with very few diverse components: although the models may be suitable for exploring some aspects of ubiquitous systems (e.g., to what degree two diverse platforms in a network could be considered unlikely to succumb to the same attack), they need scaling up to address many other questions of interest.

This chapter surveys the state of knowledge and the challenges related to the evaluation of the effectiveness of diversity in dependability, using probabilistic modelling and experimental empirical studies, and considering various facets of diversity: diversity between technical systems, diversity between people and technical systems, and diversity in arguments.

Dependability Cases

Dependability cases are usually used for safety critical systems. A safety case is a documented body of evidence that aims to provide a convincing and valid argument that a system is adequately safe for a given application in a given environment. A main problem in this context is the difficulty of reasoning with disparate evidence, and disparate techniques for manipulating it: proofs about algorithms and designs, statistical observations of system behaviour, documentation of procedures and cultures of the developer and user organizations, etc. An overview of the challenges and current approaches for building cases to support dependability claims is provided, including in particular recent research results devoted to the development of quantitative approaches to dependability cases, where the confidence that can be placed in a dependability claim is expressed probabilistically. The problems and concepts related to safety cases can be also transposed to security cases. There is a growing belief that notations and tools used in the safety domain can be effective in the security domain. Recent work has attempted to bring structure and formalisms to security assurance arguments, to improve their clarity and help elicit security requirements.

1 – Compositional Modelling for Large and Evolving Systems

Introduction

The complexity of computerized systems has become a very critical issue in our daily lives. To master complexity when evaluating a system, a modelling methodology is needed so that only the relevant system aspects need to be detailed, allowing numerical results to be effectively computable.

The complexity of models depends on the dependability measures to be evaluated, the modelling level of detail, and the stochastic dependencies among the components. State-space models, in particular homogeneous Markov chains, are commonly used for dependability modelling of computing systems, as they are able to capture various functional and stochastic dependencies among components and allow evaluation of various measures related to dependability and performance (i.e., performability measures) based on the same model, when a reward structure is associated to them. Unfortunately not all the existing systems and their features can be captured properly by Markov processes; in some cases more complex processes (e.g. semi-Markov, Markov Regenerative or even more general processes) must be used. When dealing with such processes, complex and costly analytical solution techniques may exist, and otherwise simulation is the approach used to solve the models thus providing only estimates of the measures of interest. As an alternative one can approximate an underlying non-Markovian process with a Markov one, and thus represent a non-exponential transition with an appropriate set of exponential ones (Phased-Type approach). The price to pay following this approach is a significant increase in size, in terms of number of states of the resulting model. The large size of models known as the ‘state space explosion problem’ is one of the major difficulties connected to the use of state-space models in dependability evaluation of real systems. Much work has been done and significant progress has been made during the last twenty years in addressing such problems at the model construction and model solution levels. It is worth noting that model construction techniques and model solution techniques are complementary and both are needed when detailed and large dependability models need to be generated and processed to evaluate metrics characterizing the resilience of real life systems.

At **model construction level**, we can identify three complementary approaches: i) model composition; 2) system decomposition and model aggregation referred to shortly (decomposition/aggregation), and iii) the derivation of dependability models from high-level specifications based on languages such as UML (Unified Modeling Language) or AADL (Architecture Analysis and Design Language). In the model composition approach the system model is constructed in a bottom-up fashion. The models representing parts of the systems are built in isolation, thus having a limited view of the system context. On the contrary, the decomposition/aggregation approach follows a top-down approach: starting from an overall view of the system context, the model for the overall system is decomposed in a set of simpler sub-models. Last the

derivation of a dependability model from high-level specifications builds the overall model (described e.g., as a Markov chain or a stochastic Petri net), by transformation (usually semi-automatic) from a system model expressed in a specification language such as UML or AADL.

At **model solution level**, there are several techniques that can be divided with respect to their purpose and aim (largeness avoidance, largeness tolerance. Largeness avoidance techniques try to reduce the size of the generated models using many different approaches often complemented by largeness tolerance techniques which make use of space and time efficient algorithms to reduce the storage requirements of the state space and the generator matrix and to optimize the state space exploration, generation and analysis.

It is important to note that largeness avoidance and largeness tolerance techniques are complementary and both are needed, at the model construction and model solution levels, when detailed and large dependability models need to be generated and processed to evaluate metrics characterizing the resilience of real life systems.

We present in Section 1.1 three classes of structured techniques for a modular model construction. Section 1.2 addresses model solution techniques. Finally, given the increasing importance of web, grid and mobile based systems, emphasis is put on specific methods developed to deal with such large and evolving systems in Section 1.3.

1.1. Model construction techniques

1.1.1. Composition approaches

The principle of the composition approach is to build complex models in a modular way through a composition of its submodels. This class groups those techniques that build the system model as a modular composition of simpler submodels that are then solved as a whole. Most of the works belonging to this class define the rules to be used to construct and interconnect the submodels. They provide an easy way to describe the behaviour of systems having a high degree of dependency among subcomponents. These dependencies can be exploited to manage the model complexity creating, for example, smaller, equivalent representations.

The replicate/join formalism [Sanders and Meyer 1991, Sanders et al. 1995**]** is a composition technique for Stochastic Activity Networks (SAN) that combines models by sharing state, decreasing the overall number of states of the entire model. The Join operator takes as input a) a set of sub-models and b) some shared places belonging to different sub-models of the set, and provides as output a new model that comprehends all the joined sub-models elements (places, arcs, activities) but with the shared places merged in a unique one. The Replicate operator combines multiple identical copies of a sub-model, which are called replicates. It does not store the state of each sub-model, but rather the number of copies of the model that are in a particular state. Storing the number of models in a state, rather than recording which models are in that state, decreases the overall number of states of the entire model.

[Obal 1998] introduces a graph composition approach that extends the replicate/join formalism and it also combines models by sharing a portion of the state of each sub-model, reducing the total state-space size. Contrarily to the replicate/join formalism that requires the use of a special operation, the graph composition detects all the symmetries exposed at the composition level and uses them to reduce the underlying state

space. The graph composition uses the detected symmetries to replace each set of equivalent states with one aggregate state, reducing the total state-space size.

In [Rojas 1996], a more complete set of composition operators for the generation of Stochastic Well-formed Nets (SWN) (i.e., generalised stochastic Petri nets (GSPNs) permitting the identification of symmetry by means of a symmetric reachability graph) from its components has been defined. These operators preserve the functional structure of the model and support several types of communication between components. This approach is intended to support the modelling of distributed and parallel systems where both synchronous and asynchronous communications are required. However, it addresses only the class of systems that can be modelled by SWN.

[Rabah & Kanoun 1999, Rabah & Kanoun 2003] addresses multipurpose multiprocessor systems: the upper level models the service concerns, it is related to the application running on the architectural system (it defines the service levels, the needs in terms of resources, the maintenance policy, etc) whereas the lower-level models the behaviour of the architecture components with their interactions.

The block modelling approach in [Kanoun & Borrel 2000] defines a framework for modelling the dependability of hardware and software fault-tolerant systems taking into account explicitly the dependency between the various components. The modelling approach is modular: the behaviour of each component and each interaction is represented by its own GSPN, and the system model is obtained by composition of these GSPNs. The main advantage of this modelling approach lies in its ability to reuse modules which are thus developed and validated only once. It has been applied to a Regional Centre of the French Air Traffic Control system for which 16 alternative architectures, based on a reference one, have been modelled in an efficient way [Kanoun *et al.* 1999]. Most of the block networks developed for the reference architecture have been re-used for the 15 other alternatives, and only a few block nets have been either modified or newly developed.

In the incremental modelling approach, the model is built and validated in an *incremental* manner [Fota *et al.* 1999b]. At the initial step, the behaviour of the system is described taking into account the failures of only one selected component, assuming that the others are in an operational nominal state. The failures of the other components are then integrated progressively. The component's behaviour is described by sub-models (called *modules*) and the interactions between components are modelled using *module-coupling mechanisms*. At each integration step, the GSPN model is validated. The validation is carried out at the GSPN level (structural verifications) and also at the Markov level in order to check the different scenarios represented by the model. This approach has been successfully used to model the dependability of the main subsystems of the French air traffic control computing system referred to as CAUTRA [Fota *et al.* 1999a]. CAUTRA is implemented on fault-tolerant computers geographically distributed over five Regional Centres (those modelled by the above presented block modelling approach) and one Centralised Operating Centre, connected through a Wide Area Network. The models (GSPN and Markov) constructed for the different subsystems are very complex. For example the GSPN of the Radar Data Processing and the Flight Plan Data Processing Systems has about 100 places and 500 transitions and corresponds to a reduced Markov chain of about 25 000 states.

In the stepwise refinement approach [Betous-Almeida & Kanoun 2004-a], the construction of a GSPN dependability model is carried out progressively following the system development refinement process. Three main steps are distinguished. The first step is dedicated to the construction of a functional-level model. In the second step, the functional level model is transformed into a high-level dependability model. The third step is dedicated to the refinement of the high-level dependability model into a detailed dependability model.

A set of formal rules are defined to make the successive model transformations and refinements as easy and systematic as possible. A case study illustrating this approach is described in [Betous-Almeida & Kanoun 2004-b].

In the context of object-oriented programming languages, inheritance allows the definition of new classes based on already existing one(s), from which attributes and methods definitions are inherited. Recently, [Bernardi 2003, Bernardi & Donatelli 2004] have started to discuss the role of inheritance in the definition of GSPN models for dependability evaluations. Like a class **P** inherits from another class **Q**, a GSPN model **p** for objects of type **P** can be derived from a GSPN model **q** already defined for objects of type **Q**, reusing input parameters and result definitions of **q**. This type of compositional approach mitigates the model complexity since it discriminates the part of the system that needs to be modelled in detail (by means of sub-class models) from the rest of the system that can be modelled by using model components at a higher level of abstraction (by means of super-class models).

1.1.2. Decomposition/aggregation approaches

Most decomposition and aggregation methods are characterized by a hierarchical decomposition approach to avoid the generation of large models. The overall model is decoupled in simpler and more tractable sub-models, and the measures obtained from the solution of the sub-models are then aggregated to compute those concerning the overall model. In the following, we present different examples of decomposition/aggregation approaches proposed in the literature, including some proposed by ReSIST partners

In [Ammar & Rezaul Islam 1989]**,** the authors develop a technique that decomposes a GSPN model based on time scale. It is applicable to systems containing activities whose durations differ by several orders of magnitude. The given model can be decomposed into a hierarchical sequence of aggregated sub-nets each of which is characterized by a certain time scale. These smaller sub-nets are solved in isolation, and their solutions are combined to get the solution of the whole system. The aggregation at each level is done by assuming that the transitions included in the lower level are immediate transitions. At each level of the hierarchy, the current marking of an aggregated sub-net determines the number of tokens in the sub-net at the lower level, which are then analyzed to determine the rate of transitions in the aggregated sub-net.

[Ciardo & Trivedi 1993] proposes a decomposition approach for the solution of large stochastic Petri nets in which the overall model consists of a set of submodels whose interactions are described by an import graph. Each node of the graph corresponds to a parameterized stochastic Petri net submodel and an arc from submodel A to submodel B corresponds to a parameter value that B must receive from A. The authors show that the probability that a subnet is in a state satisfying a given condition, the average time a given condition remains satisfied, and the expected time until the subnet satisfies a given condition are three quantities that suffice for intercommunication among subnets for the net structure types that they define.

A logical decomposition for stochastic rendezvous networks (SRNs) is presented in [Woodside et al. 1995]. Stochastic rendezvous networks consist of tasks that take a random amount of time to complete and that may require the services of other tasks in order to complete. Each task is modelled separately, with the dependencies between tasks specified, and they communicate by messages in a request-wait-reply sequence (which models a ‘rendezvous’). Compared to Petri Nets, the SRN framework is at a higher level of abstraction. Queuing and synchronization involving inter-task messages are implicit, so a given model can be stated much more compactly. The SRN thus applies the concepts of queuing networks, which have been of great power in modelling hardware servers, to software and hardware servers together.

[Balakrishnan & Trivedi 1995] presents a decomposition approach for solving reliability models for systems with repairable components. The essential idea is to build one sub-model per component and to synthesize the system reliability from sub-model solutions. The sub-model for a given component must contain system-state information to ensure that the repair process is active only for the system up states. A natural construction procedure is to identify if the component in question is up or down, and to augment the sub-model with the system up/down information. This leads to a four-state sub-model having two absorbing states. Each sub-model state is an aggregate of system states. The authors show that the sub-models are Markovian but time-inhomogeneous since the transition functions vary with global time. This is an approximation because the parameters used in the sub-model are approximately derived from the monolithic model, since these functions are difficult to obtain.

Mura and Bondavalli have addressed the analytical dependability modelling of Phased Mission Systems (PMS) proposing several approaches, all based on a hierarchical structure of the models. PMS are characterized by a sequence of phases in which the system configuration can change during operations. The existence of phases is a consequence of: i) diverse tasks to be performed, and ii) diverse environmental conditions, in different periods of system lifetime.

First in [Mura and Bondavalli 1999] they propose a top down modelling with a Deterministic Time Markov chain for modelling the mission coupled with several GSPNs (one for each phase) and a bottom up solution approach whereby the different models corresponding to phases are solved to find the transient solution in the order they occur in the mission and thus provide values for the parameters of the discrete-time Markov chain.

Then in [Bondavalli *et al.* 1999, Mura *et al.* 1999], the model of a PMS is seen as composed of two logically separate Petri nets: the System Net (SN), representing the system (its components, their interactions and their failure/repair behaviour) as a GSPN, and the Phase Net (PhN), a deterministic and Stochastic Petri Net - DSPN, representing the control part and describing the phase changes. In the SN, a single model is built for the whole mission, characterized by a set of phases without detailing the behaviour of the system inside each phase. This allows easy modelling of a variety of mission scenarios by sequencing the phases in appropriate ways. The parameter values to be used in the SN model are obtained by solving the PhN models

Finally, [Mura and Bondavalli 2001] generalizes the approach by proposing a new methodology based on a Markov Regenerative Stochastic Petri Nets (MRSPN) approach (a Petri Net is an MRSPN if its underlying marking process is a Markov regenerative process). In particular, the state space of the Markov regenerative process never needs to be generated and handled as a whole, but rather the various subordinate intra-phase processes are separately generated and solved. Therefore, the computational complexity of the analytical solution is reduced to the one needed for the separate solution of the different phases as demonstrated in [Bondavalli and Filippini 2004].

In [Lollini *et al.* 2005-a], the authors analyzed a class of hierarchical control systems, in which the functionalities of the whole system are partitioned among a number of subsystems working at different levels of a hierarchy. The dependability of the whole system is enhanced considering, at each level, both internal checks and interface checks. A proper modelling methodology based on a decomposition approach has been defined, able to reduce the system complexity. They first decompose a model starting from its functional specification and applying a stepwise refinement to decompose it in small sub-models (decomposition). Then, modular model solution is carried out in a bottom-up fashion (aggregation). The methodology has been applied to evaluate some dependability attributes of the trial developed in the context of the European

project CAUTION++. The obtained results have allowed to understand the impact of several factors contributing to the dependability of the single CAUTION++ components on the overall system instance.

In [Nelli et al. 1996, Bondavalli *et al.* 2001], the study of a railway interlocking system leads the authors to the definition of a modelling strategy based on a modular and hierarchical decomposition. The system is modelled at the various level of the hierarchy. Each layer has been structured for producing some results while hiding implementation details and internal characteristics: the output values from one layer are used as parameters of the next higher layer. In this way the entire modelling can be simply handled. Further, different layers can be modelled using different tools and methodologies: this leads to flexible and changeable sub-models so that one can vary the accuracy and detail with which specific aspects can be studied.

[Daly 2001, Daly et al. 2001] describe a new set of connection formalisms that reduce state-space size and solution time by identifying submodels that are not affected by the rest of a model, and solving them separately. The result from each solved submodel is then used in the solution of the rest of the model. The authors develop four abstractions that can be used to make connection models, and they involve passing a continuous-time random process, a discrete-time random process, a random variable, and an average value between the models. When these abstractions are applied, each submodel should have a smaller state space and fewer time scales than the complete model.

1.1.3. Derivation of models from high-level specification and architecture description languages

Model-driven engineering approaches based on architecture description languages are more and more extensively used in industry. In particular, UML (Unified Modeling Language) ([OMG 2004]) and AADL (Architecture Analysis and Design Language) [SAE 2004] have received an increasing interest during the last years. These description languages provide standardized textual and graphical notations for describing software and hardware system architectures and functional interfaces, and for performing various types of analysis to determine the behaviour and performance of the system being modelled.

For efficiency and cost control reasons, system designers' will is to use an integrated set of methods and tools to describe specifications and designs, and also to perform dependability analyses. Accordingly, various studies have been focused on the development of modeling approaches allowing the generation of dependability evaluation models from model-driven engineering approaches.

A significant amount of research has been carried out based on UML. For example, the European project HIDE [Majzik & Bondavalli 1998, Bondavalli *et al.* 2001a, Majzik *et al.* 2003] proposed a method to automatically analyze and evaluate dependability attributes from UML models. It defined several model transformations: i) from structural and behavioural UML diagrams into GSPNs, Deterministic and Stochastic Petri Nets and Stochastic Reward Nets to evaluate dependability measures, ii) from UML state chart diagrams into Kripke structures for formal verification and iii) from UML sequence diagrams into Stochastic Reward Nets for performance analysis.

The issue of deriving automatically GSPN models or their coloured version (SWN) from UML behavioural specifications, represented by UML State Charts and Sequence Diagrams has also been addressed in [Bernardi 2003]. In addition to GSPNs, other types of quantitative evaluation models have been researched, e.g., an algorithm is presented in [Pai & Dugan 2002] to synthesize dynamic fault trees (DFT) from UML system models (a conjunction of class, object and deployment diagrams extended with stereotypes and tagged values).

Besides UML, AADL has more recently received increasing interest from industry and academia. The *AADL Error Model Annex* [SAE 2005] has been recently defined to complement the description capabilities of the core language by providing features with precise semantics to be used for describing dependability-related characteristics in AADL models (faults, failure modes and repair assumptions, error propagations, etc.). However, at the current stage, there is a lack of methodologies and guidelines to help the developers in using the proposed notations to describe complex dependability models reflecting real-life systems. A proposal for such a methodology is presented in **[Rugina et al. 2006]** where a stepwise approach for system dependability modelling using AADL and GSPNs is developed. A GSPN dependability model is generated automatically from the transformation of AADL and is built in an iterative way using composition modelling techniques that take advantage of previous approaches such as [Kanoun & Borrel 2000].

1.2. Solution approaches

At **model solution level**, various techniques have been proposed to facilitate the processing of state-based models by: a) avoiding the generation of large models and b) developing space and time efficient algorithms to optimize the state space generation, exploration, storage and analysis.

Largeness avoidance techniques try to circumvent the generation of large models using several different approaches:

- state truncation methods [Muppala et al. 1992, Chen et al. 2002],
- hierarchical model solution [Buchholz 1995, Sahner et al. 1996, Mura and Bondavalli 2001, Kaâniche et al. 2003a, Lanus et al. 2003],
- lumping techniques (state-level, model level or compositional) [Kemeny & Snell 1960, Buchholz 1994, Obal 1998, Derisavi et al. 2003, Derisavi et al. 2005],
- fixed-point iteration techniques [Mainkar & Trivedi 1996, Tomek & Trivedi 1991],
- model decomposition aggregation techniques [Courtois 1977, Bobbio & Trivedi 1986, Daly 2004],
- hybrid combination of different types of models, [Balbo et al. 1988, Malhotra & Trivedi 1994, Balakrishnam & Trivedi 1995, Nelli 1996], etc.

Largeness avoidance techniques may not be sufficient as the resulting model may still be large even after applying such techniques. In addition to those techniques, space and time efficient algorithms have been developed to reduce the storage requirements of the state space and the generator matrix and to optimize the state space exploration, generation and analysis. These model solution algorithms are generally used in combination with the modeling techniques aimed at mastering complexity at the model construction level. Many of the recent algorithms that have achieved notable success use symbolic data structures like binary or multi-valued decision diagram data structures, matrix diagrams or Kronecker representations (see e.g., [Ciardo & Miner 1999a, Ciardo & Miner 1999b, Miner & Parker 2004, Buchholz et al. 2000, Ciardo et al. 2001]).

1.3. Large and evolving systems

1. 3.1. Dependability modelling of Web-based systems and services

Growing usage and diversity of applications on the Internet make the issue of assessing the dependability of the delivered services as perceived by the users increasingly important. The Internet is often used for money

critical applications such as online banking, stock trading, reservation processing and shopping, where the temporary interruption of service could have unacceptable consequences on the e-business. Given the critical nature of such applications, the assessment of the user perceived quality of service is a key issue for e-business service providers and developers. Indeed, it is important for the developers to analyze during the architecture design phase how hardware, software and performance related failures of the infrastructure supporting the delivered services might affect the quality of service perceived by the users.

Quantitative measures characterizing user-perceived availability for web-based applications is widely recognized as highly important to evaluate the impact of failures from the business point of view. However, there is still a lack of modelling frameworks and examples illustrating how to address this issue. Indeed a significant body of work has focused on various aspects of web performance evaluation. Although many efforts have been dedicated to analyze the availability of web hosts using measurement-based techniques [Oppenheimer & Patterson 2002, Kalyanakrishnam et al. 1999], less emphasis has been put on the modelling of web service availability taking into account the impact of server node failures and performance degradations (see [Martinello 2005] for a review of the state-of-the art).

In [Kaâniche *et al.* 2003-a], a multi-level approach is proposed for modelling the user perceived availability of internet applications considering four abstraction levels, namely, user, function, service and resource levels. The highest level (user level) describes the availability of the application as perceived by the users. Intermediate levels describe the availability of functions and services provided to the users. The lowest level (resource level) describes the availability of the component systems on which functions and services are implemented. The availability measures of a given level are obtained based on the measures computed at the immediately lower level. Various techniques can be used to model each level: fault trees, reliability block diagrams, Markov chains, stochastic Petri nets, etc. The selection of the right technique mainly depends on the kinds of dependencies between the elements of the considered level and on the quantitative measures to be evaluated. This approach is illustrated in [Kaaniche et al. 2003-b] using a web-based travel agency as an example. In addition detailed analytical performability models are presented in [Martinello et al. 2005] to analyze the availability of web services implemented on cluster architectures taking into account explicitly: 1) the cluster architecture characteristics, considering the number of nodes in the cluster, the error recovery strategy after a node failure, as well as the reliability of cluster nodes, 2) the traffic model, describing the web traffic characteristics (i.e., the access patterns), and 3) various causes of service unavailability including requests loss due to buffer overflow or node failures.

Another facet of the web-services analyses has been considered in a research effort jointly performed by University of Firenze, ISTI CNR and BUTE. Here, in **[Gönczy et al. 2006]** a dependability modelling of web-based systems and services has been performed considering a business model workflow. This research considered web service-based workflows that fit into the class of systems composed of multiple phases, and applies modelling methodologies and tools for dependability analysis of Multiple Phased Systems (MPS). It was shown that this modelling approach and the corresponding tools (DEEM) provide a very useful support to service providers in choosing the most appropriate service alternatives to build up their own composite services.

1.3.2. QoS analysis of Mobile Telephone Systems

The work presented in [Porcarelli et. al. 2002, Porcarelli et. al. 2003] contributes to the analysis of GPRS by providing a modelling approach to understand the effects of outage periods on the service provision. A modelling approach has been followed, adopting the powerful modelling capabilities of Stochastic Activity Networks. A typical GPRS configuration has been deeply analyzed and evaluated in terms of a few identified

QoS indicators, namely, the number of served users per hour (and related measures) and the variation in system availability when including outages effects with respect to the bare network availability analysis. Interesting results have been observed, which can be fruitfully exploited in devising GPRS configurations adequate to maintain an acceptable QoS also in critical, overload conditions. Additionally, some sensitivity analysis has been performed at varying the frequency and duration of outages periods and for different values of network availability.

In [Lollini *et. al.* 2005-b], the congestion analysis of GPRS infrastructures consisting of a number of cells partially overlapping has been performed in terms of QoS indicators expressing a measure of the service availability perceived by users. When congestion is experienced by one of these cells, a family of congestion management techniques is put in place, to operate a redistribution of a number of users in the congested cell to the neighbour ones, in accordance with the overlapping areas. Since the service availability perceived by users is heavily impacted by the congestion experienced by the cells, determining appropriate values for the users to switch, so as to obtain an effective balance between congestion alleviation in the congested cell and congestion inducement in the receiving cells, is a critical aspect in such contexts. In order to carry on such fine-tuning activity, a modeling methodology, appropriate to deal with the system complexity, has been defined. In particular, they introduced a solution technique following a modular approach, in which the solution of the entire model is constructed on the basis of the solutions of the individual sub-models.

[Lollini *et. al.* 2006] presents a general modular modelling approach applicable to the wide class of cellular systems, including GSM, GPRS and UMTS networks. It is based on the identification of the building-blocks, the basic parts of the system to be modelled, and of their interfaces, the part of the building-block models that can interact with the others, and it enhances the modularity, reusability, scalability and the maintenance of the overall model. The modelling approach is applied to a concrete case-study, an UMTS network with overlapping cells, also accounting for the soft-handover mechanism. The goal is to analyze the QoS perceived by the users camped in the normal operational mode and during outage events that decrease the availability of the network resources.

1.3.3. Service Provisioning and Grid Systems

Various novel IT business models depend on adaptive infrastructure mechanisms to share resources, create distributed and collaborative applications, and manage and maintain systems and applications. Such platforms go by different names, including grid computing, service provisioning, utility computing, on-demand computing, etc. Such systems pose various new challenges on evaluation methods and techniques.

As an example, consider an organisation that provides infrastructure for large scale (possibly distributed) computation; other organisations may then access that infrastructure to provide further services to customers. Obviously, such partnerships pose great challenges with respect to trade-off between cost-effective high quality services. Since all parties in this process depend on the services they use, providers must ensure that their users are receiving a service quality which is acceptable; otherwise they will fail to meet the scientific or commercial aspirations of their clients and will lose business.

Similar to the emerging web services technologies, the increasing level of abstraction of the considered metrics is important. In [Jarvis et al. 2004] we listed various new challenges with respect to the performability evaluation of such systems. In the context of this discussion, the relevant challenges are: (1) the use of models in on-line decision-making, (2) abstraction to higher-level metrics and (3) unified and trade-off metrics crossing security, reliability, performance, etc.

The University of Newcastle has researched grid-related technologies in some detail. Of particular interest is to support the dynamism of the underlying system by deriving practical adaptation policies based on sound mathematical evaluation approaches. In [Palmer & Mitrani 2005], theoretically optimal policies to allocate resources to customers are computed, and compared with a newly proposed heuristic that is a variation on the well-known 'μc' rule in operations research. The heuristics has been validated and tuned using the experimental system described in [Fisher et al. 2004].

A methodological approach is proposed in [Machiraju et al. 2002] to systematically introduce metrics that have meaning for the business' operation and managers. It relies on the concept of 'Quality of Business' introduced in [van Moorsel, 2002], and is implemented based on contracts and/or service level agreements (SLAs). World-wide, research on SLAs has been particularly prevalent over recent years, and we review this area in [Molina et al. 2005]. The overarching trend is to use computer-readable versions of agreements (including, but not limited to, traditional SLAs) as the tool to set goals, rewards and penalties, and share these across adaptive mechanisms and organisations. On-line algorithms utilise these agreements, and policies may be parameterised (and changed) based on the agreements. For details, see [Molina et al. 2005].

Conclusion

As can be seen from the discussion in Section 2.2, the increasing scale and complexity of modern-day computing systems continues to put a premium on good techniques for the construction and solution of large quantitative models. In addition, these large, dynamic and evolving systems pose some new challenges that the ReSIST partners aim to address in their future research.

In particular, evaluation methods must deal with metrics at an increasingly high level of abstraction, to express the impact of the computing infrastructure on an enterprise business. Quality of Experience of a user is critical to evaluate the effectiveness of web services, and translation to Quality of Business metrics further increases the ability of managers to trade-off IT decisions, including decisions about resilience mechanisms.

Of increased significance is also the use of quantitative evaluation methods to support the effective use of adaptation mechanisms prevalent in modern-day systems. Virtualisation and service-oriented technologies have made possible the sharing of resources and services across users as well as between companies. Throughout such systems, mechanisms exist to dynamically change configurations, and well-founded quantitative techniques are necessary to run such systems in resilient fashion.

2 – Evaluation with respect to Malicious Threats

Introduction

Very recently, several systems have been deployed to gain understanding of the malicious threats that computing systems are facing. They monitor malicious activities while remaining invisible to the attacker. The concept, by itself, is not new [Stoll 1988] but its application to a large scale offers to the analysts very large datasets that can be used to understand the modus operandi of the attackers. As a result, we observe a significant change in the research aiming at evaluating the resilience of systems with respect to malicious threats. In the following, we present the contributions made by the ReSIST partners in this area but we also include, for the sake of conciseness, a brief reminder of the historical, classical way of addressing this issue. Therefore, three sections are proposed to the reader. Section 2.1, on security evaluation criteria, recalls when malicious threats have first been considered in the traditional evaluation of systems. Section 2.2, on model-based evaluation, focuses on the seminal work, carried out by several ReSIST partners, to introduce the utilisation of probabilistic models in such evaluation. Finally, Section 2.3, on experimental evaluation, presents some ongoing work and introduces papers published in the course of 2006 by ReSIST partners on these issues. It is noteworthy that other relevant issues related to resilience evaluation with respect to malicious faults are addressed in Section 3.3 of Chapter 3 dealing dependability benchmarking approaches with respect to intrusions.

2.1. Security evaluation criteria

Historically, malicious threats have been considered by researchers and engineers dealing with security issues. When they started evaluating the security of systems, the assumption was that a proper design would suffice to prevent the introduction or activation of malicious fault into the system under consideration. In 1967, a working group was created to study how to protect the classified information stored in computing systems. Its final report, published in 1970 [Ware 1970], led eventually to the publication in 1983 of the famous "*Trusted Computer Security Evaluation Criteria"*, also known as TCSEC or "*orange book"* [TCSEC 1983]. A revised version of this document became the reference document used by the Department of Defense in the USA [TCSEC 1985]. The evaluation of a product in this approach comes down to assigning it a security level, comprised between 1 and 7, by checking if several criteria are fulfilled.

As a follow up, several countries have proposed their own evaluation method. The most well known are the ones by the European Community [ITSEC 1991], Japan [JCSEC 1992] and Canada [CTCPEC 1993]. The Common Criteria (CC) represents the outcome of all these efforts. It is an alignment and development of them. The official version of the Common Criteria and of the Common Evaluation Methodology is v2.3. These standards have also been published as ISO Standards ISO/IEC 15408:2005 and ISO/IEC 18405:2005.

The CC structure provides great flexibility in the specification of secure products. Consumers and other parties can specify the security functionality of a product in terms of standard protection profiles, and independently select the evaluation assurance level from a defined set of seven increasing Evaluation Assurance Levels, from EAL1 up to EAL7. Depending on the chosen levels, different evaluation techniques have to be used by the evaluators.

A complete description of an "evaluation", as defined by the CC, lies beyond the scope of this document but it is worth noting that it goes much further than a simple identification of the required functionalities. It may, for instance, require to identify all exploitable vulnerabilities, which could be introduced by construction, operation, misuse or incorrect configuration of the system under evaluation. We refer the interested reader to the on-line portal dedicated to the common criteria for further information on this topic [CC 2006].

2.2. Model-based evaluations

In [Nicol et al. 2004], the authors offer a thorough review of the existing techniques for quantitative, model-based evaluation of computer system dependability. They explain that the use of quantitative techniques for security evaluation is much less common, and has typically taken the form of formal analysis of small parts of an overall design, or experimental red team-based approaches. The authors survey existing model-based techniques for evaluating system dependability, and summarize how they are now being extended to evaluate system security. We refer the reader to this reference for a complete treatment of these approaches and, instead, focus on a few seminal contributions from ReSIST partners as these are important to introduce the most recent, ongoing work, presented in the next Section. It is noteworthy that Chapter 3 discusses the various existing approaches to assess dependability and, therefore, also touches upon security evaluation. Also, the techniques discussed in this section can be seen as resulting from the adaptation and extension to the context of security, of the model construction and solution techniques presented in Chapter 1, that are commonly used to evaluate the impact of accidental faults.

As early as 1994, Brocklehurst et al. [Brocklehurst et al. 1994] have discussed similarities between reliability and security with the intention of obtaining "operational security" measures similar to the reliability measures that existed already. Ideally, such measures should capture quantitatively the intuitive notion of "the ability of the system to resist attack". The authors introduce in this paper the concept of *effort* to derive measures such as the *mean effort to next failure* and others. They validate the feasibility of such approach by reporting on a controlled experiment where students had been asked to penetrate various systems. (See [Littlewood et al. 1993] and [Jonsson and Olovsson 1997] for more on this work).

Concurrently and in the context of the same European Project, PDCS, [Dacier et al. 1993] present a method to model the influences of the remaining faults in a system if misused by an attacker. The authors use the very same notion of *effort* but also *time* to evaluate the security of system thanks to what the authors call a privilege graph [Dacier et al. 1994], which can be seen as an extension of the classical fault trees. [Ortalo et al. 1999] have built upon these ideas and have shown its usefulness by using a prototype, in an operational environment during several months.

These two important contributions to the domain of quantitative evaluation of the resilience of systems to malicious threats have been accomplished by ReSIST partners (City University and LAAS-CNRS).

More recently, intrusion detection techniques have become well accepted techniques as second lines of defence to detect the possibility of successful ongoing attacks. It has become well accepted not only that systems could fail in the presence of malicious threats but also, and this is more important, that these

defences could be subject to attack themselves. Evaluating the resilience of the protection systems itself against malicious threats becomes therefore of prime importance. Here too, two seminal contributions have been made by ReSIST partners. First, the approach proposed at IBM Zürich by Alessandri in [Alessandri 2000], where the author models the intrusion detection sensors in terms of their intrinsic properties. He also models the various classes of known attacks and is able to automatically derive the risks of a given detector to miss certain type of attacks. The author conjectures that, by the observation of real networks, one could quantitatively assess the resilience of an IDS (Intrusion Detection System) or even of a combination of several of them thanks to this method. Along the same line, but in a more formal manner, [Littlewood and Strigini 2004] note that redundancy and diversity, while commonly applied principles for fault tolerance against accidental faults, are less generally accepted in security. That paper discusses their roles and limits, and to what extent lessons from research on their use for reliability can be applied to security, in areas such as intrusion detection. The authors take a probabilistic approach to the problem, and argue its validity for security. They then discuss the various roles of redundancy and diversity for security, and show that some basic insights from probabilistic modelling in reliability and safety indeed apply to examples of design for security. They discuss the factors affecting the efficacy of redundancy and diversity, the role of "independence" between layers of defence, and some of the trade-offs facing designers. A more thorough treatment of diversity in this context can be found in Chapter 4. Also a more detailed discussion about the assessment of intrusion detection systems can be found in Chapter 3.

Common to all these approaches is the significant challenge that constitutes the need to obtain sound, representative and unbiased data to derive quantitative information about the typical behaviour of attackers. Indeed, it appears that all of these approaches require to get some or all of the following information: classes of attacks to be expected, rate of attacks, stability of the attack processes, skills of the attackers, etc.. For obvious reasons, companies that have suffered from attack are usually unwilling to share this painful experience with others. As a consequence, until recently, it was almost impossible to validate any such kind of model with meaningful data. Fortunately, as explained in the next Section, times are changing and, here too, several important contributions of the ReSIST partners, obtained since the beginning of the network, are to be mentioned.

For the sake of completeness, it is also worth mentioning an important emerging area of research, namely risk management in the context of cyber-insurance, which is closely related. In this field, the researchers not only need to evaluate the resilience of systems with respect to malicious threats but they also need to assess the risk that the failures could occur all together due to a common cause. The challenge is thus to evaluate a system of systems rather than a single one. Recent work has investigated the usefulness of diversity for security [Chen et al. 2006], the influence of the type of attacker on the security assessment [Collins et al. 2006] as well as the importance of having sound data sets at hand to validate some theoretical models [Boehme and Kataria, 2006].

2.3. Experimental evaluations

Empirical study of attacks in the Internet is an emerging research field. Early identification of new threats is a key element to implement counter-measures in a timely manner. Previous work focused on the propagation models of worms in general [Chen et al. 2003, Staniford et al. 2002, Zou et al. 2002] or the analysis of specific worms [FSECURE, McAfee, Moore 2002, Spafford 1989]. A few initiatives aimed at monitoring worms and attacks propagation in the Internet such as the so-called telescopes (also known as blackholes/darknets) or the DShield web site [CAIDA a, DSHIELD, Cooke et al. 2005, Zhou and Lang 2003]. Those approaches have highlighted the existence and dangers of certain types of malware (see for

instance the recent Witty worm analysis in [CAIDA b]). More traditional experimental evaluation concerning the performance overhead of security mechanisms has been carried out by **[Lamprecht et al. 2006]**. Such evaluation is important for future work to be carried out by ReSIST partners on dealing with the trade-off between security mechanisms and the resulting penalty in quality of service.

In the following subsections, we firstly introduce the notions of honeypots, honeynets and honeytokens for the reader who is not familiar with these concepts and, then, we briefly introduce four papers published by ReSIST partners since January 2006 and position their contributions to the evaluation of the resilience of systems with respect to malicious threats.

2.3.1. Honeypots, Honeynets, Honeytoken

The concept of “honeypots” has been introduced in computing systems by Clifford Stoll in the late 80’s. In the 'Cuckoo's Egg' [Stoll 1988], he describes the monitoring and tracking of an intruder. For this purpose, he created a complete but non-existent government project, with realistic but false files which intruders spent an extended period of time downloading and analyzing, providing an opportunity for him to trace back their origin. In the 90’s, Cheswick implemented and deployed a real “honeypot” [Cheswick 92]. Bellovin discussed the very same year the advantages and problems related to its usage [Bellovin 1992]. In 1998, Grundschober and Dacier ([Grundschober 1998, Grundschober and Dacier 1998]) introduced the notion of “sniffer detector” (see also [Abdelallah 2002]), one of the various forms of what is called today a “honeytoken”. As one can see, honeypots, honeytokens and honeynets have been used for some time in computing systems even if this terminology is quite recent. Indeed, it is only in 2001 that the term “honeypot” has been introduced by Lance Spitzner. Since then, several authors have proposed ad-hoc definitions and classifications. We refer the interested reader to [Pouget et al. 2003] for a thorough review of the terminological issues existing around this term.

In the context of this document, we will use the definition proposed by Lance Spitzner in his book *“Honeypots, Tracking hackers”* [Spitzner 2002]: “*A honeypot is security resource whose value lies in being probed, attacked or compromised.” [Spitzner 2002, page 40]*”. A Honeynet is a network of honeypot. A honeytoken is a honeypot where the security resource is intangible. A classic example of a honeytoken is described in the “John F. Kennedy” medical records example [Spitzner 2003]. U.S. hospitals are required to enforce the privacy of their patients. Access to their records is restricted to a few employees. A bogus medical record called “John F. Kennedy” is created and loaded into the database. As this patient does not exist, no one should ever try to access this record. If his happens, this is probably due to someone misusing his privileges and it can be detected thanks to this *honeytoken.*

2.3.2. Billy Goat

In [Riordan et al. 2006], the authors describe a tool named Billy Goat which is a worm detection system widely deployed throughout IBM and several other corporate networks. They describe the tools and constructions that they have used in the implementation and deployment of the system, and discuss contributions which could be useful in the implementation of other similar systems. They also discuss the features and requirements of worm detection systems in general, and how they are addressed by Billy Goat, allowing it to perform reliably in terms of scalability, accuracy, resilience and rapidity in detection and identification of worms without false positives.

The dataset used by the authors is unfortunately confidential as it belongs to IBM or IBM's customers, and could not be shared with other ReSIST partners. However, the lessons learned, the expertise gained and the

vision obtained for the future are elements that are discussed within the consortium. More specifically, regular exchanges exist between the responsible of this project and the one described in the next subsection. The ReSIST network offers an ideal framework to identify possible avenues for a real integration of these efforts without violating the secrecy of the information owned by the customers.

2.3.3 Leurré.com

In [Pouget et al. 2006], the authors describe new results obtained thanks to a worldwide collaborative environment to which several ReSIST partners participate. This environment is based on a low interaction honeypot platform duplicated in several places in the world (see [Pouget et al. 2004] for details). It emulates three virtual machines running different operating systems (Windows 98, Windows NT Server, Linux RedHat 7.3) with various services. Logs are centralized in a database that contains, for each attack, a large variety of information, such as:

- Raw packets (entire frames including the payloads are captured with tcpdump);
- IP geographical localization obtained with NetGeo, MaxMind and IP2location;
- Passive Operating System fingerprinting obtained with several tools (Disco, p0f and ettercap);
- TCP level statistics using Tcpstat;
- DNS reverse lookup, whois queries, etc.

The motivation behind the platform is to gather statistical data on attacks in the Internet taking a long-term perspective. The platform has been active for more than three years. During this period, more than a million of different IP addresses have been observed. Those addresses originate from more than 100 different countries. Only a negligible number of those addresses have been observed twice, that is on two different days. Currently, the distributed honeypot environment consists of around 40 entities located in 30 different countries, covering the 5 continents.

In this paper ([Pouget et al. 2006]), the authors present a method to detect the existence of sophisticated attack tools in the Internet that combine, in a misleading way, several exploits. These tools apply various attack strategies, resulting into several different attack fingerprints. A few of these sophisticated tools have already been identified, e.g. Welchia. However, devising a method to automatically detect them is challenging since their different fingerprints are apparently unrelated. They propose a technique to automatically detect these tools through their *time signatures*. They exemplify the interest of the technique on a large set of real world attack traces and discover a handful of those new sophisticated tools.

These results, combined with the ones obtained with the Billy Goat, open new perspectives for all researchers who desperately need sound statistical data about the real threats and also some understanding of the types of attacks that systems are likely to fight against. The lessons learned thanks to these complementary approaches are extremely useful and encouraging for the future work to be achieved for the evaluation of the resilience with respect to malicious threats.

2.3.4. Modelling of attack processes

The paper published at WEED2006 by ReSIST partners offers a nice illustration of the usefulness of the data gathered in the context of the Leurré.com project **[Kaaniche et al. 2006]**. This paper presents simple examples and preliminary models illustrating various types of empirical analysis and modelling activities that can be carried out based on the data obtained from the Leurré.com project in order to characterise attack processes.

For the authors, modelling involves three main steps: i) The definition of the objectives of the modelling activities and the quantitative measures to be evaluated; ii) The development of one (or several) models that are suitable to achieve the specified objectives; iii) The processing of the models and the analysis of the results to support system design or operation activities.

The data collected from the honeypots can be processed in various ways to characterise the attack processes and perform predictive analyses. In particular, modelling activities can be used to:

- Identify the probability distributions that best characterize the occurrence of attacks and their propagation through the Internet.
- Analyse whether the data collected from different platforms exhibit similar or different malicious attack activities.
- Model the time relationships that may exist between attacks coming from different sources (or to different destinations).
- Predict the occurrence of new waves of attacks on a given platform based on the history of attacks observed on this platform as well as on the other platforms.

The authors present several simple preliminary models based on the data collected from the honeypots that are aimed at fulfilling such objectives. These first results, that need to be investigated further, are extremely encouraging in the sense that they offer positive insight regarding the possibility to use such data to model attack processes happening over the Internet and even, potentially, to carry out some sort of limited forecasting as well.

2.3.5. Validation of the privilege graph concept

Finally, ReSIST partners have worked on building and using an experimental environment aiming at validating some of the models proposed in the past in the context of the privilege graph [Dacier et al. 1993, Dacier et al. 1994, Dacier et al. 1996, Ortalo et al. 1999]. Using some high-interaction honeypot under close scrutiny and configured in a very precise manner, they have studied the behaviour of attackers, over a long period of time and in a repetitive way, who had managed to gain access into a new target machine. The preliminary lessons learned from this work have been summarized in **[Alata et al. 2006]**. The analysis presented in this paper is also taking advantage of the data collected on the Leurré.com project. Indeed, the correlation of both sources of information enables the authors to better understand the various steps of a real world attack process. It offers them quantitative data to identify the techniques used by attackers to hide themselves and, last but not least, it enables them to have a sound estimation of the representativeness of the processes that they have analysed.

Here too, the results are encouraging in the sense that they have led to numerous discoveries. They also indicate, though, that the ideal target environment needs still to be worked out in order to deliver meaningful data to be used more specifically to feed the privilege graphs. This is one avenue for further work.

Conclusion

Today's, the development of efficient techniques and tools for analyzing malicious threats and evaluating their impact on the security of computing systems is necessary to enable the systems designers and administrators to improve the capacity of their systems in resisting to potential attacks. As reflected by the state of knowledge summary presented in this section, several advances have been achieved in this field

during the last decade, in particular by ReSIST partners. However, research on this topic is still at a preliminary stage. There are several open issues that need to be addressed with respect to security evaluation to cope with the resilience scalability challenges targeted by the ReSIST project. We can mention the following issues:

- definition of representative metrics and assumptions for quantifying security;
- development of analysis and modelling methodologies allowing security evaluation to be used for driving design decisions or to support the development of efficient early warning and intrusion detection systems that will enable the system administrators to better react to the attacks targeting their systems;
- trade-off analysis between security mechanisms and ensuing quality-of-service penalties. Such trade-offs are routinely made by engineers during the design stages, but this process lacks methodological support. Moreover, to support self-managing systems such trade-offs must be made increasingly frequent in on-line, automated fashion. This requires techniques to deal explicitly and formally (that is, using quantitative metrics) with trade-off issues that would otherwise be based on intuition or subjective arguments.

3 – Dependability Benchmarking

Introduction

Performance benchmarks are widely used to evaluate system performance while dependability benchmarks are hardly emerging. Several popular performance benchmarks exist for many specific domains that constitute invaluable tools to objectively assess and compare computer systems or components. This is a well-established area that is led by organizations such as the TPC[1] (Transaction Processing Performance Council) and SPEC[2] (Standard Performance Evaluation Corporation), and supported by major companies in the computer industry [Gray 1993]. Benchmarking the dependability of a system consists in evaluating dependability or performance-related measures, experimentally or based on experimentation and modelling, in a well-structured and standardized way. The development and conduct of benchmarks are still at an early stage in spite of major efforts and progress made in recent years both by the IFIP WG 10.4 SIG on Dependability Benchmarking[3] and within the IST project DBench[4] to which several ReSIST partners are contributing or have contributed.

To be meaningful, a benchmark should satisfy a set of properties (representativeness, repeatability, portability, cost-effectiveness, etc.). The relevant properties and the main activities to be considered in dependability benchmarking are outlined in Section 3.1. Sections 3.2 and 3.3 summarize the challenges and recent studies carried out on dependability benchmarking, with respect to accidental and malicious faults, respectively.

3.1. Dependability benchmarking approaches

Dependability benchmarking involves a new dimension to the *Workload* and (performance) *Measure* dimensions that typically characterize a performance benchmark: the *Faultload*. The Faultload defines the types of faults that are combined with the Workload to exercise the target system being benchmarked to obtain relevant Measures. In addition to performance measurements, more specifically performance degradation in presence of faults, the relevant measures attached to dependability benchmarking encompass a wide variety of facets including the assessment of system resilience, the characterization of failure modes and error signalling, the estimation of restart times, etc.

1 www.tpc.org
2 www.spec.org/
3 www.dependability.org/wg10.4/SIGDeB
4 www.laas.fr/DBench

Dependability benchmarks are usually based on modelling and experimentation, with tight interactions between them. In particular, analysis and classification of failure data can support the identification of realistic faultloads for the experimentation. On the other hand, measurements collected during experiments can be processed via analytical models to derive dependability measures. Some possible benchmarking scenarios can be sketched in reference to Figure 1, where layers identify the three steps *analysis*, *modelling* and *experimentation* and arrows A to E represent the corresponding activities and their interrelations [Kanoun et al. 2002]. For example a scenario consisting of modelling supported by experimentation would include the three steps and links A, B and E; in that case, the expected outputs are comprehensive measures obtained from processing the models.

Favouring the combined effects of the workload and the faultload in order to reduce the cases of non-significant experiments (e.g., see [Tsai et al. 1999]) involves complex composability analyses. Nevertheless, it is worth noting that the careful study of the interactions between the workload and the faultload is also central to the work addressing verification via robustness testing (e.g., see **Part Verif - Chapter 4**). Accordingly, cross-fertilization from these studies is expected to develop more efficient dependability benchmarks.

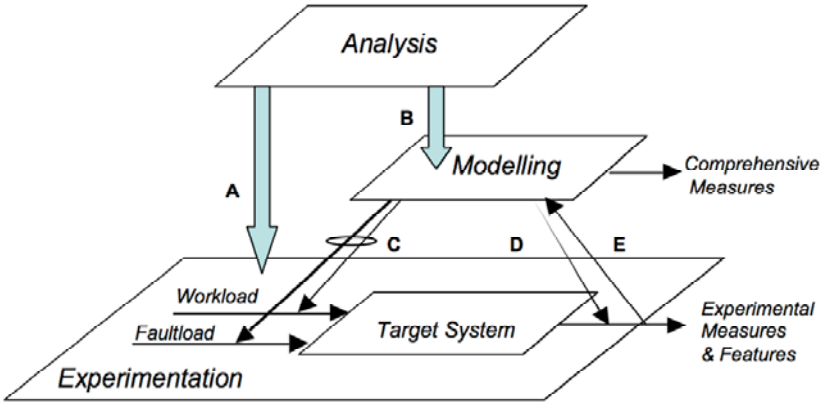

Figure 1: Dependability benchmarking steps and activities

To be useful and widely accepted by the community as a means to obtain a fair assessment and comparison of systems in a given class, dependability benchmarks must fulfil a set of specific properties. These properties must be taken into consideration from the earliest phase of the benchmark specification. The main relevant properties are briefly identified hereafter:

Representativeness: A benchmark must reflect the typical use of the target system. This requirement is a key issue and encompasses all the dependability benchmarking dimensions:

- It is essential that the workload profile used in the benchmark includes a realistic set of the activities found in real systems for the application area being considered. In practice, this aspect is often handled by considering existing performance benchmarks to implement the Workload.
- Faultload representativeness relies on how well injected faults correspond to real faults, i.e., faults affecting the target system in real use. Field data is the best way to validate whether a set of faults is to be considered in the faultload. The problem is that field data on the underlying causes of computer failures is often not widely available. The emergence of Open Source Software (OSS) paradigms and the wide usage of such OSS components provide new opportunities to address this issue. It is pertinent to note that previous works are available in the dependability literature that has been used to achieve

confidence on the representativeness of a given faultload (e.g., see [Christmansson & Chillarege 1996, Christmansson et al. 1998, Durães & Madeira 2002, Arlat et al. 2003, Jarboui et al. 2003, Siewiorek et al. 2004]).

- Measure representativeness has to do with the adequacy of i) the features that are reported [Wilson et al. 2002, Bartlett & Spainhower 2004], ii) the measurements that are carried out and optionally of their post processing to derive comprehensive measures (as illustrated by Figure 1), with the service expected from target system and its usage. In particular, the life cycle and the category of end-user to which the benchmark is focused, have an impact on the relevance of the measures. In practice, multi-dimension measures are preferred to an "averaged" figure. Indeed, in the first case, it is still possible to tailor the interpretation to suit end-user primary requirements (e.g., error reporting or continuity of service) [Durães & Madeira 2003, Albinet et al. 2004].

Repeatability: The results obtained for a specific application of the benchmark should be repeatable (within acceptable statistical margins) for the same system set up, even if executed by different end-users. This is clearly a prerequisite for the credibility of the benchmark. The resulting impact of this property differs whether the considered benchmark is provided under the form of a specification (TPC-like approach) or under the form of an executable prototype (SPEC-like approach).

Portability: This property is intrinsic to the concept of benchmark that by definition must be executed on various target systems. Portability depends on the way some key benchmark dimensions (such as the faultload and workload) are specified. For example, the more general the faultload is defined the more easily portable will be the benchmark. In practice, the underlying technology (fault injection mechanisms, measurement tools, etc.) must be portable to different platforms. The availability of standard interfaces greatly facilitates the fulfilment of this property.

Non-intrusiveness: The implementation of the benchmark in the experimentation step (particularly to what concerns the faultload and the measurements) should induce no — or really minimal — change to the benchmark target (either at the structure level or at the behaviour level). One implication in order to satisfy non-intrusiveness results in a practical rule that is to avoid fault injection within the benchmark target[5]. However, faults can be injected in the parts surrounding the benchmark target. The respect of this property has a restrictive impact on the granularity of the faultload and of the time measurements that can be carried out and reported.

Cost effectiveness: The effort in time and cost necessary to run the benchmark should be reasonably low, otherwise many users could be prevented from using it. It is worth noting that the benchmarking time consists of three parts: i) set-up and preparations, ii) running the actual benchmark (i.e., execution time) and iii) data analysis. Ideally one would acclaim benchmarks that are able to assess the target system thoroughly and accurately, while featuring short benchmarking times. Clearly, in practice, these properties call for a trade-off.

Scalability: The various properties of a benchmark must hold for differing sizes of instantiations of a class of target system. In connection with the viability property, this means in particular that time and cost of running the benchmark must increase at a radically lower pace than the complexity and size of the target system. Usually, scaling is achieved by defining a set of rules in the benchmark specification. Scaling rules mostly

[5] It is worth noting that the impact of this restriction can be related to the *robustness* attribute (i.e., dependability with respect to external faults [Avizienis *et al.* 2004]) and accordingly to the case of *robustness testing*.

affect the workload, but also the size of the faultload as the latter may affect the time needed to execute the benchmark.

In the subsequent sections, we summarize the current status of the results obtained on dependability benchmarking with respect to two categories of faults: accidental faults and intrusions, both by the ReSIST partners and elsewhere. It is worth pointing out that while quite a substantial body of research has addressed accidental faults, benchmarking with respect to intrusions is almost inexistent.

3.2. Accidental faults

Elaborating on the pioneering work started in the first half of the 1990's (e.g., see [DeBrunner & Gray 1992, Siewiorek *et al.* 1993]) several efforts have been conducted in the past decade to advance the conceptual issues and the development of prototype dependability benchmarks based on specific faultloads encompassing physical faults, software bugs or, to a less extent, operator mistakes.

Physical faults have received first a special attention both from academia and industry. In order to facilitate the conduct of the controlled experiments, these works have significantly built on several variants of the SWIFI (technique to implement the faultload, e.g., see [Barton et al. 1990, Kanawati et al. 1995, Carreira et al. 1998, Buchacker & Sieh 2002, Yuste et al. 2003]). Indeed, SWIFI exhibits relevant features, such as low-intrusiveness and good portability and it can emulate a wide variety of hardware faults and, to some extents, software faults as well [Madeira *et al.* 2000]. These proposals have covered a wide spectrum of application domains (critical embedded control systems, highly available distributed systems) as well as various components and layers in a computer architecture (processors, middleware, etc.) [Siewiorek et al. 1993, Tsai et al. 1996, Marsden et al. 2002, Wilson et al. 2002, Buchacker et al. 2003, Zhu et al. 2003, Ruiz et al. 2004, Siewiorek et al. 2004, Constantinescu 2005].

Accidental human interaction faults are reportedly a dominant factor in many critical systems and information infrastructures. In particular, the statistics covering the period 1959-2005[6] show that they steadily contribute as the principal cause. The figures for the period 1996-2005, indicate that they contribute as the major factor to 63% of the accidents in commercial aviation: crew 55%, airport/air traffic control 5%, maintenance 3%. Operator faults constitute also a significant ratio (almost 50%) of the causes of diagnosed failures in large-scale data centres as reported in [Oppenheimer & Patterson 2002] that has analysed three major Internet sites. This explains why efforts have tried to address this state of affair by including the human-factor aspect in dependability benchmarking proposals (e.g., see [Brown et al. 2002, Vieira & Madeira 2003]). These attempts are still preliminary and much work is still needed to cope with the complexity of the task.

Software faults constitute an increasingly large share of reported service disruptions in many application domains [Xu et al. 1999, Fenton & Ohlsson 2000, Charette 2005]. This trend is easily explained by the penetration of software-intensive computing systems in every day usage for example a typical cellphone now contains 2 million lines of software code) and also in many critical applications (65 million lines are quoted for the Airbus A380). Increased time to market pressure and the complexity attached to the handling of large software programs impose a heavy burden on the software life cycle (e.g., see [Bailey & Davidson 2003, Pighin & Marzona 2003]). This trend also favours the reuse of previously developed software. Accordingly, several works have addressed the dependability characterization of Off-The-Shelf (OTS) —

[6] *Statistical Summary of Commercial Jet Airplane Accidents – Worldwide Operations 1959-2005*, Boeing Commercial Aircraft Group, Seattle, USA, 2006 (http://www.boeing.com/news/techissues).

proprietary or open source — software components including both the software executive layer (e.g., operating systems (OS) [Koopman & DeVale 1999]) and the application layer (e.g., database management systems [Vieira & Madeira 2003], automotive application [Ruiz et al. 2004] or web services [Durães et al. 2004]). Due to their central role in a computer architecture, special emphasis has been put on the development of benchmarks to support the assessability and the testing of the robustness of software executives. These especially addressed general-purpose OS kernels, but some relevant works have also considered specific microkernels (e.g., see [Arlat et al. 2002]) or the middleware layer (e.g., see [Pan et al. 2001]).

As already pointed out, several attempts have been made to propose and implement **benchmark prototypes** supporting the assessability of OS kernels. Figure 2 depicts the software architecture of a computer system and provides a basis on which this thread of work can be illustrated. As shown on the figure, the kernel features three main interfaces with its environment. The first one (bottom) is basically concerned with hardware interactions, while the other two (top and right) are software related.

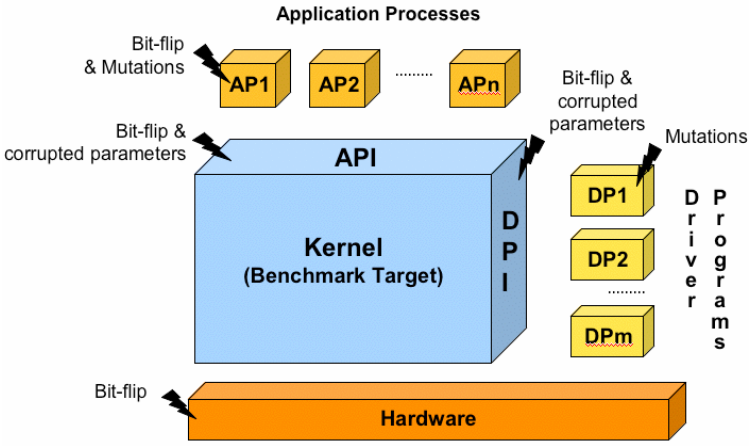

Figure 2: Interactions between an OS kernel and its environment and possible fault injection locations

The “lightning” symbols in Figure 2 identify possible locations where faults can be applied. These locations and related faults are briefly described as follows:

1) The main interactions concerning the bottom interface are made by raising hardware exceptions. Several studies (e.g., see [Arlat *et al.* 2002, Gu *et al.* 2003]) have been reported in which faults were injected by means of bit-flips into the memory of the system under benchmark or via special debugging interfaces [Ruiz *et al.* 2004].

2) The top interface corresponds to the Application Programming Interface (API). The main interactions are made via kernel calls. A significant number of studies were reported that target the API to assess the robustness of OS (e.g., under the form of code mutations [Durães & Madeira 2002]), by means of bit-flips [Jarboui *et al.* 2003] or by altering the system calls [Koopman & DeVale 1999, Kalakech et al. 2004, Kanoun et al. 2005]).

3) The third type of interactions are made via the interface between the drivers and the kernel. The proposal in [Durães & Madeira 2003] has concentrated on drivers code mutation. The work reported in [Albinet *et al.* 2004] proposes a complementary alternative that explicitly targets the exchanges made between the device drivers and the kernel via their interface the “DPI (Driver Programming Interface)” [Edwards & Matassa 2002].

The work reported in **[Kanoun and Crouzet 2006]** and **[Kanoun et al. 2007]** fit item 2 above, while item 3 is exemplified by the work described in **[Albinet et al. 2007]**. These contributions illustrate the benchmarking efforts targeting general purpose OS that were carried out at LAAS within the DBench project.

3.3 Intrusions

There are no intrusion detection benchmarks per se. However, a number of papers related to testing intrusion-detection systems have been published in the literature. In most cases testing methodology is proposed by the developers of a given intrusion-detection system method or tool, with the purpose of showing their own development at its best. This type of work is considered biased. We put emphasis on three testing approaches: the Lincoln Lab experiments, university and research testing environments and commercial environments, that are more detailed hereafter.

3.3.1. The Lincoln Lab experiments

One of the best known testing experiments in the intrusion-detection community is the Lincoln Lab experiment [Lippman et al. 2000-a, Lippman et al. 2000-b]. The initial purpose of this experiment is to test the various intrusion-detection technologies developed under DARPA funding, and to compare them in order to choose the most appropriate one. This is achieved by simulating a network of workstations to create normal traffic and by inserting a set of attacks carefully chosen to cover various situations, summed up as remote attacks, local attacks and denial-of-service attacks. This experiment has received scientific criticism from the community, most notably [McHugh 2000]. Our main criticisms are:

- Focus on background traffic: As the test includes behaviour-based intrusion-detection systems, realistic background data must be generated to ensure that these systems will be properly trained. However, while background traffic generation is required for performance testing, the Lincoln Lab testbed does not provide a way of calibrating the characteristics of this background traffic and verifying their compliance with specifications.
- Focus on a broad set of attacks: The Lincoln Lab tests aimed at exercising the largest possible set of attacks for the largest possible set of intrusion-detection systems. However, in some contexts one would like to focus on network traffic close to firewalls. Such kind of analysis cannot be done based on the Lincoln Lab tests that are too wide for such a use.
- Lack of reference point or baseline: The Lincoln Lab tests compare prototypes in a closed circle. There is no notion of a minimal set of requirements that a tested tool has to meet, only relative data comparing them with each other. The lack of a baseline that all tools would have to fulfil was felt as particularly lacking in the Lincoln Lab experiment.

3.3.2. University and research testing environments

The most representative work concerning university tests has been carried at UCDavis [Puketza et al. 1996, Puketza et al. 1997], with related and de facto similar work at IBM Zurich [Debar et al. 1998].

The objective of the UC Davis test is to simulate the activity of normal users and the activity of attackers, using script written in the *expect* language. *Expect* simulates the presence of a user by taking over the input and output streams of a tty-based application, matching expected responses from the application with the displayed output, and feeding appropriate input as if the user typed it on its keyboard.

A similar approach was followed at IBM Zurich. In addition to user-recorded scripts the IBM approach introduced software testing scripts from the *DejaGnu* platform (also in *expect*) to ensure that all aspects of an application were exercised, regardless of the fact that they were obscure features of an application or not.

3.3.3. Commercial testing environments

Tests classified as commercial include tests published by commercial test laboratories mandated by a particular company, and tests carried out by independent publications.

Several test labs have published test results related to intrusion-detection systems. In particular, Mier Communications has released at least two test reports, a comparative of BlackICE Sentry, RealSecure and NetProwler on one hand, and a test of Intrusion.com SecurenetPro Gigabit appliance. Appropriate queries on mailing list archives show that these test results have been the subject of many controversies. The general feeling is that it is appropriate for the testing laboratory to provide results that highlight the advantages of the product sponsored by the company financing the tests. Even if direct test manipulation is not suspected, at least test configurations for the competing products is likely not to be handled by experts, thus resulting in unfair comparison.

Our feeling is that even this cannot be determined, as the description of the tests is very sparse and does not provide information that would allow an impartial judgement of the tests. Normal traffic generation, for example, is done using commercial products and the test report is considered complete with only the mention of the name of the traffic generator, without any information on the kind of traffic it generates or its configuration.

Journalists have also been involved in the testing of intrusion-detection systems. The most interesting article that we have found is by Mueller and Shipley [Mueller & Shipley 2001]. It is the only comparative test of commercial intrusion-detection systems that we have found, where a number of intrusion-detection products are compared on equal footing and on live traffic, hence with a higher probability of either missing attacks or triggering false positives. The main drawback of this kind of test is reproductibility: when observing alerts it is quite difficult to reproduce the conditions in which these alerts are generated. Also, we believe that the tuning phase carried out during testing initiation is problematic; the testers describe a process in which alerts that they believe are false are turned off.

The most serious work related to Intrusion-Detection Systems / Intrusion-Prevention Systems testing is regularly published since 2004 by the NSS group (http://www.nss.co.uk). They regularly select a set of Intrusion-Detection Systems / Intrusion-Prevention Systems appliances and provide comparative testing of performance and detection capabilities. Their findings are a widely used source of information for Intrusion-Detection Systems / Intrusion-Prevention Systems buyers, but the rationale for selecting the tested appliances seems unclear. A complete description of their methodology is available on their web site.

Conclusion

The development of practical benchmark instances is a complex task as it needs to mitigate all the properties of a benchmark (e.g., representativeness, repeatability, portability, non-intrusiveness). Indeed, in practice, these properties are not independent. Some are correlated (e.g., portability and non-intrusiveness) while others are conflicting (representativeness and cost effectiveness). For example, a proper selection of the faultload relying on the so-called software-implemented fault injection technique (see e.g., [Carreira *et al.* 1998]), or via the concept of “failure acceleration” [Chillarege & Bowen 1989]), can contribute positively to

fulfil the cost effectiveness property, but at the risk of degrading representativeness. Work on these directions is still needed.

Finally, it is worth to mention that most of the advances made to date in the development of dependability benchmarking have concentrated on moderate size target systems. Only few and preliminary studies (e.g., see [Labovitz et al. 1999]) have addressed the assessment of large-scale infrastructures from a dependability viewpoint, but not from a benchmarking perspective. Accordingly, much research would be needed to better master the scalability issue with respect to large-scale systems and also with respect to all scalability properties considered in ReSIST.

4 – Diversity

Introduction

In this chapter we consider the role that diversity can play in achieving and evaluating dependability. We begin with a background section that gives an introduction to what we mean by diversity, and in particular how it differs from simpler forms of redundancy that have been used for many years.

Sections 4.1-4.3 consider the application of diversity to dependability achievement, and the particular problems this poses for evaluation. The earliest work concerned diversity in 'technical' systems - i.e., computer systems - where there have been extensive modelling advances in the last 20 years; this work is described in section 4.1. More recently, there has been a realisation that the notion of 'system' needs to be wider than simply 'computer system', in particular to include humans: section 4.2 examines research on human-computer diversity. Section 4.3 addresses work on the role of diversity in the processes used to build systems: firstly, the use of diversity of process to build non-redundant systems; secondly the use of process diversity to create diverse systems.

Section 4.4 examines a different, and new, application of diversity to dependability evaluation itself: the use of diverse methods of reasoning about dependability in order to have greater confidence in dependability claims about systems.

4.1. Background

We use the word "diversity", in this context, to indicate those forms of difference between components that provide redundancy for one another in a system, that reduce or are intended to reduce the likelihood of common failures of these components defeating redundancy and causing system failures. In computing, the study of diversity has meant mostly diversity of *design*. The intuitive rationale behind the use of design diversity is simply the age-old human belief that 'two heads are better than one'. For example, we are more likely to trust our answer to some complex arithmetic calculation if a colleague has arrived independently at the same answer. In this regard, Charles Babbage was probably the first person to advocate using two computers - although by computer he meant a person [Babbage 1837]. Much of the interest in using diversity in recent years has concerned protection against *software* failure; however, similar issues arise in other areas, for example when any kinds of *design* fault may be present; or when a computer monitors a human's performance (or vice versa).

People building hardware systems have known for a very long time that the reliability of a system can be increased if redundancy can be built into its design. Thus, when a component fails, if there is another component waiting to take over its task, such a failure can be masked. Indeed, if we were able to claim that

components failed independently of one another, we might be able to claim to make arbitrarily reliable systems from arbitrarily unreliable components. In practice, though, such statements are largely mathematical curiosities, since complete independence rarely, if ever, occurs in practice (and complex redundant structures bring their own, novel, forms of unreliability). It is this issue of dependence of failures, which were often assumed independent when modelling hardware fault-tolerant systems, that made modelling of *software* fault tolerance a particularly difficult, novel problem.

Of course, simply replicating a component (hardware or software) with one or more identical copies of itself will provide no guarantee against the effect of design faults, since these will themselves simply be replicated. If all copies are exposed to the same demands, whenever a demand triggers a design fault all versions will fail together. The natural defence against design faults in a component is thus to add different code, which may not be subject to the same faults.

In its simplest form, design diversity involves the 'independent'[1] creation of two or more versions of a program, which are all executed on each input reading so that an adjudication mechanism can produce a 'best' single output. Here the versions may be developed from the same specification, or the higher level (and usually more informal) engineering requirements may be used to produce diverse formal specifications. Typically, the teams building the versions work 'independently' of one another, without means of direct communication (indirect communication may still occur: for example one team may discover faults in the common specification, causing the project managers to issue a correction to all teams). The teams may be allowed complete freedom of choice in the development process, or they may have these imposed upon them in order to 'force' diversity (e.g. different algorithms, different programming languages). In the former case, the hope is that identical mistakes will be avoided by the natural, 'random' variation between people and their circumstances; in the latter, the same purpose is pursued by intentionally varying the circumstances and constraints under which the people work to solve the given problem.

A recent survey of fault-tolerant computing architectures using diversity is in [Strigini 2005]. Examples of simple fault-tolerant schemes for software are:

- pure *N-version* software, with multiple versions produced as we outlined above, and executed on the redundant processors of an N-modular redundant system;
- *recovery blocks*, in which one version is executed at a time, its failures are detected by an acceptance test and recovered via roll-back and retry with a different version of the software;
- *N-self checking* software, in which version pairs are used as self-checking components, which self-exclude when a discrepancy between the pair is detected: for instance, two such pairs form a redundant system able to tolerate the failure of anyone of the four versions.

More generally, some form of diversity is used against design faults in most well-built software, in the form of defensive programming, exception handling and so on. In other engineered systems, diversity in hardware components or principles of operation between redundant components is often required to protect against common failures caused not only by design faults, but by e.g., maintenance-caused faults or limitations of each individual technology (e.g., in sensors).

The importance of diversity for the "ubiquitous" systems that are the focus of ReSIST lies in at least the

[1] We use quotes here, because of the profligate way in which words such as 'independence' and 'independent' are used in writing about diversity and fault tolerance. Strictly, the only use of the terms which has a formal definition concerns statistical independence, for example here between the failure processes of two or more versions.

following factors:

- in large-scale systems, common failures can defeat the natural redundancy typical of these systems and may cause large-scale system failures;
- the limited dependability, and/or evidence thereof, of many off-the-shelf components on which these systems are built makes diversity a necessary remedy for many services to achieve their required QoS levels;
- "natural" diversity is often available through the interconnection of diverse large-scale systems (e.g. multiple communication networks), and evaluating the degree of protection - of resiliency - this offers to the services these ubiquitous systems provide, is essential.

Our knowledge about modelling diversity is mostly about modelling systems with very few diverse components: although our models may be suitable for exploring some aspects of ubiquitous systems (e.g., to what degree two diverse platforms in a network could be considered unlikely to succumb to the same attack), they need scaling up to address many other questions of interest.

Evaluation of resilience and thus dependability with respect to diversity covers several angles:

- direct experimental (measurement-based) evaluation of a specific system. Difficulties arise, as usual, if one wishes to estimate very small probabilities (e.g., probability of interruption in a very reliable service), or to estimate them for scenarios for which realistic measurement conditions are difficult to set up (e.g., estimating the probability of a novel worm propagating through diverse computing platforms on a network);
- probabilistic modelling, which can help both in evaluating a specific system and in evaluating or comparing alternative system configurations, development processes, etc. Here one can discriminate between:
  - evaluating system-level dependability figures, assuming as given some parameters characterising the "efficacy of diversity": e.g., the probability of two components failing together when subjected to the same demand, or the coverage of an error recovery mechanism. Given these parameters, the evaluation problem and methods are substantially the same as for any other system. This kind of modelling has been applied by various ReSIST partners (City, LAAS, Pisa), for small-scale systems [Arlat et al. 1990, Kanoun et al. 1993, Chiaradonna et al. 1994, Xu et al. 1995, Giandomenico et al. 1997, Bondavalli et al. 2002], and is in common use in probabilistic Safety Assessment for many kinds of engineered systems; similarly, researchers have applied to ICT networks epidemiological models, where the diversity among nodes can be described by parameters describing their susceptibility of diverse nodes to catch and transmit disease.
  - including in the evaluation problem the estimation of these crucial "diversity describing" parameters. This has been the focus of much research at City, mostly on very simple diverse-redundant systems with diversity.

Figure 1 shows the way in which it is generally believed, in an informal way, that design diversity works. At the top level there is 'process diversity', i.e., diversity in the design practices, in the personnel, etc., involved in the production of the versions. This diversity - people doing things differently - results in the versions themselves being 'different' from one another, and in particular, it is hoped, it is unlikely that faults in one version will be identical to those in another. These differences between versions in turn affect the chance that, if one version fails on a particular demand, another version will also fail: it is this diversity of failure

behaviour that is the goal we seek. We would like independence between the version failure processes or, even better, a negative dependence, so that the situations in which one version is especially likely to fail will be ones in which the other is especially unlikely to fail.

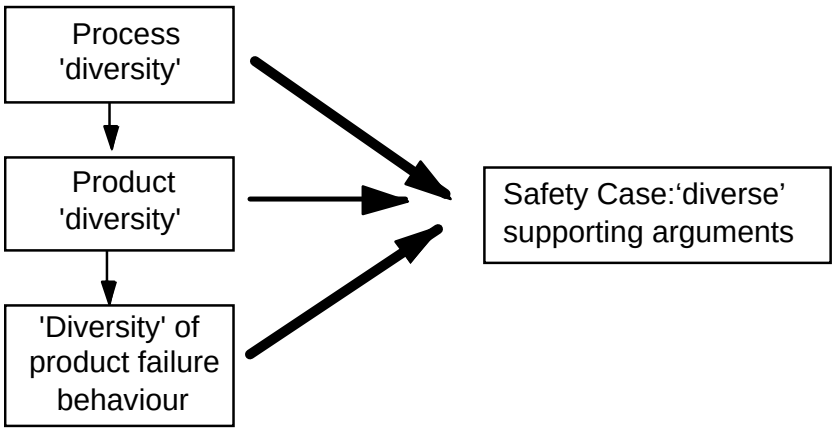

Figure 1: Different types of 'diversity' at different stages of the software design and development process

We note in passing that diversity can defend against malicious faults - attacks exploiting vulnerabilities, which may not be common to diverse components - as well as against accidental ones [Deswarte et al. 1999]. Modelling in the area of security is however at its beginning [Littlewood & Strigini 2004].

Claims for the reliability of a software-based system that utilises design diversity (or, indeed, any system) – may themselves benefit from the use of diversity. For example, they might involve evidence from all three stages shown in Figure 1. We shall briefly describe in a later section how the use of such diverse *evidence*, as part of a dependability case, can be used to increase confidence in a dependability claim.

4.2. Diverse parallel systems

4.2.1 Probability models based on 'difficulty'

In the early days of software fault tolerance, there was some controversy about whether independence of failures between versions was a reasonable goal, or even possible. This prompted theoretical and experimental studies of how dependence might arise. One explanation for the fact that people tend to make similar mistakes in certain circumstances is that some problems are intrinsically harder than others - i.e., that there may be some parts of a programming task that most people will find difficult. This intuition is at the heart of the first probability model that attempts to capture the nature of failure dependency [Eckhardt and Lee 1985]. This elegant and influential model is based on a notion of 'variation of difficulty' over the demand space: it is shown that the greater this variation, the greater the dependence in failure behaviour between versions (and thus the less benefit in a fault-tolerant system). In particular, it is shown that even versions that are *truly independently developed* (and there is a precise meaning given to this in the model) will fail dependently.

The key variable in the model, the *difficulty function*, $\theta(x)$, is defined to be the probability that a program chosen at random will fail on a particular demand, x. Here the program is chosen via the probability distribution over all programs. That is, if we were to select (i.e., build) very many programs independently in

this way, $\theta(x)$ would be the proportion of these that failed when presented with demand x. This seems a natural definition of the intuitive notion of 'difficulty': the more difficult a demand, the greater we would believe the chance that an unknown program will fail. The important point here is that difficulty varies across the demand space. Then the reason programs fail dependently is that a demand that is 'difficult' for one program will be difficult for another: more precisely, if you know that program *A* has failed, you believe that the demand is probably a difficult one, and thus that program *B* is more likely to fail (than you previously thought). The severity of the effect – and thus the extent of the departure from independence – turns out to depend upon the *variance* of the difficulty function.

Instead of merely allowing diversity to arise willy-nilly as in EL, another approach is to force diversity in the development processes of the different versions. Thus it might be insisted that the different teams use different programming languages, different testing regimes, etc. Such forced diversity has been used in industrial practice (e.g., the Airbus A320 flight control software [Traverse 1988; Briere & Traverse 1993] and in experiments (e.g. the DARTS project [Smith et al. 1991]).

The LM [Littlewood & Miller 1989] model generalises the EL model to take account of forced diversity by defining *different* distributions over the population of all programs. Thus a program will have a different probability of selection (i.e., being developed) under methodology *A* from that under methodology *B.* The effect of there being these different distributions over the programs is that there are different difficulty functions induced over the demand space. The degree of dependence of the failure processes of the versions now turns out to depend upon the association between these difficulty functions, i.e., on their *covariance.* Interestingly, it is *possible* within the model for this to be negative (i.e., roughly, what one program finds difficult, another finds easy, and vice-versa), in which case a pair of versions would exhibit better behaviour (less coincident failures) even than would be the case if they failed independently.

Even if such negative correlation is unlikely to be obtained in practice, the LM model goes some way to rescue design diversity from the pessimistic conclusions that arise from the EL model. Specifically, it seems reasonable to believe that the difficulty functions *A* and *B* will always be different. There will always be *some* differences between the programming teams, or the methods that they use, that are significant in determining the nature of the errors that they make. Even if, as seems likely, the difficulty functions are positively correlated, the expected reliability of a 1-out-of-2 system will always be greater under the LM model assumptions than under those of EL (keeping the version reliabilities fixed between the two cases). In this sense, EL can be seen as the most extremely pessimistic case within the more general LM model, and it is reasonable to think that it is unattainable.

In [Littlewood & Miller 1989] the authors go further than this and provide some further results for forced diversity. For example, they prove that for 1-out-of-n systems, when fewer than n methodologies are available, the best design is the one that uses all the methodologies and spreads them as evenly as possible: e.g., in the case where n=5 and there are three methodologies available, it is better to build a *AABBC* design than a *AAABC* one.

The approach used in these models - represented by the formalisation of notions like variation of difficulty and forced diversity - seems quite powerful and generally applicable. It has been applied by City University to many related problems of evaluation - of systems or processes - in the presence of diversity. It relies on modelling how the "stress" on redundant elements varies across the range of the demands and environmental factors they face, since this variation determines the probability of common failures. Depending on how precisely one knows factors like the conditional probability of failures of components on specific demands, and which assumptions of conditional independence hold, the models provide outputs that range from

general insight in how diversity affects system dependability to more precise indications for designing diverse systems and predictions of their reliability.

For example, in [Hughes 1987; Littlewood 1996], the EL and LM models are generalised to account for common mode failures in non-software based systems. Modelling design diversity is indeed just one case of the general problem of predicting the probability of common failure between redundant components, a difficult problem for all highly dependable systems. The approach casts doubt upon claims for failure independence even in systems with "functional" diversity [Littlewood et al. 1999].

On the other hand, a limitation of this approach should be emphasised: most of the results arising from the models concern averages - over demand spaces, over populations of programs, etc. Thus the 'difficulty function' in EL, the probability that a randomly chosen program will fail on *x*, is an average over all programs. The 'reliabilities' are averages over programs and demands. Actual realisations may differ from these averages. For example Knight and Leveson [Knight & Amman 1985] found their 2-out-of-3 systems an order of magnitude more reliable than the single versions *on average*, but there were *particular* single versions that were more reliable than *particular* 2-out-of-3 systems. In the absence of information about the particular - e.g., when taking an early design decision as to whether to use forced or unforced diversity - these average results give useful guidance. But they do not allow strong claims to be made for a particular system - e.g., a particular system based upon forced diversity. In other words, they do not absolve us of the responsibility to evaluate what has actually been achieved. Furthermore, the results all concern preferences - the theorems involve inequalities - and do not quantify the advantages of different approaches.

One of the active research areas is about selecting practical methods for using data from testing and operation for evaluating a specific system. One may consider an internally diverse-redundant system as a black box, measure the amount of use and events of interest (e.g., count failures) in realistic or real use, and apply textbook inference methods to estimate dependability measures. However, this throws away potentially useful information: we know the redundant structure of the system, the failures of any one channel are more frequent - easier to observe - than system failures; and, the component types that constitute the individual channels may have seen operational use before, bringing a record of their dependability in those uses. The problem is how to use these data, since the system failure probabilities depend both on those of the individual components and on their probabilities of common failure. These problems, and limited solutions under restrictive assumptions, have been reported in [Littlewood et al. 2002; Popov et al. 2003; Popov and Littlewood 2004].

4.2.2 Empirical studies

There was considerable experimental work on diversity in the 1980s. The best-known is probably the experiment by Knight and Leveson (see above). The original intention of the authors was to carry out a statistical test of the hypothesis that 'independently' developed versions failed independently. The experimental results allowed the authors to reject this hypothesis resoundingly: there was overwhelming evidence that simultaneous version failures were more likely than would be the case if they were failing independently. However, as we saw in the previous section, there were nevertheless benefits (at least on average) arising from the use of diversity.

All controlled experiments we are aware of produced similar results (see, e.g., [Eckhardt et al. 1991]): multiple-version systems were, on average, more reliable than individual versions, and sometimes much more so.

In recent years, City University has conducted empirical studies in two areas: exploring the possible gains in dependability and performance from using diverse off-the-shelf database servers; conducting statistical studies based on the massive numbers of diverse equivalent programs available from a world-wide programming contest.

This work has shown empirical evidence that current data replication solutions are insufficient to protect against the range of faults documented for database servers [Gashi et al. 2004-a, **Gashi, et al. 2006 (*in the Appendix for Part Arch)***]. In particular, we refuted the common assumption that fail-safe failures (i.e., crashes) are the main problem to be solved by database replication [Gashi et al. 2004-b]. This conclusion may cast serious doubts that the current state-of-the-art in database replication is adequate. Diverse redundancy is the only known technique which avoids this limitation, and we argued strongly in favor of using it for database replication. We have outlined possible fault-tolerant architectures using diverse servers and discussed the design problems involved.

We have also demonstrated empirically the potential for *performance* improvement through diverse redundancy [Popov et al. 2004, **Stankovic and Popov 2006 (*in the Appendix for Part Arch)***]. Diverse SQL servers exhibit systematic differences in their processing of SQL statements – server *A* executes some types of statements faster than server *B* while *B* is faster than *A* to execute other types of statements. Deploying diverse SQL servers in parallel thus allows for performance boost (e.g., when the fastest response is returned to the client) impossible to achieve with multiple replicas of the same SQL server. Performance gains achievable with diverse redundancy vary depending on the application profiles and can be significant for read-intensive applications.

Finally, we argued and have demonstrated empirically, that intelligent trade-offs in the form of service license agreements (SLAs) can be struck between dependability and performance of a fault-tolerant SQL server built with diverse SQL servers, impossible without diversity.

In other work, City has taken advantage of a remarkable source of *massive* program diversity. These data come from a programming contest run by the University of Valladolid, which has a website containing more than 1,500 specifications for programs. Anyone can write programs to these specifications and submit them over the internet. The website automatically tests every submission and gives feedback to the author whether it detected failures. In total, over fifty thousand authors have submitted more than three million programs. With this large resource of diverse programs, it is possible to perform empirical studies of the impact of a range of diversity techniques on program reliability, which despite the limits of using a special “population” like this, can provide useful examples and suggest conjectures about possible and non-obvious effects:

- [Bentley et al. 2004] used the large sample sizes of programs to estimate the ‘difficulty function’ that underpins the EL and successive models.
- [Meulen et al. 2004] measured the improvement from the use of a diverse 1-out-of-2 architecture. For unreliable programs, The average probability of failure of the pair was approximately the product of the probabilities of the individual programs (approximate independence), but the reliability improvement reached a “plateau” at about a factor of a hundred. This plateau has often been conjectured, but never observed before.
- [Meulen & Revilla 2005] examined the impact of diverse languages (C, C++ and Pascal) on the reliability improvement of a 1-out-of-2 pair by choosing different languages for the two programs in the pairs. C/Pascal pairs were significantly better, i.e., produced 1-out-of-2 systems that were, on average, more reliable.

- [Meulen et al. 2005] measured the effectiveness of run-time checks, demonstrating e.g., advantages from using diverse run-time checks.

4.3. Human-computer diversity

An important special case of diverse redundancy is that between people and machines: in many systems, components of each one of these two types are expected to detect errors (and/or recover from them or support their correction) arising in the other kind of component.

Evaluation here suffers from an additional difficulty: the statistics of the behaviour of human components vary, between individual instances (people who can fulfil a role in the system), between systems in which these components are deployed, and over time within each system, even more than those for inanimate components. Over time, in particular, people's degree of reliance on computer components, alertness to their failures, and ability to cope with them, all evolve interdependently with their subjective experience of the computer components' dependability. So, while the interplay between dependability of computer and human components can be modelled by models that would apply to computer-only [Strigini et al. 2003], diverse-redundant systems with the same architecture, quantitative modelling of human evolution when interacting with computers is an active research area[2] (cf Part ”socio” in this report).

4.4. Diversity in development to generate dependability

An important issue concerns the effect of diversity at the process level in producing useful diversity at the system level, i.e. diversity in failure behaviour. More generally, diverse redundancy among *tasks* to achieve dependability of any (even non-redundant) product is heavily used.

4.4.1 Diversity in developing non-redundant systems

In [Littlewood et al. 1998] we examine the efficacy of diverse *fault finding* techniques, i.e. where several different techniques are used together. We show that the effectiveness of such multi-technique approaches depends upon quite subtle interplay between their individual efficacies and *dependence* between them. We define measures of fault finding effectiveness, and of diversity, and show how these might be used to give guidance for the optimal application of different fault finding procedures to a particular program. We show that the effects upon reliability of repeated applications of a particular fault finding procedure are not statistically independent - in fact such an incorrect assumption of independence will always give results that are too optimistic. For *diverse* fault finding procedures, on the other hand, things are different: here it is possible for effectiveness to be even greater than it would be under an assumption of statistical independence. We show that diversity of fault finding procedures is, in a precisely defined way, ‘a good thing’, and should be applied as widely as possible. The modelling results are illustrated using some data from an experimental investigation into diverse fault finding on a railway signalling application.

Some authors maintain that open source software processes are particularly well-suited for delivering reliability because of their use of diversity in fault-finding – ‘given enough eyes, all bugs are shallow.’ In [Bosio et al. 2002] we discuss this kind of statement, first clarifying the different measures of reliability, and of a process's ability to deliver reliability, that can be of interest. We then propose a way of addressing some

[2] This is one of the topics of the INDEED project funded in the U.K. between the City University and the Universities of York, Edinburgh and St Andrews.

of the issues via probabilistic modelling. We present a model of the reliability growth that results from finding and fixing faults during software use, which takes account of the diversity of use patterns within the user community. We obtain preliminary, interesting, and non-intuitive results concerning the conjecture that a more diverse population of users engaged in reporting faults may give OSS processes an advantage over conventional industrial processes that employ less diversity. For example, we show that one should always prefer a diverse population of users, even in preference to a population of users like oneself (under quite plausible conditions of 'all things being equal', e.g. concerning equal total exposure in each case).

4.4.2 Process diversity to generate failure diversity

As for diversity between the processes that produce the diverse-redundant parts of a system, an attractive but elusive goal is to give explicit guidance, e.g. about how to combine measures intended to increase diversity, or how to manage trade-offs if measures that increase diversity also damage, as a side effect, the expected dependability of (some or all) the individual components. In this direction, recent work at City **[Salako & Strigini 2006]** has produced a generalisation of the EL/LM models to represent various possible forms of dependency between developments. These dependencies are visualised via Bayesian belief networks, and theorems have been found that characterise scenarios in which e.g., a specific added precaution in development (e.g., meant to improve "separation" between developments) is certainly beneficial for system dependability. This approach allows one to add rigour to the claims usually made informally for the merits (or weaknesses) of various ways of achieving "more effective diversity", by identifying hidden premises and formulating at least sufficient conditions under which such claims would be correct.

4.5. Diversity in arguments

We have seen that diversity can bring benefits when applied at the process and system levels. It turns out that it can also bring benefits when applied to the arguments used to support claims for dependability.

Rather informally, we take a dependability case to be: some *reasoning* based on *assumptions* and *evidence* that supports a dependability *claim* at a particular level of *confidence* (or *doubt*). Doubt about the correctness of a dependability claim can arise from several sources:

- the strength/weakness/relevance of the evidence;
- the extent of our confidence/doubt in the truth of the assumptions;
- the extent of our confidence/doubt in the correctness of the reasoning in the argument.

One way of looking at the problem is to think that cases can be faulty, just as systems can be faulty: this suggests that multiple diverse argument legs might increase confidence in the way that the use of diverse systems can improve reliability.

As might be expected from the systems analogy, issues of dependence between cases are important here. So the effectiveness of a two-legged case will depend on the effectiveness of each individual case *and on the dependence between these*. We have shown that, just as system failures cannot be assumed to be independent, so dependability case failures will not be independent. So far this work [Bloomfield & Littlewood 2003, **Littlewood & Wright 2006**] has concentrated on studying an idealized case involving two legs: a leg based on a statistical argument from operational testing, and a leg based on formal verification. In spite of the simplicity of this example, it has thrown up some surprisingly counter-intuitive results. For example, adding to an existing dependability case a second, entirely supportive, second case can sometimes result in less confidence in a claim than came from the original single case: i.e., extra 'good news' can

sometimes reduce confidence. The explanation lies in unexpected subtleties in reasoning that can sometimes arise when there is dependence between assumption 'doubts'. Less surprisingly, the association between doubts concerning assumptions for the two legs has a strong impact on the effectiveness of the diverse approach.

Conclusion

Research on evaluation problems concerning diversity has increased in recent years, following a fallow period after the burst of modelling and experimental work in the 1980s. By far the most work has addressed issues concerning diverse parallel 'technical' systems. Here there is now a good understanding, from sophisticated probability models, of some important underlying issues concerning 'dependence' (and by inference, 'independence') which determine the benefits that will be gained from diversity. This modelling work has been accompanied by empirical studies, but there has been nothing recently to match the large and expensive *controlled* experiments funded by NASA 20 years ago (although the recent work on a massive database of programs from a programming competition illustrates a different way of obtaining statistical results).

Modelling of human-computer diversity (and the application of the models to case studies) suggests that there may be some novel issues here that have not arisen in studies of software systems. For example, the effectiveness of a human-computer diverse system seems to depend upon a complex interaction between variation in problem 'difficulty' and variation in human 'ability' (to solve the problem).

Other recent work has examined *process* diversity: 'doing things differently'. Here again, there seem to be benefits, e.g. by using diverse testing methods to find faults in a program. Interestingly, the same (or very similar) probabilistic models as those used for system diversity seem to be useful in understanding the underlying principles here.

Guidance on how to *make* systems usefully diverse continues to be elusive. There are plausible intuitions here, but very little hard empirical evidence. We seem to be far from being able to 'design in' diversity on cost-effective ways.

Finally, work is beginning on ways of using diversity in the arguments used to justify claims about the dependability of systems (either single or diverse). An early expectation that existing diversity modelling approaches would be applicable here does not seem likely to be fulfilled, but slow progress is being made using some BBN models.

5 – Dependability Cases

Introduction

A dependability case is a documented body of evidence that aims to provide a convincing and valid argument that a system is adequately dependable for a given application in a given environment. A main problem in this context is the difficulty of reasoning with disparate evidence, and disparate techniques for manipulating it: proofs about algorithms and designs, statistical observations of system behaviour, documentation of procedures and cultures of the developer and user organizations, expert judgement, etc. In this Chapter, an overview of the challenges and current approaches for building cases to support dependability claims is provided, including in particular recent research results devoted to the development of quantitative approaches to dependability cases, where the confidence that can be placed in a dependability claim is expressed probabilistically. Whilst early work mainly concentrated upon cases to support claims about *reliability* and *safety*, there is a growing belief that the notations and tools developed there can also be effective in the security domain. Thus recent work has attempted to bring structure and formalisms to security assurance arguments, to improve their clarity and help elicit security requirements. This Chapter is devoted to safety and security cases for supporting dependability claims.

5.1. Safety cases

Assurance and assessment of dependability of software-based systems has long been acknowledged to be difficult [Littlewood 1991, Littlewood & Strigini 1993]. There are several reasons for this. Software is often novel, so that claims can rarely be based upon previous experience. Much of the evidence available concerns the software process – how it was built – and not the built product itself. There may be doubt about the truth of some of the assumptions that underpin the reasoning used to support a claim, e.g., that a test oracle is correct. There is a great reliance upon expert judgment – e.g., in how claims about the quality of the build process can be turned into claims about the delivered product's dependability.

These problems are particularly acute when systems are critical to some aspects of our lives. For this reason safety cases are increasingly mandated as a means of assuring critical systems.

Increasingly, regulatory agencies are making the case for a goal-based approach, in which claims (or goals) are made about the system, and arguments and evidence support these claims. This approach goes back more than a decade, and was heavily influenced by Toulmin's early work [Toulmin 1958] and by the contemporary perspective of proof as a social process [DeMillo et al. 1979, MacKenzie 2001]. The work of Toulmin underpins the Adelard goal-based approach, ASCAD [Bloomfield et al. 1998], together with

experience of developing and assessing safety cases from the 1980s (e.g., Sizewell B Primary Protection System and major hazard areas [Bishop & Bloomfield 1998]).

There is a close relationship between the Toulmin concepts and the Goal Structured Notation, GSN [Kelly and McDermid 1997]. Tools to support such notations have been developed and are essential if a graphical approach is used. One early development was the SAM safety argument manager [McDermid 1994]. SAM could present and edit safety justifications graphically using the GSN notation, and it also contained supporting tools (like fault tree analysis). The more recent ASCE Tool [Emmett and Cleland 2002] has a more open structure, supporting different notations, like ASCAD claims-argument-evidence and GSN, and allows direct links to external documents and tools.

In recent years there has been considerable interest in more quantitative approaches to dependability cases, where the confidence that can be placed in a dependability claim is expressed probabilistically. The idea here is that some reasoning, based on assumptions and evidence will support a (claim, confidence) pair (in fact, generally, an infinite number of such pairs). Confidence (or its complement, doubt) in the truth of a claim will depend on the extent of doubts about the truth of assumptions, or about the correctness of the reasoning, or about the strength of the evidence. Thus a case can be seen as a means of propagating the doubts about the truth of a claim that arise from many disparate sources.

Much of the difficulty here arises from the disparity of the evidence used: it may be statistical evidence from operational testing; logical evidence from verification and analysis; human expert judgement about the impact of the quality of development processes upon product dependability; and so on. CSR in recent years has been studying BBNs (Bayesian Belief Nets) as a formalism for dealing with these problems [Littlewood et al. 1995, Fenton et al. 1996, Neil et al. 1996, Delic et al. 1997, Courtois et al. 2000, **Littlewood & Wright 2006**].

BBNs are a powerful and attractive formalism. They facilitate a ‘divide and conquer’ approach to the complexity of the problem. Their graphical structure is a powerful aid to representing and thus understanding the many conditional independence assumptions implicit in this approach. Powerful tools such as Hugin (http://www.hugin.com/) allow automated computation of the doubt propagation.

However, the very power of the automated BBN approach can mask pitfalls for the unwary. Whilst the results will always be consistent in a probabilistic sense, their correctness and relevance to the problem will, of course, depend on the accuracy of the input data – i.e., the many conditional independence assumptions, and the (possibly very many) node probability table entries. It is well known that people find it hard to describe multi-dimensional uncertainty accurately and consistently. In recent work **[Littlewood & Wright 2006]**, City eschewed the use of the automated numerical approach to BBNs and instead investigated some simple dependability cases purely analytically, i.e., retaining a complete parametric mathematical model of all the uncertainty involved in the case. It has been shown that even in simple, idealised cases some surprising counter-intuitive results occur that might not be noticed in a purely numeric treatment. The current view is that BBNs are indeed a powerful tool, but the very seductiveness of the approach can bring unwarranted confidence in their output: they need to be used with respect and humility.

5.2 Security cases

Section 2.1 of this part (Security Evaluation Criteria) explained that the traditional approach to security assessment is embodied in the “*orange book*” [TCSEC 1983] and its successors, culminating in today’s

Common Criteria [CC 2006]. In each case, the requirement is to perform a checklist of evaluation activities, where the particular checklist depends on the security level claimed.

Common issues in the assurance of safety, security and reliability have recently been studied, for example in [Lautieri et al. 2004, Lautieri et al. 2005]. An early example of this approach is outlined below.

QinetiQ introduced a GSN-based approach for developing security cases and elucidating system security requirements. The general approach has three steps:

1) Identify credible mechanisms for intercepting communications-layer traffic and for compromising communications-layer security mechanisms. Characterise the practicality of these mechanisms.
2) Build a security case top-down, using GSN as a structuring aid. Choose low-level requirements to counter the practical attack mechanisms identified in step 1.
3) Study the GSN security case to identify gaps to be addressed, and hence elucidate security requirements.

The first step is generic to all applications built on this communications layer. The second and third steps are specific to the system being assessed.

Compared to the traditional approach to security assessment, this approach has several advantages. First, it supports decomposition of system security requirements to a component level, and so supports solutioneering and procurement from third parties. Second, it provides a notation for directly linking the assessment of assets requiring protection, the risk to those assets, and the resulting system information security requirements. Third, it uses an intuitive and scalable graphical notation. Fourth, it provides an efficient method for assessing the impact of environmental change on the requirements, solutions and assurance methods for the entire system.

Other examples presented in [Azeem et al 2004] and [Cole et al 2005] focused on Bluetooth traffic and Bluetooth security mechanisms. These reports apply the above approach in the context of security of Bluetooth-based applications in three scenarios: ‘Commercial media provision’, ‘Personal digital environments’, and ‘Remote monitoring of health in the home’. A summary of the approach was presented at a recent GSN User Club meeting [Pygott and Stuart 2006]. A similar approach was adopted by QinetiQ for a security review of Irish e-voting equipment.

Conclusion

There are several reasons why it is difficult to build convincing dependability cases for computer-based systems. As we have remarked earlier, one of these is that dependability claims are inevitably dependent upon very disparate evidence: this can range from informed expert judgment through to hard empirical evidence. Another is the ubiquity of *uncertainty* (e.g., in the ‘strength’ of evidence, in the correctness of assumptions and modes of reasoning). Understanding how such uncertainty propagates through a case is hard, particularly when we want to do this quantitatively via probabilities, which is necessary when the case is supporting some wider evaluation, e.g., for risk assessment. In spite of these difficulties, real progress is being made. Already, *safety* cases for computer-based systems are a common requirement in some industries. There are opportunities for applying the case-based approach to reliability evaluation (which is sometimes easier because the claim levels can be more modest). Some slow progress is being made in security cases,

where there are acute difficulties, particularly in obtaining empirical evidence so that there is a high dependence on expert judgement.

Conclusion

In this part, we summarized the current state of knowledge within the ReSIST network, and in the dependability community, concerning the techniques and methods that can be used to support the evaluation of resilience and to justify the level of resilience achieved.

As highlighted in the five chapters of this part, a large body of knowledge has been accumulated on several topics related to resilience evaluation, with a key role played by the ReSIST partners. However, applying the existing expertise to the resilience evaluation of ubiquitous systems is still a challenging task. In this conclusion, we summarize some open issues that need to be addressed to ensure the scalability of current techniques to ubiquitous systems.

State-based stochastic models are commonly used to support the evaluation of quantitative measures characterizing the dependability and resilience of computing systems. As can be seen from the discussion in Chapter 1, the increasing scale and complexity of modern computing systems continues to put a premium on efficient techniques for the construction and solution of large quantitative models. The development of hierarchical and composable modelling approaches that are well suited to take into account the specificities of ubiquitous systems is needed. In addition, these large, dynamic and evolving systems pose some new challenges that the ReSIST partners aim to address in their future research. In particular, evaluation methods must deal with metrics at an increasingly high level of abstraction, to express the impact of the computing infrastructure on an enterprise business. Of increased significance is also the use of quantitative evaluation methods to support the effective use of adaptation mechanisms prevalent in modern systems.

So far, the research on resilience evaluation of computing systems with respect to accidental faults and with respect to malicious faults, have generally followed distinct directions. However, the variety of threats to be encountered in ubiquitous systems implies a need for an integrated comprehensive framework taking into account accidental and malicious faults. Traditionally, only accidental faults in hardware and software have been taken into account in the evaluation of quantitative dependability measures. The evaluation of security has been mainly based on qualitative evaluation criteria. Such criteria are widely recognized to be insufficient for analysing and assessing the impact of malicious attacks and vulnerabilities on the security of systems in operation. Today, the development of efficient techniques and tools for analyzing malicious threats and evaluating their impact on the security of computing systems is necessary to enable the systems designers and administrators to improve the capacity of their systems in resisting to potential attacks. The definition of an evaluation approach based on probabilistic modelling is a promising research direction aimed at filling this gap. As reflected by the state of knowledge summary presented in Chapter 2, several advances have been achieved in this field during the last decade, in particular by ReSIST partners. However, research on this topic is still at a preliminary stage. There are several open issues that need to be addressed, in particular concerning: i) the definition of representative metrics and assumptions for quantifying security, and ii) the development of analysis and modelling methodologies allowing security evaluation to be used for driving design decisions or to support the development of efficient early warning and intrusion detection

systems that will enable the system administrators to react efficiently to attacks. Methodologies are also needed to carry out trade-off analyses between security mechanisms and ensuing quality-of-service penalties. In particular, to support self-managing systems, such trade-offs must be made increasingly frequent in an on-line, automated fashion. This requires techniques to deal explicitly and formally (that is, using quantitative metrics) with trade-off issues that would otherwise be based on intuition or subjective arguments. The development of such methodologies requires the definition of realistic assumptions about system vulnerabilities and attacker behaviours. Such assumptions could be derived based on the observations of real attacks using for instance widely deployed honeypots or intrusion detection systems.

Recently, a significant effort has been put on the development of dependability benchmarking approaches, that are aimed at evaluating dependability or performance-related measures, experimentally or based on experimentation and modelling, in a well-structured and standardized way. The development of practical benchmark instances is a complex task as it needs to mitigate all the properties of a benchmark (e.g., representativeness, repeatability, portability, non-intrusiveness). The development of dependability benchmarks is still at an early stage in spite of major efforts and progress made in recent years. With a few exceptions, most of the advances made to date have concentrated on moderate size target systems, taking into account accidental faults, mainly. Accordingly, much research would be needed to better master the scalability issue with respect to large-size systems and also to cover the variety of threats to be faced by ubiquitous systems, including malicious faults and accidental human-interaction faults.

The use of diversity in the design of ubiquitous systems is generally recommended to provide resilience against accidental as well malicious faults. Reliance on the diversity of the design requires appropriate techniques and models to measure and justify the level of diversity of the system, and to find the right trade-offs considering different and possible conflicting dependability properties. Research on evaluation problems concerning diversity has mainly addressed issues concerning diverse parallel ‘technical’ systems. Recently, the application of probabilistic quantitative models to address other forms of diversity such as human computer diversity, process diversity and diversity in arguments used in dependability cases, has been investigated. In particular, modelling of human-computer diversity is highly relevant in the context of ubiquitous systems as human interaction faults are reportedly a dominant factor in many critical systems and information infrastructures. Quantitative modelling of human-computer diversity is an active research area and brings some novel issues that have not arisen in studies of software systems, and need to be further explored in the future. Other important work related to diversity, is the study of process diversity and the quantification of the benefits that can be obtained by ‘doing things differently’, e.g. by using diverse testing methods to find faults in a program. Interestingly, the same (or very similar) probabilistic models as those used for system diversity seem to be useful in understanding the underlying principles here. So far only limited size case studies and examples have been investigated, and the generalization of the results to large-scale ubiquitous systems remains an open issue. Another original use of the diversity concept is its application to the arguments used to justify claims about the dependability of systems. This is particularly useful to support the development of dependability cases. An early expectation that existing diversity modelling approaches would be applicable in this context does not seem to be easily fulfilled, however work in this direction is only at a preliminary stage and significant progress is expected in the future.

The discussion on dependability cases is more general and raises the problem of how to build convincing arguments to justify the level of confidence to be placed in the dependability and resilience of the target system. There are several reasons why it is difficult to build convincing dependability cases for computer-based systems. One of these is that dependability claims are inevitably dependent upon very disparate evidence: this can range from informed expert judgment through to hard empirical evidence. Another is the ubiquity of uncertainty (e.g., in the ‘strength’ of evidence, in the correctness of assumptions and modes of

reasoning). Understanding how such uncertainty propagates through a case is hard, particularly when we want to perform this quantitatively via probabilities, which is necessary when the case is supporting some wider evaluation, e.g., for risk assessment. In spite of these difficulties, real progress is being made. However, most of the research carried out in this are has focussed on safety cases for computer-based systems, which are now a common requirement in some industries. However, the extension of the case approach to security has been considered only recently and more investigations are needed in this area.

References

[Abdelallah et al. 2002] H. Abdelallah Elhadj, H. M. Khelalfa and H. M. Kortebi, "An experimental sniffer detector: SnifferWall", Sécurité des Communications sur Internet Workshop (SECI'02), Tunisia, Sept. 2002.

[Alata et al. 2006] E. Alata, V. Nicomette, M. Kaaniche and M. Dacier, "Lessons learned from the deployment of a high-interaction honeypot", LAAS_Report 06-331, April 2006. To appear in Proc. Sixth European Dependable Computing Conference (EDCC-6), Coimbra, Portugal, October 18-20, 2006.

[Albinet et al. 2004] A. Albinet, J. Arlat and J.-C. Fabre, "Characterization of the Impact of Faulty Drivers on the Robustness of the Linux Kernel", in Proc. IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN-2004), (Florence, Italy), pp.867-876, IEEE CS Press, 2004.

[Albinet et al. 2007] A. Albinet, J. Arlat and J.-C. Fabre, "Robustness of the Device Driver-Kernel Interface: Application to the Linux Kernel", LAAS_Report 06-351, May 2006. To appear in *Dependability Benchmarking of Computer Systems*, (K. Kanoun and L. Spainhower, Eds.), IEEE CS Press, 2007.

[Alessandri 2000] D. Alessandri, "Using Rule-Based Activity Descriptions to Evaluate Intrusion-Detection Systems", H. Debar, L. Mé, S. F. Wu (Eds.): Recent Advances in Intrusion Detection, Third International Workshop, RAID 2000, Toulouse, France, October 2-4, 2000, Proceedings. Lecture Notes in Computer Science 1907 Springer 2000, ISBN 3-540-41085-6: pp. 183-196.

[Ammar & Rezaul Islam 1989] H. H. Ammar, and S. M. Rezaul Islam, "Time scale decomposition of a class of generalized stochastic Petri net models", in IEEE Transactions on Software Engineering, 15(6), pp. 809-820, June 1989.

[Arlat et al. 1990] J. Arlat, K. Kanoun, J.-C. Laprie "Dependability modelling and evaluation of software fault-tolerant systems", IEEE Transactions on Computers, Special Issue on Fault-Tolerant Computing, 39(4): 504-13, 1990.

[Arlat et al. 2002] J. Arlat, J.-C. Fabre, M. Rodríguez and F. Salles, "Dependability of COTS Microkernel-Based Systems", IEEE Transactions on Computers, 51 (2), pp.138-163, February 2002.

[Arlat et al. 2003] J. Arlat, Y. Crouzet, J. Karlsson, P. Folkesson, E. Fuchs and G. H. Leber, "Comparison of Physical and Software-Implemented Fault Injection Techniques", IEEE Transactions on Computers, 52 (8), pp.1115-1133, August 2003.

[Avizienis et al. 2004] A. Avizienis, Jean-Claude Laprie, B. Randell and C. Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing", IEEE Transactions on Dependable and Secure Computing, 1 (1), pp.11-33, Jan.-March 2004.

[Azeem et al 2004] K. Azeem et al, "Analytical Assessment of Bluetooth Security Mechanisms", January 2004, 45 p., http://www.forward-project.org.uk/PDF_Files/D4.pdf

[Babbage 1837] C. Babbage, "On the Mathematical Powers of the Calculating Engine" (Unpublished manuscript, December 1837). The Origins of Digital Computers: Selected Papers. B. Randell, Springer: 17-52, 1994.

[Bailey & Davidson 2003] M. W. Bailey and J. W. Davidson, "Automatic Detection and Diagnosis of faults in Generated Code for Procedure Calls", IEEE Transactions on Software Engineering, 29 (11), pp.1031-1042, November 2003.

[Balakrishnam & Trivedi 1995] M. Balakrishnam and K. S. Trivedi, "Component-wise Decomposition for an Efficient Reliability Computation of Systems with Repairable Components", in 25th International Symp. on Fault-Tolerant Computing (FTCS-25), (Pasadena, CA, USA), pp.259-68, IEEE Computer Society Press, 1995.

[Balbo et al. 1988] G. Balbo, S. C. Bruell and S. Ghanta, "Combining Queuing Networks and GSPNs for the Solution of Complex Models of System Behaviour", IEEE Trans. on Computers, 37, pp.1251-68, 1988.

[Bartlett & Spainhower 2004] W. Bartlett and L. Spainhower, "Commercial Fault Tolerance: A Tale of Two Systems", IEEE Transactions on Dependable and Secure Computing, 1 (1), pp.87-96, Jan.-March 2004.

[Barton et al. 1990] J. H. Barton, E. W. Czeck, Z. Z. Segall and D. P. Siewiorek, "Fault Injection Experiments Using FIAT", IEEE Transactions on Computers, 39 (4), pp.575-582, April 1990.

[Bellovin 1992] S. Bellovin, "There Be Dragons", Proc. of the Third Usenix Security Symposium, Baltimore MD. September 1992, pp. 1-16.

[Bentley et al. 2004] J.G.W. Bentley, P.G. Bishop, M. van der Meulen, "An Empirical Exploration of the Difficulty Function", Safecomp, (Potsdam, Germany), LNCS 3219, Springer, pp 60-71, 2004.

[Bernardi 2003] S. Bernardi, "Building Stochastic Petri Net Models for the Verification of Complex Software Systems", PhD Dissertation of the Department of Computer Science of the University of Torino, 2003.

[Bernardi & Donatelli 2004] S. Bernardi and S. Donatelli: Stochastic Petri Nets and Inheritance for Dependability Modelling. 10th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC 2004), Papeete, Tahiti, pp. 363-372, 2004.

[Betous-Almeida & Kanoun 2004-a] C. Betous-Almeida and K. Kanoun, "Construction and Stepwise Refinement of Dependability Models", Performance Evaluation, 56 (2004), Elsevier, pp.277-306, 2004.

[Betous-Almeida & Kanoun 2004-b] C. Betous-Almeida and K. Kanoun, "Dependability modelling of Intsrumentation and Control Systems: A Comparison of Competing Architectures", Safety Science, 42 (2004), Elsevier, pp.457-80, 2004.

[Bishop & Bloomfield 1998] P. G. Bishop and R. E. Bloomfield, "A methodology for safety case development", Safety-Critical Systems Symposium, Birmingham, UK, 1998.

[Bloomfield et al. 1998] R. E. Bloomfield, P. G. Bishop, et al. (1998). ASCAD - Adelard Safety Case Development Manual, Adelard, 1998.

[Bloomfield & Littlewood 2003] R. E. Bloomfield and B. Littlewood, “Multi-legged arguments: the impact of diversity upon confidence in dependability arguments”, International Conference on Dependable Systems and Networks (DSN), San Francisco, IEEE Computer Society, pp. 25-34, 2003.

[Bobbio & Trivedi 1986] A. Bobbio and K. S. Trivedi, “An Aggregation Technique for the Transient Analysis of Stiff Markov Chains”, IEEE Trans. on Computers, C-35 (9), pp.803-14, 1986.

[Boehme and Kataria, 2006] R. Boehme and G. Kataria, "Models and Measures for Correlation in Cyber-Insurance," 5th Workshop on Economics of Information Security, Cambridge, UK, 2006.

[Bondavalli et al. 1999] A. Bondavalli, I. Mura and K. S. Trivedi, “Dependability Modelling and Sensitivity Analysis of Scheduled Maintenance Systems”, in 3rd European Dependable Computing Conference (EDCC-3), (A. Pataricza, J. Hlavicka and E. Maehle, Eds.), (Prague, Czech Republic), pp.7-23, Springer, 1999.

[Bondavalli et al. 2001a] A. Bondavalli, M. Dal Cin, D. Latella, I. Majzik, A. Pataricza and G. Savoia “Dependability Analysis in the Early Phases of UML Based System Design”, Journal of Computer Systems Science and Engineering 16(5): 265-275, 2001.

[Bondavalli et al. 2001b] A. Bondavalli, M. Nelli, L. Simoncini and G. Mongardi, “Hierarchical Modelling of Complex Control Systems: Dependability Analysis of a Railway Interlocking”, Journal of Computer Systems Science and Engineering 16(4): 249-261, 2001.

[Bondavalli et al. 2002] A. Bondavalli, S. Chiaradonna, F. Di Giandomenico, and J. Xu, “An adaptive approach to achieving hardware and software fault tolerance in a distributed computing environment”, JSA - Journal on Systems and Architectures, 47(9):763-781, 2002.

[Bondavalli and Filippini 2004] A. Bondavalli and R. Filippini "Modeling and Analysis of a Scheduled Maintenance System: a DSPN Approach," The Computer Journal 47(6): 634-650.2004.

[Bosio et al. 2002] D. Bosio, B. Littlewood, M.J. Newby and L. Strigini, “Advantages of open source processes for reliability: clarifying the issues”, Open Source Software Development Workshop, University of Newcastle upon Tyne, 2002.

[Briere & Traverse 1993] D. Briere and P. Traverse “Airbus A320/A330/A340 Electrical Flight Controls - A Family Of Fault-Tolerant Systems” 23rd International Symposium on Fault-Tolerant Computing (FTCS-23), Toulouse, France, 22 - 24, IEEE Computer Society, 1993.

[Brocklehusrt et al. 1994] S. Brocklehurst, T. Olovsson, B. Littlewood, E. Jonsson, “On Measurement of Operational Security”, Proc. 9th Annual IEEE Conference on Computer Assurance (COMPASS'94), Gaithersburg, USA, June 29-July 1, 1994. ISBN 0- 7803-1855-2. IEEE Aerospace and Electronics Magazine, vol 9, no 10, October 1994.

[Brown et al. 2002] A. B. Brown, L. C. Chung and D. A. Patterson, “Including the Human Factor in Dependability Benchmarks”, in Supplemental Volume of the IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN-2002) - Workshop on Dependability Benchmarking, (Washington, DC, USA), pp.F.9-F.14, 2002.

[Buchacker & Sieh 2002] K. Buchacker and V. Sieh, “UMLinux — A Versatile SWIFI Tool”, in Proc. 4th European Dependable Computing Conference (EDCC-4), (Toulouse, France), pp.159-171, Springer, 2002.

[Buchacker et al. 2003] K. Buchacker, M. Dal Cin, H. J. Höxer, R. Karch, V. Sieh and O. Tschäche, "Reproducible Dependability Benchmarking Experiments Based on Unambiguous Benchmark Setup Descriptions", in Proc. IEEE/IFIP Int. Conference on Dependable Systems and Networks (DSN-2003), (San Francisco, CA, USA), pp.469-478, IEEE CS Press, 2003.

[Buchholz 1994] P. Buchholz, "Exact and Ordinary Lumpability in finite Markov Chains", Journal of Applied Probabilities, 31, pp.59-74, 1994.

[Buchholz 1995] P. Buchholz, "Hierarchical Markovian Models: Symmetries and Reduction", in Performance Evaluation, 22 (1), pp.93-110, 1995.

[Buchholz et al. 2000] P. Buchholz, G. Ciardo, S. Donatelli and P. Kemper "Kronecker Operations and Sparse Matrices with Applications to the Solution of Markov Models", in INFORMS Journal on Computing, 12 (3), pp.203-222, 2000.

[CAIDA a] CAIDA Project – The UCSD Network Telescope home page: http://www.caida.org/

[CAIDA b] CAIDA Project, "The Spread of the Witty Worm", available at: http://www.caida.org/analysis/security/witty/

[Carreira et al. 1998] J. Carreira, H. Madeira and J. G. Silva, "Xception: A Technique for the Experimental Evaluation of Dependability in Modern Computers", IEEE Transactions on Software Engineering, 24 (2), pp.125-136, February 1998.

[CC 2006] Common Criteria Portal Home page, available online at http://www.commoncriteriaportal.org/, last visited in May 2006.

[Charette 2005] R. N. Charette, " Why Software Fails", IEEE Spectrum September 2005.

[Chen et al. 2002] D. Chen, D. Selvamuthu, D. Chen, L. Li, R. R. Some, A. P. Nikora, and K. Trivedi, "Reliability and Availability Analysis for the JPL Remote exploration and Experimentation System", in Int Conf. on Dependable Systems and Networks (DSN-02), (Washington DC, USA), pp.337-44, IEEE Computer Society Press, 2002.

[Chen et al. 2003] Z. Chen, L. Gao, K. Kwiat. "Modeling the Spread of Active Worms". In Proc. of the IEEE INFOCOM, San Francisco, CA, USA, 2003.

[Chen et al. 2006] Chen, Pei-Yu, Kataria, Gaurav and Krishnan, Ramayya, "On Software Diversification, Correlated failures and Risk Management". Available at SSRN: http://ssrn.com/abstract=906481

[Cheswick 1992] B. Cheswick, "An evening with Berferd in which a cracker is lured, endured and studied", *Proc Winter USENIX Conference*, San Francisco, January, 1992.

[Chiaradonna et al. 1994] S. Chiaradonna, A. Bondavalli, and L. Strigini, "On performability modeling and evaluation of software fault tolerance structures", 1st European Dependable Computing Conference (EDCC-1), pages 97-114, Berlin, Germany, 1994. Springer-Verlag, 1994.

[Chillarege & Bowen 1989] R. Chillarege and N. S. Bowen, "Understanding Large System Failures — A Fault Injection Experiment", in Proc. 19th Int. Symp. on Fault-Tolerant Computing (FTCS-19), (Chicago, IL, USA), pp.356-363, IEEE CS Press, 1989.

[Christmansson & Chillarege 1996] J. Christmansson and R. Chillarege, "Generation of an Error Set that Emulates Software Faults Based on Field Data", in Proc. 26th Int. Symp. on Fault-Tolerant Computing (FTCS-26), (Sendai, Japan), pp.304-313, IEEE CS Press, 1996.

[Christmansson et al. 1998] J. Christmansson, M. Hiller and M. Rimén, "An Experimental Comparison of Fault and Error Injection", in Proc. 9th Int. Symp. on Software Reliability Engineering (ISSRE'98), (Paderborn, Germany), pp.369-378, IEEE CS Press, 1998.

[Ciardo et al. 2001] G. Ciardo, G. Lüttgen, and R. Siminiceanu, "Saturation: An Efficient Iteration Strategy for Symbolic State-Space Generation", in 20th Int. Conference on Tools and Algorithms for the Construction and Analysis of Systems and Theory of Petri Nets, pp.328-42, 2001.

[Ciardo & Miner 1999a] G. Ciardo and A. Miner, "Efficient Reachability Set Generation and Storage Using Decision Diagrams", in 20th Int. Workshop on Applications and Theory of Petri Nets, pp.6-25, IEEE Computer Society Press, 1999.

[Ciardo & Miner 1999b] G. Ciardo and A. Miner, "A Data Structure for the Efficient Kronecker Solution of GSPNs", in 8th Int. Workshop on Petri Nets and Performance Models, (Zaragoza, Spain), pp.22-31, IEEE Computer Society Press, 1999.

[Ciardo & Trivedi 1993] G. Ciardo and K. S. Trivedi, "Decomposition Approach to Stochastic Reward Net Models", Performance Evaluation, 18 (1), pp.37-59, 1993.

[Cole et al 2005] A. Cole et al, "An investigation into System Security Requirements for Next Wave Information Provision Services", March 2005, http://www.forward-project.org.uk/PDF_Files/D22.pdf

[Collins et al. 2006] M. Collins, C. Gates and G. Kataria, "A Model for Opportunistic Network Exploits: The Case of P2P Worms", 5th Workshop on Economics of Information Security, Cambridge, UK, 2006.

[Constantinescu 2005] C. Constantinescu, "Dependability Benchmarking Using Environmental Test Tools", in Proc. Annual Reliability and Maintainability Symp. (RAMS'05), (Alexandria, VA, USA), pp.567-571, IEEE Computer Society Press, 2005.

[Cooke et al. 2005] E.Cooke, M. Bailey, D. Watson, F. Jahanian, J. Nazario. "The Internet Motion Sensor: a distributed blackhole monitoring system". Technical Report Version. In Proc. of the 12th Network and Distributed System Security Symposium (NDSS), San Diego, CA, February 2005.

[Courtois et al. 2000] P.-J. Courtois, B. Littlewood, L. Strigini, D. Wright, N. Fenton, and M. Neil, "Bayesian Belief Networks for Safety Assessment of Computer-based Systems, in System Performance Evaluation: Methodologies and Applications", (E. Gelenbe, Ed.), pp. 349-363, CRC Press, 2000.

[Dacier et al. 1993] M. Dacier, M. Kaâniche, Y. Deswarte, "A Framework for Security Assessment of Insecure Systems", First Year Report of the ESPRIT Basic Research Action 6362: Predictably Dependable Computing Systems (PDCS2), September 1993, pp. 561-578.

[Dacier et al. 1994] M. Dacier, Y. Deswarte "Privilege Graph: an Extension to the Typed Access Matrix Model", Lecture Notes in Computer Science, Springer Verlag, vol. 875, pp. 319-334, November 1994 (Proc. of Esorics'94, November 1994, Brighton, UK).

[Dacier et al. 1996] M. Dacier, Y. Deswarte, M. Kaâniche, “Models and Tools for Quantitative Assessment of Operational Security”, in 12th IFIP Information Systems Security Conference (IFIP/SEC'96), (S. K. Katsikas, D. Gritzalis, Eds.), Samos, Greece, May 21-24, pp.177-186, ISBN 0-412-78120-4, Chapman & Hall, 1996.

[Daly 2001] D. Daly, “Analysis of connection as a decomposition technique”, PhD Thesis, Univ. of Illinois at Urbana Champaign, Illinois, 2001.

[Daly et al. 2001] D. Daly and W. H. Sanders “A connection formalism for the solution of large and stiff models”, in Proc. 34th Annual Simulation Symposium, pp. 258-265, April 2001.

[Debar et al. 1998] H. Debar, M. Dacier and A. Wespi, “Reference Audit Information Generation for Intrusion Detection Systems”, Proceedings of IFIPSEC'98, Vienna, Austria and Budapest, Hungaria, August 31--September 4, 1998, pages 405-417.

[DeBrunner & Gray 1992] L. S. DeBrunner and F. G. Gray, “A Methodology for Comparing Faut Tolerant Computers”, in Proc. 26th Asilomar Conf. on Signals, Systems and Computers, (Asilomar, CA, USA), pp.999-1003, IEEE Computer Society Press, 1992.

[Delic et al. 1997] Delic, K. A., F. Mazzanti, L. Strigini, “Formalising engineering judgement on software dependability via Belief Networks”, DCCA-6, Sixth IFIP International Working Conference on Dependable Computing for Critical Applications, Garmisch-Partenkirchen, pp. 291-305, 1997.

[DeMillo et al. 1979] R. A. DeMillo, R. J. Lipton, et al., “Social processes and proofs of theorems and programs.” Comm ACM 22(5): 271-280, 1979.

[Derisavi et al. 2003] S. Derisavi, H. Hermanns, and W. H. Sanders “Optimal State space lumping in Markov Chains”, Information Processing Letters, 87 (6), pp.309-15, 2003.

[Derisavi et al. 2005] S. Derisavi, P. Buchholz, and W. H. Sanders “Lumping Matrix Diagram Representations of Markov Models”, in International Conference on Dependable Systems and Networks (DSN-2005), (Yokohoma, Japan), pp.742-51, 2005.

[Deswarte et al. 1999] Y. Deswarte, J.C. Laprie, K. Kanoun, “Diversity against accidental and deliberate faults”, in Computer Security, Dependability & Assurance: from needs to solutions, IEEE Computer Society, Eds. P. Ammann, B.H.Barnes, S. Jajodia, E.H. Sibley (Eds.), ISBN-0-7695-0337-3, 1999, pp.171-181.

[Di Giandomenico et al. 1997] F. Di Giandomenico, A. Bondavalli, J. Xu, and S. Chiaradonna, “Hardware and software fault tolerance: Definition and evaluation of adaptive architectures in a distributed computing environment”, In Int. Conference on Safety and Reliability (ESREL'97), pages 341-348, Lisbon, Portugal, June 17-20. Pergamon Press, 1997.

[DSHIELD] DShield Distributed Intrusion Detection System home page: http://www.dshield.org.

[Durães & Madeira 2002] J. Durães and H. Madeira, “Emulation of Software Faults by Selective Mutations at Machine-code Level”, in Proc. 13th Int. Symp. on Software Reliability Engineering (ISSRE-2002), (Annapolis, MD, USA), pp.329-340, IEEE CS Press, 2002.

[Durães & Madeira 2003] J. Durães and H. Madeira, “Mutidimensional Characterization of the Impact of Faulty Drivers on the Operating Systems Behavior”, IEICE Transactions on Information and Systems, E86-D (12), pp.2563-2570, December 2003.

[Durães et al. 2004] J. Durães, M. Vieira and H. Madeira, “Dependability Benchmarking for Web-Servers”, in Proc. 23rd International Conference on Computer Safety, Reliability and Security (SAFECOMP-2004), (Postdam, Germany), 2004.

[Eckhardt & Lee 1985] D. E. Eckhardt and L. D. Lee "A Theoretical Basis of Multiversion Software Subject to Coincident Errors." IEEE Trans. on Software Engineering 11: 1511-1517, 1985.

[Eckhardt et al. 1991] D. E. Eckhardt, A. K. Caglayan, et al. "An experimental evaluation of software redundancy as a strategy for improving reliability." IEEE Trans Software Eng 17(7): 692-702, 1991.

[Edwards & Matassa 2002] D. Edwards and L. Matassa, “An Approach to Injecting Faults into Hardened Software”, in Proc. Ottawa Linux Symposium, (Ottawa, ON, Canada), pp.146-175, 2002.

[Emmett & Cleland 2002] L. Emmett and G. Cleland, “Graphical notations, narratives and persuasion: a pliant systems approach to hypertext tool design”. ACM Hypertext 2002 (HT02), College Park, Maryland, 2002.

[Fenton et al. 1996] N. Fenton, B. Littlewood, M. Neil, “Applying Bayesian belief networks in systems dependability assessment”. Proc. Safety Critical Systems Symposium, Leeds, Springer-Verlag, pp. 71-94, 1996.

[Fenton & Ohlsson 2000] Norman E. Fenton and N. Ohlsson, “Quantitative Analysis of Faults and Failures in a Complex Software System”, IEEE Transactions on Software Engineering, 26 (8), pp.797-814, August 2000.

[FSECURE] F-Secure Corporation. "Deloder Worm Analysis". Available at: http://www.f-secure.com

[Gashi et al. 2004 a] I. Gashi, P. Popov, V. Stankovic, L. Strigini, “On Designing Dependable Services with Diverse Off-The-Shelf SQL Servers”, in “Architecting Dependable Systems II”. (R. de Lemos, C. Gacek and A. Romanovsky, Eds.), LNCS 3069, Springer, pp.191-214, 2004.

[Gashi et al. 2004 b] I. Gashi, P. Popov, L. Strigini, “Fault diversity among off-the-shelf SQL database servers”, Int. Conf. on Dependable Systems and Networks (DSN '04), (Florence, Italy), IEEE Computer Society, pp 389-398, 2004.

[Gashi et al. 2006] I. Gashi, P. Popov, and L. Strigini, “Fault Tolerance via diversity for Off-the-Shelf Products: a Study with SQL Database Servers”, London, Centre for Software Reliability, 2006.
(*in the Appendix for Part Arch)*

[Gönczy et al. 2006] L. Gönczy, S. Chiaradonna, F. Di Giandomenico, A. Pataricza, A. Bondavalli, and T. Bartha, “Dependability evaluation of web service-based processes”. In Proc. of European Performance Engineering Workshop (EPEW 2006), LNCS Vol. 4054, pp. 166-180, Springer, 2006.

[Gray 1993] J. Gray (Ed.), The Benchmark Handbook for Database and Transaction Processing Systems, Morgan Kaufmann Publishers, San Francisco, CA, USA, 1993.

[Grundschober 1998] S. Grundschober, “Sniffer Detector Report”, Master Thesis, Eurecom Institute, June 1998, 50 pages, ref. Eurecom: CE-98/IBM/GRUN - Document number: 1914.

[Grundschober and Dacier 1998] S. Grundschober, M. Dacier, “Design and Implementation of a Sniffer Detector”, Recent Advances on Intrusion Detection Workshop (RAID98), 1998.

[Gu et al. 2003] W. Gu, Z. Kalbarczyk, R. K. Iyer and Z. Yang, "Characterization of Linux Kernel Behavior under Errors", in Proc. Int. Conf. on Dependable Systems and Networks (DSN-2003), (San Francisco, CA, USA), pp.459-468, IEEE CS Press, 2003.

[Hughes 1987] R. P. Hughes, "A new approach to common cause failure", Reliability Engineering 17: 211-236, 1987.

[ITSEC 1991] "Information Technology Security Evaluation Criteria (ITSEC), Harmonizes Criteria of France, Germany, the Netherlands, the United-Kingdom, Commission of the European Communities, 1991.

[Jarboui et al. 2003] T. Jarboui, J. Arlat, Y. Crouzet, K. Kanoun and T. Marteau, "Impact of Internal and External Software Faults on the Linux Kernel", IEICE Transactions on Information and Systems, E86-D (12), pp.2571-2578, December 2003.

[Jarvis et al. 2004] S. Jarvis, N. Thomas and A. van Moorsel, "Open Issues in Grid Performability," international Journal of Simulation, UK Simulation Society, Vol. 5, Issue 5, pp. 3—12, 2004.

[Jonsson and Olovsson 1997] E. Jonsson and T. Olovsson, "A Quantitative Model of the Security Intrusion Process Based on Attacker Behavior," IEEE Trans. Software Eng., vol. 23, no. 4, pp. 235-245, Apr. 1997.

[Kaâniche et al. 2003-a] M. Kaâniche, K. Kanoun and M. Rabah, "Multi-level modeling approach for the Availability assessment of e-business applications", Software: Practice and Experience, John Wiley & sons, pp. 1323-41, 2003

[Kaâniche et al. 2003-b] M. Kaâniche, K. Kanoun and M. Martinello, "A User-Perceived Availability Evaluation of a Web Based Travel Agency", 2003 International Conference on Dependable Systems and Networks (DSN-2003), pp. 709-18, 2003.

[Kaâniche et al. 2006] M. Kaâniche, E. Alata, V. Nicomette; Y.Deswarte, M. Dacier, "Empirical analysis and statistical modeling of attack processes based on honeypots" WEEDS 2006 - workshop on empirical evaluation of dependability and security (in conjunction with the international conference on dependable systems and networks, (DSN2006), Philadelphia (USA), June 25 - 28, 2006, pp. 119-124.

[Kalakech et al. 2004] Kalakech A., K. Kanoun, Y. Crouzet and A. Arlat, Benchmarking the Dependability of Windows NT, 2000 and XP, Int. Conf. on Dependable Systems and Networks, Florence, Italy, pp. 681-686, 2004

[Kalyanakrishnam et al. 1999] Kalyanakrishnam M., Iyer R. K. and Patel J. U., "Reliability of Internet Hosts: a Case Study from the End User's Perspective", Computer Networks, 31, pp.47-57, 1999.

[Kanawati et al. 1995] G. A. Kanawati, N. A. Kanawati and J. A. Abraham, "FERRARI: A Flexible Software-Based Fault and Error Injection System", IEEE Transactions on Computers, 44 (2), pp.248-260, February 1995.

[Kanoun et al. 1993] K. Kanoun, M. Kaâniche, C. Béounes, J.C. Laprie, and J. Arlat, "Reliability growth of fault-tolerant software", IEEE Transactions on Reliability, IEEE Computer Society, 42(2), pp.205-19, 1985.

[Kanoun et al. 1999] K. Kanoun, M. Borrel, T. Moreteveille and A. Peytavin, "Modeling the Dependability of CAUTRA, a Subset of the French Air Traffic Control System", IEEE Transactions on Computers, 48 (5), pp.528-35, 1999.

[Kanoun & Borrel 2000] K. Kanoun and M. Borrel, "Dependability of Fault-tolerant Systems — Explicit Modeling of the Interactions Between Hardware and Software Components", in IEEE Transactions on Reliability, Vol.49, N°4, pp.363-376, 2000.

[Kanoun et al. 2002] K. Kanoun, H. Madeira and J. Arlat, "A Framework for Dependability Benchmarking", in Supplemental Volume of the 2002 Int. Conf. on Dependable Systems and Networks (DSN-2002) - Workshop on Dependability Benchmarking, (Washington, DC, USA), pp.F.7-F.8, 2002, see also http://www.laas.fr/DBench.

[Kanoun et al. 2005] K. Kanoun, Y. Crouzet, A. Kalakech, A.-E. Rugina and Ph. Rumeau, "Benchmarking the Dependability of Windows and Linux using PostMark Workloads, 16th IEEE International Symposium on Software Reliability Engineering (ISSRE'2005), Chicago (USA), pp.11-20, 2005.

[Kanoun and Crouzet 2006] K. Kanoun and Y. Crouzet, "Dependability Benchmarks for operating Systems", International Journal of Performability Engineering, Vol. 2, No. 3, July 2006, 275-287.

[Kanoun et al. 2007] K. Kanoun, Y. Crouzet, A. Kalakech and A.-E. Rugina, "Windows and Linux Robustness Benchmarks With Respect to Application Erroneous Behavior", LAAS report, May 2006. To appear in "Dependability Benchmarking of Computer Systems", (K. Kanoun and L. Spainhower, Eds.), IEEE CS Press, 2007.

[Kelly & J. McDermid 1997] T. Kelly and J. McDermid, "Safety case construction and reuse using patterns". 16th Conference on Computer Safety, Reliability and Security (Safecomp'97), (York, UK), Springer-Verlag, 1997.

[Kemeny & Snell 1960] J. G. Kemeny and J. L. Snell, "Finite Markov Chains", D. Van Nostrand Company, In. 1960.

[Knight & Amman 1985] J. C. Knight and P. E. Amman, "An experimental evaluation of simple methods for seeding program errors", 8th International Conference on Software Engineering, IEEE Computer Society, 1985.

[Koopman & DeVale 1999] P. Koopman and J. DeVale, "Comparing the Robustness of POSIX Operating Systems", in Proc. 29th Int. Symp. on Fault-Tolerant Computing (FTCS-29), (Madison, WI, USA), pp.30-37, IEEE CS Press, 1999.

[Labovitz et al. 1999] C. Labovitz, A. Ahuja and F. Jahanian, "Experimental Study of Internet Stability and Backbone Failures", in Proc. 29th IEEE Int. Symposium on Fault-Tolerant Computing (FTCS-29), (Madison, WI, USA), pp.278-285, IEEE CS Press, 1999.

[Lanus et al. 2003] M. Lanus, L. Yin, and K. S. Trivedi, "Hierarchical Composition and Aggregation of State-based Availability and Performability Models", IEEE Trans. on Reliability, 52 (1), pp.44-52, 2003.

[Lamprecht et al. 2006] C. Lamprecht, A. van Moorsel, P. Tomlinson and N. Thomas, "Investigating the Efficiency of Cryptographic Algorithms in Online Transactions," International Journal of Simulation: Systems, Science and Technology, UK Simulation Society, Vol. 7, Issue 2, pp. 63—75, 2006.

[Lautieri et al. 2004] S. Lautieri, D. Cooper, D. Jackson, T. Cockram. "Assurance Cases: how assured are you?". In a supplemental volume to DSN-2004, the proceedings of the 2004 International Conference on Dependable Systems and Networks 2004

[Lautieri et al. 2005] S. Lautieri, D. Cooper, D. Jackson. "SafSec: Commonalities Between Safety and Security Assurance". In Proceedings of 13th Safety Critical Systems Symposium, Southampton, February 2005. Springer-Verlag London Ltd.

[Lippman et al. 2000-a] R. Lippmann, D. Fried, I. Graf, J. Haines, K. Kendall, D. McClung, D. Weber, S. Webster, D. Wyschogrod, R. Cunninghan and M. Zissman, "Evaluating intrusion detection systems: The 1998 darpa off-line intrusion detection evaluation", Proc. of the 2000 DARPA Information Survivability Conference and Exposition, January 2000. http://www.ll.mit.edu/IST/ideval/pubs/2000/discex00_paper.pdf

[Lippman et al. 2000-b] R. Lippman, J. W. Haines, D. J. Fried, J. Korba and K. Das, Analysis and Results of the 1999 DARPA Off-Line Intrusion Detection Evaluation, Proc. of RAID 2000, October 2000, pp. 162-182.

[Littlewood & Miller 1989] B. Littlewood and D. R. Miller, "Conceptual Modelling of Coincident Failures in Multi-Version Software", IEEE Trans on Software Engineering 15(12): 1596-1614, 1989.

[Littlewood 1991] B. Littlewood, "Limits to Evaluation of Software Dependability", Software Reliability and Metrics (Proceedings of 7th Annual CSR Conference, Garmisch-Partenkirchen). B. Littlewood and N. E. Fenton. London, Elsevier: 81-110, 1991

[Littlewood et al. 1993] B. Littlewood, S. Brocklehurst, N. Fenton, P. Mellor, S. Page, and D. Wright, "Towards Operational Measures of Computer Security," J. Computer Security, vol. 2, pp. 211-229, 1993.

[Littlewood & Strigini 1993] B. Littlewood and L. Strigini, "Validation of Ultra-High Dependability for Software-based Systems", Communications of the ACM, vol. 36(11), pp. 69-80, 1993.

[Littlewood et al. 1995] B. Littlewood, M. Neil, and G. Ostrolenk, "Uncertainty in software-intensive systems", High Integrity Systems Journal, 1(5), 1995.

[Littlewood 1996] B. Littlewood, "The impact of diversity upon common mode failures", Reliability Engineering and System Safety 51(1): 101-113, 1996.

[Littlewood et al. 1998] B. Littlewood, P. Popov, et al. "Modelling the effects of combining diverse software fault removal techniques." IEEE Trans Software Engineering 26(12): 1157-1167, 1998.

[Littlewood et al. 1999] B. Littlewood, P. Popov, and L. Strigini "A note on modelling functional diversity." Reliability Engineering and System Safety 66(1): 93-95, 1999.

[Littlewood et al. 2002] Littlewood, B., P. Popov, L. Strigini, "Assessing the Reliability of Diverse Fault-Tolerant Software-Based Systems", Safety Science 40: 781-796, 2002.

[Littlewood and Strigini 2004] B. Littlewood, L. Strigini, "Redundancy and Diversity in Security", P. Samarati, P. Y. A. Ryan, D. Gollmann, R. Molva (Eds.): Computer Security - ESORICS 2004, 9th European Symposium on Research Computer Security, Sophia Antipolis, France, September 13-15, 2004, Proceedings. Lecture Notes in Computer Science 3193 Springer, ISBN 3-540-22987-6: pp.423-438, 2004.

[Littlewood & Wright 2006] B. Littlewood and D. Wright, "The use of multi-legged arguments to increase confidence in safety claims for software-based systems: a study based on a BBN of an idealised example", 2006 (www.csr.city.ac.uk/people/ david.wright/dirc/ arguments bbn.pdf).

[Lollini et. al. 2005-a] P. Lollini, A. Bondavalli, and F. Di Giandomenico, "A modeling methodology for hierarchical control systems and its application", in Journal of the Brazilian Computer Society, 10(3), pp.57-69, 2005.

[Lollini et. al. 2005-b] P. Lollini, A. Bondavalli, and F. Di Giandomenico, "Evaluation of the impact of congestion on service availability in GPRS infrastructures", in Lecture Notes in Computer Science, Springer-Verlag, M. Malek et. al. (Eds.), ISAS 2005, LNCS 3694, pp.180-195, 2005.

[Lollini et. al. 2006] P. Lollini, A. Bondavalli, and F. Di Giandomenico, "A general modeling approach and its application to a UMTS network with soft-handover mechanism", Technical Report RCL060501, University of Firenze, Dip. Sistemi e Informatica, May 2006, (http://dcl.isti.cnr.it/Documentation/Papers/Techreports.html).

[Machiraju et al. 2002] V. Machiraju, J. Rolia and A. van Moorsel, "Quality of Business Driven Service Composition and Utility Computing," Hewlett Packard Technical Report HPL 2002-66, 2002.

[MacKenzie 2001], MacKenzie, "Mechanizing Proof: Computing, Risk and Trust", MIT Press, 2001.

[Madeira et al. 2000] H. Madeira, D. Costa and M. Vieira, "On the Emulation of Software Faults by Software Fault Injection", in Proc. Int. Conference on Dependable Systems and Networks (DSN-2000), (New York, NY, USA), pp.417-426, IEEE CS Press, 2000.

[Mainkar & Trivedi 1996] V. Mainkar and K. Trivedi, "Sufficient Conditions for Existence of a Fixed Point-Iteration in Stochastic Reward Net-based Iterative Models", IEEE Transactions on Software Engineering, 22(9), IEEE Computer Society, 1996.

[Majzik and Bondavalli 1998] I. Majzik and A. Bondavalli, "Automatic Dependability Modelling of Systems Described in UML", 9th International Symposium on Software Reliability Engineering (ISSRE), Paderborn, Germany, IEEE Computer Society Press, 1998.

[Majzik et al. 2003] I. Majzik, A. Pataricza and A. Bondavalli, "Stochastic Dependability Analysis of System Architecture Based on UML Models", Architecting Dependable Systems, LNCS 2677, R. de Lemos, C. Gacek and A. Romanovsky Eds., Berlin, Heidelberg, New York, Springer-Verlag: 219-244, 2003.

[Malhotra & Trivedi 1994] M. Malhotra and K. S. Trivedi, "Power-Hierarchy of Dependability Model Types", IEEE Trans. on Reliability, 43 (1), pp.493-502, 1994.

[Marsden et al. 2002] E. Marsden, J.-C. Fabre and J. Arlat, "Dependability of CORBA Systems: Service Characterization by Fault Injection", in Proc. 21st Int. Symposium on Reliable Distributed Systems (SRDS-2002), (Osaka, Japan), pp.276-285, IEEE CS Press, 2002.

[Martinello 2005] M. Martinello, "Availability Modeling and Evaluation of Web-based services - A pragmatic Approach", PhD. Dissertation, Institut National Polytechnique de Toulouse, LAAS-CNRS Report Number 05552, 2005.

[Martinello et al. 2005] M. Martinello, M. Kaâniche and K. Kanoun "Web Service Availability — Impact of Error Recovery and Traffic Model", Reliability Engineering and System Safety, 89 (2005), pp. 6-16, 2005.

[McAfee] McAFee Security Antivirus, "Virus profile: W32/Deloder Worm". Available at: http://us.mcafee.com/virusInfo/

[McDermid 1994] McDermid, J., "Support for safety cases and safety arguments using SAM." Reliability Engineering and System Safety 43: 111-127, 1994.

[McHugh 2000] J. McHugh, “The 1998 Lincoln Laboratory IDS Evaluation, A Critique”, Proceedings of RAID 2000, Toulouse, France, October 2000, pages 145-161.

[Meulen et al. 2004] M. J. P. v. d. Meulen, P. G. Bishop, M. Revilla “An Exploration of Software Faults and Failure Behaviour in a Large Population of Programs”, ISSRE’04, Rennes, France, IEEE Computer Society, pp. 101-12, 2004.

[Meulen & Revilla 2005] M. J. P. v. d. Meulen and M. Revilla, “The effectiveness of choice of programming language as a diversity seeking decision”, 5th European Dependable Computing Conference (EDDC-5), Budapest, Hungary, April 2005, (Dal Cin, M., Kaâniche, M., Pataricza, A., Eds.), pp. 199-209, Springer, Lecture Notes on Computer Science, 2005.

[Meulen et al. 2005] M. J. P. v. d. Meulen, L. Strigini, and M. Revilla, “On the Effectiveness of Run-Time Checks”, Safecomp’05, Halden, Norway, 2005.

[Miner & Parker 2004] A. Miner and D. Parker, “Symbolic Representations and Analysis of large probabilistic systems”, in Validation of Stochastic Systems, LNCS 2925, Springer, 2004.

[Molina et al. 2005] C. Molina-Jimenez, J. Pruyne and A. van Moorsel, “The Role of Agreements in IT Management Software,” in Architecting Dependable Systems III, R. de Lemos, C. Gacek and A. Romanovsky (Eds.), Springer Verlag, Lecture Notes in Computer Science, Vol. 3549, 2004.

[Moore 2002] D. Moore. "Code-Red: a case study on the spread and victims of an Internet worm". Available at: http://wwww.icir.org/vern/imw-2002/imw2002-papers/209.ps.gz, 2002.

[Mueller & Shipley 2001] Patrick Mueller and Greg Shipley, To Catch a Thief, Network Computing, August 2001. http://www.nwc.com/1217/1217f1.html.

[Muppala et al. 1992] J. K. Muppala, A. Sathaye, R. Howe and K. S. Trivedi, “Dependability Modeling of a Heterogeneous VAX-cluster System Using Stochastic Reward Nets”, in Hardware and Software Fault Tolerance in Parallel Computing Systems (D. R. Avresky, Ed.), pp.33-59, 1992.

[Mura et al. 1999] I. Mura, A. Bondavalli, X. Zang and K. S. Trivedi, “Dependability Modelling and Evaluation of Phased Mission Systems: a DSPN Approach”, DCCA-7 - 7th IFIP Int. Conference on Dependable Computing for Critical Applications, San Jose, CA, USA, IEEE Computer Society Press, 1999.

[Mura and Bondavalli 1999] I. Mura and A. Bondavalli, “Hierarchical Modelling and Evaluation of Phased-Mission Systems”, IEEE Transactions on Reliability 48(4): 360-368, 1999.

[Mura and Bondavalli 2001] I. Mura and A. Bondavalli, “Markov Regenerative Stochastic Petri Nets to Model and Evaluate the Dependability of Phased Missions”, IEEE Transactions on Computers 50(12): 1337-1351, 2001.

[Neil et al. 1996] Neil, M., B. Littlewood, et al. “Applying Bayesian belief networks to systems dependability assessment”. Proc 4th Safety Critical systems Symposium, Leeds, Springer-Verlag, 1996.

[Nelli et al. 1996] M. Nelli, A. Bondavalli and L. Simoncini, “Dependability Modelling and Analysis of Complex Control Systems: an Application to Railway Interlocking”, EDCC-2 European Dependable

Computing Conference, Lecture Notes in Computer Science N. 1150. Taormina, Italy, Springer-Verlag: 93-110, 1996.

[Nicol et al. 2004] David M. Nicol, William H. Sanders, and Kishor S. Trivedi, “Model-Based Evaluation: From Dependability to Security”, IEEE Transactions on Dependable and Secure Computing, , Vol. 1, No. 1, January-March, 2004.

[Obal 1998] W. D. Obal, “Measure-Adaptive State-Space Construction Methods”, PhD Dissertation, Univ. of Arizona, 1998.

[OMG 2004] OMG "Unified Modelling Language Specification: version 2.0," http://www.omg.org October 2004

[Oppenheimer & Patterson 2002] D. Oppenheimer and D. A. Patterson, “Architecture and Dependability of Large-Scale Internet Services”, IEEE Internet Computing, 6 (5), pp.41-49, Sept.-Oct. 2002.

[Ortalo et al. 1999] R. Ortalo, Y. Deswarte, and M. Kaâniche, “Experimenting with Quantitative Evaluation Tools for Monitoring Operational Security,” IEEE Trans. Software Eng., vol. 25, pp. 633-650, Oct. 1999.

[Pai & Dugan 2002] G. Pai, J. Bechta Dugan: Automatic Synthesis of Dynamic Fault Trees from UML System Models. International Symposium on Software Reliability Engineering (ISSRE) 2002: 243-256

[Palmer & Mitrani 2005] J. Palmer and I. Mitrani, “Optimal and Heuristic Policies for Dynamic Server Allocation,” Journal of Parallel and Distributed Computing, Vol. 65, Issue 10, pp. 1204—1211, 2005.

[Pan et al. 2001] J. Pan, P. J. Koopman, D. P. Siewiorek, Y. Huang, R. Gruber and M. L. Jiang, “Robustness Testing and Hardening of CORBA ORB Implementations”, in Proc. 2001 Int. Conference on Dependable Systems and Networks (DSN-2001), (Göteborg, Sweden), pp.141-150, IEEE CS Press, 2001.

[Pighin & Marzona 2003] M. Pighin and A. Marzona, “An Empirical Analysis of Fault Persistence through Software Releases”, in Proc. ACM/IEEE Int. Symp. on Empirical Software Engineering (ISESE 2003), (Rome, Italy), pp.206-212, IEEE CS Press, 2003.

[Popov et al. 2003] P. Popov, L. Strigini, J. May and S. Kuball, "Estimating Bounds on the Reliability of Diverse Systems", IEEE Transactions on Software Engineering, vol. 29, no. 4, 2003, pp.345-359, 2003.

[Popov & Littlewood 2004] P. Popov and B. Littlewood, “The Effect of Testing on Reliability of Fault-Tolerant Software”. Int. Conference on Dependable Systems and Networks (DSN'04), Florence, Italy, IEEE Computer Society Press, pp.265-274, 2004.

[Popov et al. 2004] P. Popov, L. Strigini, A. Kostov, V. Mollov and D. Selensky, "Software Fault-Tolerance with Off-the-Shelf SQL Servers", Proc. 3rd International Conference on Component-Based Software Systems (ICCBSS'04), 2-4 Feb. 2004, Redondo Beach, CA, U.S.A., pp. 117-126, Springer, 2004.

[Porcarelli et al. 2002] S. Porcarelli, F. Di Giandomenico and A. Bondavalli. Analyzing Quality of Service of GPRS Network Systems from a Users Perspective. ISCC02 - IEEE Symposium on Computers and Communications, Taormina, Italy, 2002.

[Porcarelli et al. 2003] S. Porcarelli, F. Di Giandomenico, A. Bondavalli, M. Barbera and I. Mura "Service Level Availability Estimation of GPRS," IEEE Transactions on Mobile Computing 2(3): 233-247.2003.

[Pouget et al. 2003] F. Pouget, M. Dacier, and H. Debar, “White paper: honeypot, honeynet, honeytoken: terminological issues”, Eurecom Institute, Research Report RR-03-081

[Pouget & Dacier, 2003] F. Pouget, M. Dacier, “White paper: honeypot, honeynet: a comparative survey”, Eurecom Institute, Research Report, RR-03-082

[Pouget et al. 2004] F. Pouget, M. Dacier and H. Debar, “Honeypots, a practical mean to validate malicious fault assumptions”. In Proc. 10th IEEE International Symposium Pacific Rim Dependable Computing (PRDC-2004), (Tahiti, French Polynesia), 2004, pp.383-388

[Pouget et al. 2006] F. Pouget, G. Urvoy-Keller and M. Dacier “Time signatures to detect multi-headed stealthy attack tools”, in Proc. 18th Annual FIRST Conference, (Baltimore, USA), June 25-30, 2006.

[Puketza et al. 1996] N. J. Puketza, K. Zhang, M. Chung, B. Mukherjee and R. A. Olsson, A Methodology for Testing Intrusion Detection Systems, IEEE Transactions on Software Engineering 22:10, pages 719-729, October 1996.

[Puketza et al. 1997] N. J. Puketza, M. Chung, R. A. Olsson and B. Mukherjee, A Software Platform for Testing Intrusion Detection Systems, IEEE Software 14:5, September--October 1997, pages 43—51.

[Pygott and Stuart 2006] C H Pygott, D G Stuart. “The use of GSN in Network Security”. Presented at the GSN User Club meeting, 24 May 2006, York

[Rabah & Kanoun 1999] M. Rabah and K. Kanoun, “Dependability Evaluation of a Distributed Shared Memory Multiprocessor System”, in 3rd European Dependable Computing Conference (EDCC-3), (A. Pataricza, J. Hlavicka and E. Maehle, Eds.), (Prague, Czech Republic), pp.42-59, Springer, 1999.

[Rabah & Kanoun 2003] M. Rabah and K. Kanoun, “Performability evaluation of multipurpose multiprocessor systems: the "separation of concerns" approach”, in IEEE transactions on Computers, Special Issue on Reliable Distributed Systems, 52 (2), pp.223-36, 2003.

[Riordan et al. 2006] J. Riordan, D. Zamboni and Y. Duponchel, “Building and deploying Billy Goat, a Worm-Detection System”, in Proc. 18th Annual FIRST Conference, (Baltimore, USA), June 25-30, 2006.

[Rodríguez et al. 2002] M. Rodríguez, A. Albinet and J. Arlat, “MAFALDA-RT: A Tool for Dependability Assessment of Real Time Systems”, in Proc. Int. Conf. on Dependable Systems and Networks (DSN-2002), (Washington, DC, USA), pp.267-272, IEEE CS Press, 2002.

[Rojas 1996] I. Rojas, “Compositional Construction of SWN Models”, The Computer Journal, 38 (7), pp.612-21, 1996.

[Rugina et al. 2006] A.-E. Rugina, K. Kanoun and M. Kaâniche, “A System Dependability Modeling Framework using AADL and GSPNs”, LAAS-CNRS Report N° 05666, April 2006.

[Ruiz et al. 2004] J.-C. Ruiz, P. Yuste, P. Gil and L. Lemus, “On Benchmarking the Dependability of Automotive Engine Control Applications”, in Proc. IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN-2004), (Florence, Italy), pp.857-866, 2004.

[SAE 2004] SAE-AS5506, "Architecture Analysis and Design Language," Society of Automotive Engineers, Warrendale, PA 2004.

[SAE 2005] AADL-WG, "AADL Error Model Annex," submitted for formal ballot in September 2005. Copies can be asked by e-mail to info@aadl.info.

[Sahner et al. 1996] R. K. Sahner, K. S. Trivedi and A. Puliafito, “Performance and Reliability Analysis of Computer Systems: An example-based approach using the SHARPE Software Package”, in Kluwer Academic Publishers, 1996.

[Salako & Strigini 2006] K. Salako and L. Strigini, “Diversity for fault tolerance: effects of "dependence" and common factors in software development", Centre for Software reliability, City University, DISPO project technical report KS DISPO5 01, Sept 2006.

[Sanders and Meyer 1991] W. H. Sanders, J. F. Meyer, "Reduced Base Model Construction Methods for Stochastic Activity Networks," IEEE Journal on Selected Areas in Communications 9(1): 25-36, 1991

[Sanders et al. 1995] W. H. Sanders, W. D. Obal II, M. A. Qureshi, F. K. Widjanarko: The UltraSAN Modeling Environment. Perform. Eval. 24(1-2): 89-115,1995.

[Siewiorek et al. 2004] D. P. Siewiorek, R. Chillarege and Z. Kalbarczyk, “Reflection on Industry Trends and Experimental Research in Dependability”, IEEE Tansactions on Dependable and Secure Computing, 1 (2), pp.109-127, 2004.

[Siewiorek et al. 1993] D. P. Siewiorek, J. J. Hudak, B.-H. Suh and Z. Segall, “Development of a Benchmark to Measure System Robustness”, in Proc. 23rd Int. Symp. on Fault-Tolerant Computing (FTCS-23), (Toulouse, France), pp.88-97, IEEE CS Press, 1993.

[Smith et al. 1991] Smith, I. C., D. N. Wall, et al., “DARTS - an experiment into cost of and diversity in safety critical computer systems. IFAC/IFIP/EWICS/SRE Symposium on Safety of Computer Control Systems (SAFECOMP '91), Trondheim, Norway, Pergamon Press, 1991.

[Spafford 1989] E. Spafford. "An Analysis of the Internet Worm". In Proc. of the European Software Engineering Conference, Lecture Notes in Computer Science, Vol. LNCS 387, pp.446-468, Springler-Verlag, 1989.

[Spitzner 2002] L. Spitzner, “Honeypots: Tracking Hackers”, Addison-Wesley, ISBN from-321-10895-7, 2002.

[Spitzner 2003] L. Spizner, “Honeytokens: The Other Honeypot”, Security Focus information, July 2003.

[Staniford et al. 2002] S. Staniford, V. Paxson, N. Weaver. “How to Own the Internet in Your Spare Time”, In Proc. of the 11th USENIX Security Symposium, pp. 149-167, 2002.

[Stankovic and Popov 2006] V. Stankovic and P. Popov, “Improving DBMS Performance through Diverse Redundancy”, 25th IEEE Symposium on Reliable Distributed Systems (SRDS 2006), Leeds, UK, IEEE Computer Society, 2006 (*in the Appendix for Part Arch).*

[Stoll 1988] Stalking the wily hacker Communications of the ACM Volume 31, Issue 5, May 1988, pp.484 – 497, ISSN:0001-0782.

[Strigini et al. 2003] Strigini, L., A. A. Povyakalo, et al. “Human-machine diversity in the use of computerised advisory systems: a case study”. International Conference on Dependable Systems and Networks (DSN 2003), San Francisco, U.S.A., 2003.

[Strigini 2005] L. Strigini, “Fault Tolerance Against Design Faults”. Dependable Computing Systems: Paradigms, Performance Issues, and Applications, H. Diab and A. Zomaya, J. Wiley & Sons: 213-241, 2005.

[TCSEC 1985] “Trusted Computer System Evaluation Criteria (TCSEC), Department of Defense, USA DoD-5200.28-STD, 1985.

[Tomek & Trivedi 1991] L. Tomek and K. S. Trivedi, “Fixed-Point Iteration in Availability Modeling”, in: Informatik-Fachberichte, Vol. 283: Fehlertolerierende Rechensysteme, M. Dal Cin (ed.), pp. 229-240, Springer-Verlag, Berlin, 1991.

[Toulmin 1958] S. E. Toulmin, “The Uses of Argument”, Cambridge University Press, 1958.

[Traverse 1988] P. J.Traverse, “AIRBUS and ATR System Architecture and Specification. Software diversity in computerized control systems” U. Voges, Springer-Verlag. 2: 95-104, 1988.

[Tsai et al. 1996] T. K. Tsai, R. K. Iyer and D. Jewitt, “An Approach Towards Benchmarking of Fault-Tolerant Commercial Systems”, in Proc. 26th Int. Symp. on Fault-Tolerant Computing (FTCS-26), (Sendai, Japan), pp.314-323, IEEE CS Press, 1996.

[Tsai et al. 1999] T. K. Tsai, M.-C. Hsueh, Z. Kalbarczyk and R. K. Iyer, “Stress-Based and Path-Based Fault Injection”, IEEE Transactions on Computers, 48 (11), pp.1183-1201, November 1999.

[Vieira & Madeira 2003] M. Vieira and H. Madeira, “Benchmarking the Dependability of Different OLTP Systems”, in Proc. IEEE/IFIP Int. Conference on Dependable Systems and Networks (DSN-2003), (San Francisco, CA, USA), pp.305-310, IEEE CS Press, 2003.

[Wilson et al. 2002] D. Wilson, B. Murphy and L. Spainhower, “Progress on Defining Standardized Classes for Comparing the Dependability of Computer Systems”, in Supplemental Volume of the IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN-2002) - Workshop on Dependability Benchmarking, (Washington, DC, USA), pp.F1-F.5, 2002.

[Woodside et al. 1995] C. Murray Woodside, John E. Neilson, Dorina C. Petriu, Shikharesh Majumdar: The Stochastic Rendezvous Network Model for Performance of Synchronous Client-Server-like Distributed Software. IEEE Trans. Computers 44(1): 20-34, 1995.

[Xu et al. 1995] J. Xu, A. Bondavalli, and F. Di Giandomenico, “Dynamic adjustment of dependability and efficiency in fault-tolerant software”. In B. Randell, J.C. Laprie, H. Kopetz, and B. Littlewood, editors, Predictably Dependable Computing Systems, pages 155-172. Springer-Verlag, 1995.

[Xu et al. 1999] J. Xu, Z. Kalbarczyk and R. K. Iyer, “Networked Windows NT System Field Failure Data Analysis”, in Proc. 6th Pacific Rim Int. Symp. on Dependable Computing (PRDC-2000), (Hong-Kong), pp.178-185, IEEE Computer Society Press, 1999.

[Yuste et al. 2003] P. Yuste, J.-C. Ruiz, L. Lemus and P. Gil, “Non-Intrusive Software-Implemented Fault Injection in Embedded Systems”, in Proc. 1st Latin American Symposium on Dependable Computing (LADC-2003), (R. de Lemos, T. S. Weber and J. B. Camargo Jr., Eds.), (São Paulo, Brazil), LNCS 2847, pp.23-38, Springer, 2003.

[Zhou and Lang 2003] M. Zhou, S.D. Lang, “A Frequency-Based Approach to Intrusion Detection”. In Proc. of the Workshop on Network Security Threats and Countermeasures, July 2003.

[Zhu et al. 2003] J. Zhu, J. Mauro and I. Pramanick, "Robustness Benchmarking for Hardware Maintenance Events", in Proc. IEEE/IFIP Int. Conference on Dependable Systems and Networks (DSN-2003), (San Francisco, CA, USA), pp.115-122, IEEE CS Press, 2003.

[Zou et al. 2002] C.C. Zou, W. Gong, D. Towsley, "Code Red Worm Propagation Modeling and Analysis". In Proc. of the ACM Conference on Computer and Communications Security CCS'02, (Washington DC, USA), Nov. 2002.

Part Verif – Methods and Tools for Verifying Resilience

Co-ordinator: Friedrich von Henke

Contributors: Jean Arlat[1], Cinzia Bernardeschi[2], Péter Bokor[3], Dave Jackson[4], Istvan Majzik[3], Paolo Masci[2], Zoltan Micskei[3], Nick Moffat[4], Holger Pfeifer[5], Birgit Pfitzmann[6], Marco Serafini[7], Neeraj Suri[7], Friedrich von Henke[5], Hélène Waeselynck[1]

[1]LAAS-CNRS, [2]Pisa, [3]Budapest University, [4]QinetiQ, [5]Ulm, [6]IBM Zürich, [7]Darmstadt

Chapter co-ordinators:

1 - Deductive Theorem Proving: Friedrich von Henke

2 - Model Checking: Friedrich von Henke

3 - Symbolic Execution and Abstract Interpretation: Friedrich von Henke

4 - Robustness Testing: Hélène Waeselynck

5 - Verification of Systems Containing Cryptography: Birgit Pfitzmann

Introduction

Verification means demonstrating that a system satisfies certain requirement. Claims of resilience, dependability, or fault tolerance attributes of systems must be substantiated by suitable verification processes, so that trust in the correct and expected working of the systems is justified. The need for proper verification and validation is well understood in particular for systems that perform functions critical for safety and/or security, such as in the avionics, automotive or process control, and electronic business; it is to be expected that verification will become equally important for the emerging ubiquitous systems.

The term "verification" refers to a range of techniques. This part is concerned with techniques that are – to varying degrees – formal in that they have a basis in mathematical logic or make use of formal models. Specifically, this part addresses the following verification techniques:

- *Deductive Theorem Proving*: A system is being modelled, using a logic-based specification language, at a certain level of abstraction suitable for expressing and verifying the property under consideration. The property is then derived from the system as a "theorem", using deductive inference techniques. Many applications rely on the greater expressive power of an underlying higher-order logic, as provided by specification and proof systems such as Isabelle, PVS and Coq. The process of generating proofs is typically interactive and usually requires specialized skills.

- *Model Checking.* The basis of model-checking techniques is usually a finite-state model of the system. Efficient algorithms are used for checking whether desired properties hold for the system, essentially by state space exploration. System properties are commonly expressed in some form of temporal logic (primarily CTL or LTL). In contrast to verification using theorem proving, the checks can be performed automatically, i.e. without user interaction; furthermore, when a check fails the model checking system will generally be able to generate counterexamples which then can be used for debugging. Both of these properties contribute to the growing popularity of model checking techniques in industry. However, the usability of model checking is limited by the state explosion problem: the number of states grows exponentially with the number of variables (and possible variable values) that together form the system state, thus making the model-checking problem eventually intractable. Various abstraction techniques are used to overcome those limitations; in particular, abstraction may also be used to reduce an inherently infinite state space to a finite one.

- *Abstract Interpretation and Symbolic Evaluation.* The term "abstract interpretation" refers to an analysis technique whereby a program is executed in a simplified, possibly symbolic, space that abstracts from its proper value space. This can be used, for instance, to analyze information flow in a security context. Abstract interpretation and other forms of symbolic execution are typically performed on actual programs, hence this kind of techniques is closer to testing than model checking.

- *Testing* is certainly the most common and time-honoured form of verification. Strictly speaking, the presence of a system property (or the absence of a system error) can be "verified by testing" only if it can be demonstrated that the set of test cases is representative for all possible system behaviours. However, here the term verification is also used in a looser, partial sense, without the claim of completeness. On the other hand, testing is more realistic than strictly formal methods as it can be performed on the real system. Testing can be used to demonstrate various types of properties. The chapter included in this part focuses on testing for a property that is of particular importance for resilience and emerging ubiquitous systems: the robustness of systems, i.e. their capability to withstand unexpected or faulty external behaviour.

The verification techniques described here should be regarded not as competing, but as complementary. In fact, there are numerous situations in which the techniques interact. For example:

- Theorem proving is used to justify an abstraction necessary to make model checking of a particular property feasible.
- Model checking may be used as an exploratory test whether a general property to be verified by theorem proving holds in a reduced finite model.
- State exploration techniques derived from model checking may be used to generate suitable sets of test cases.

The exploration of such interactions among verification techniques will be the subject of further work by the working group.

Modelling and Abstraction

A theme common to all verification techniques is that of modelling and abstraction. Verification by theorem proving or model checking relies on the existence of a formal model; abstract interpretation and testing also often make use of a formal model, for instance to justify an abstraction or argue for adequacy of test sets. In some cases, the model can be the system itself if a formal semantics of the implementation is available. Building a more abstract model of a system by itself already has benefits usually associated with formal methods: the opportunity of detecting, at an early stage of system development, subtle mistakes, inconsistencies and incompleteness.

Every model represents an abstraction of the modelled system, and systems can be modelled at different levels of abstraction. This is very important for scalability of formal verification; in principle, there is no inherent limit on the size of a system to be verified, as long as an appropriate level of abstraction is selected. It remains, however, a challenge to come up with suitable abstraction for modelling dynamically changing, ubiquitous systems and specifying interfaces of components systems and system-wide resilience properties.

When a property has been verified for a system model, there still remains the gap between the model and the actual implementation of the system. Ideally, the implementation would be derived, through refinement or similar methods, from the abstract model, or, if the system was there first, at least the refinement relation would be formally established. In practice, this is often not (yet) feasible, at least not for larger systems. Current developments in the areas of "model-driven architecture" (MDA) and "model-driven design" (MDD) with their strong focus on using models in the process of engineering systems promise to fill the gap; a requisite for truly combining formal verification with system development is a formal semantics of the modelling languages (which, for languages such as UML are still lacking). Formal models that have been

built for verification purposes can also be used to drive test case generation; also, if specifications are in some sense (symbolically) executable, they can be executed together with the actual implementation, thus providing some means of verifying the implementation.

The following chapters summarize the knowledge and work of ReSIST partners in the broad area of verification. They are organized primarily according to the verification techniques as listed above.

An exception is the chapter on "Verification of systems containing cryptography". Modelling cryptographic aspects of a system naturally involves probability and computational complexity, which do not easily fit into the typical approaches to formal verification. The chapter thus represents an example for developing techniques that integrate specialized analysis and verification methods for particular system aspects with the general formal methods approaches – an issue that will be crucial for scalability of verification techniques. In particular, the presented techniques permit to abstract from probabilities and computational issues of cryptographic subsystems; such abstractions are thus usable in the normal way in formal methods that have a concept of layered verification, but are nevertheless sound with respect to the more detailed cryptographic definitions.

1 – Deductive Theorem Proving

Introduction

Theorem proving means deriving theorems by logical deduction. In the context of verifying that a system satisfies certain requirements (of fault tolerance, security, etc.), the system – or some essential aspects of its behaviour – is described (modelled) in a logic-based language, and a property to be verified is expressed as a 'theorem' that is to be proved from the system model.

Given a sufficiently expressive logic in which to describe both system models and properties, theorem proving can in principle be applied to any type of verification problem. In fact, for many realistic applications the expressiveness of some higher-order logic proves useful to build concise system models. Moreover, other specialized formalisms dedicated to modelling particular kinds of systems, such as, for example, the pi-calculus for modelling concurrent and mobile systems [Milner et al. 1992], can be embedded in the language of higher-order logics, thus providing a means to reason about models expressed in that formalism [Honsell et al. 2001, Röckl et al. 2001].

However, higher-order logics are not decidable and thus cannot provide automatic decision procedures to derive properties from specifications. Hence, using expressive logics comes at the price of reduced proof automation, and theorem proving tools based on higher-order logic such as Isabelle [Nipkow et al. 2002], Coq [Bertot and Castéran 2004], or PVS [Owre et al. 1995], are therefore typically used in an interactive style. Indeed, substantial theorem proving tasks are practically not feasible without machine assistance. While the verification expert guides the deduction of properties by interactively applying the proof rules, the theorem proving system can perform simple deduction steps automatically and assists in managing large proving efforts. Computer-based systems are much more reliable than humans when it comes to keeping track of the minute, but possibly essential, details of proofs; in addition, a proof that has been generated with machine assistance can usually be repeated and examined by others (humans or machines); ultimately, such proofs may be introduced as evidence in certification processes.

The distinguishing power of theorem proving is its capability to handle systems of arbitrary size. The reason for this is that the underlying (higher-order) logic provides for systems being described in a declarative way using logic formulas. The fact that system models do not need to be executable, like the state-machine models required in approaches based on model checking, allows systems or components be modelled as a set of axioms that describe the behaviour of the system. Properties are then derived as theorems from these axioms using the proof rules of the logic, and these theorems are valid for all possible implementations of the system that satisfy the stated axioms. Modern theorem proving tools facilitate this kind of reasoning by providing suitable specification mechanisms. In PVS, for example, systems models, which are called *theories* there, can be parameterized by both values (e. g., the number of entities of a system) and functions (e. g., the method to use for performing a majority vote). The intended meaning of concrete values of these parameters can be constrained by so-called *assumptions*, and theorems are derived based on these assumptions. In order to establish these theorems for a given instance of the parameters, only the validity of the assumptions has to be shown for the parameters. The verification is thus performed in a both *generic* and *abstract* way. Genericity refers to the fact that a significant part of the deduction is carried out once and for all for a whole class of possible concrete instantiations, while the assumptions on parameters of a model are

usually expressed in a way that abstracts from particular details that are not relevant for establishing the proofs for the desired correctness properties.

Parameterization provides another aspect that is crucial when modelling large and complex systems, which is that of decomposition. In order to cope with complexity, it is advantageous to divide the analysis into smaller and manageable parts, and decompose a system, where possible, into a set of modules that can be analysed more or less independently from the each other. The parameters of a given module then serve as an interface to other parts of the system, and assumptions on these parameters can be used to express, again in an abstract way, what is required from the other modules. The overall proof of the whole is then composed from the proofs of the individual modules.

Finally, analysis techniques based on theorem proving show their full strength in the domain of fault-tolerant systems. While other methods generally need to provide several analyses with respect to various different fault scenarios, theorem proving can treat all possible fault scenarios at once. The reason for this is that there is no need to explicitly model the behaviour of faulty components of a system. Instead, the analyses can be carried out relative to components whose faulty behaviour is left completely unconstrained. Thus, fault-tolerance properties of a system can be proved for the most general case where one does not make any assumption about the kinds of faults that can occur.

The application of theorem proving to verifying essential properties of systems requires fairly specialized skills, both for modelling the system under consideration and developing proofs. It is therefore also a time-consuming and cost-intensive process, which constitutes one of the main reasons why methods based on theorem proving, compared to more automatic approaches, have not found wide-spread usage in industry, yet. Consequently, the practical use of theorem proving is generally concerned with formalizing and verifying essential properties of systems that are considered so critical as to warrant the effort. One such domain is that of hardware, where it can extremely expensive if design faults are detected only after the mass production in silicon. Modern hardware designs, such as floating-point units, multi-level caches, or pipeline processors, however, have become so complex that automated methods based on exhaustive exploration of all possible behaviour reach their limit with regard to time and space resources required. Theorem proving approaches can copy with the complexity by modelling the designs at higher levels of abstraction.

Another important application area is that of fault-tolerant real-time systems of the kind that is increasingly deployed in aircraft and automobiles. The following section summarizes the modelling and verification (by theorem proving) of a communication architecture that was specifically designed for such critical applications: the Time-Triggered Architecture (TTA). TTA is a good example of the difficulties that arise when carrying out formal analyses of these kinds of systems. On the one hand, there is the ever growing complexity of such systems, and industrial needs to reduce costs wherever possible adds to this complexity by imposing additional requirements on the design of the system. In the case of TTA, for instance, basic services of the underlying protocol are not implemented by clearly separated algorithms; instead the services are deeply integrated so that one service makes use of – and thus relies on – other services, not the least in order to reduce the size needed when the system is implemented in silicon. On the other hand, systems such as TTA are typically designed for a variety of applications and hence need to be adaptable to different configurations. TTA, for example, can be used in systems of different size in terms of the number of nodes that are connected to the communication network. While the maximum number is limited by the length of messages and the size of look-up tables, different degrees of fault tolerance that one wishes the system to provide impose restrictions on the minimum number of nodes that must be available. It is, however, desirable, not having to repeat the verification of critical properties of TTA for every single instance of a

TTA system. As laid out above, using a theorem proving approach, models can be parameterized such that properties are derived generically and hold for all possible instantiations.

1.1 Deductive Theorem Proving for Fault-Tolerant Real-Time Systems

Through its participation in a number of European research projects Ulm University has conducted research in the area of formally modelling and analysing critical fault tolerance properties of a particular real-time system, viz. the Time-Triggered Architecture (TTA). TTA is a distributed computer architecture for the implementation of highly dependable real-time systems specifically targeting embedded applications, such as digital control systems in the automotive and avionics domain. Ulm have formally modelled and analysed various aspects of the underlying communication protocol TTP/C and its fault tolerance properties. The algorithms implementing the basic protocol services of TTP/C are heavily intertwined and pose challenging problems for formal analysis. This is true not only with regard to the construction of formal proofs, but also for the development of the formal models themselves.

To properly assess the lessons learned it is important to note that the work has been carried out over the course of several years, during which the object of the analyses, the Time-Triggered Architecture, has undergone a number of design modifications. Most notably, the structure of the employed communication network shifted from the previously used bus topology to a more modern star-based topology, and thus special components that control the nodes' access to the communication medium, the so-called guardians, changed from node-local bus guardians to central guardians located in the centre of the star. While early work in Ulm has been concerned with the formal analysis of the most critical mechanisms of TTP/C, namely group membership and clock synchronisation, subsequent research addressed the questions of how to carry over results established for the earlier bus architecture to the new, and how to integrate the correctness properties gained in separate analyses into a combined result.

The following provides a broader overview of that work and specifically focuses on the modelling approaches taken to connect the correctness arguments constructed for the individual items. The experience gained confirms the need for an adequate structuring of the models and proofs along different levels of abstraction to enable the formal modelling of the central protocol algorithms, make their analysis feasible, and resolve the mutual dependencies among the services.

1.1.1 Formal Models for TTA

The central part of the formal analysis work for TTA is represented by a formal framework for modelling and analysing the Time-Triggered Architecture and the services provided by the Time-Triggered Protocol. One of the goals in the design of the framework was to allow for an easy integration of work carried out earlier on the formal analysis of fundamental algorithms of the Time-Triggered Protocol, such as the group membership algorithm [Pfeifer 2000] and the clock synchronisation algorithm [Pfeifer et al. 1999]. To this end, a series of generic and abstract formal models was developed that enable the various aspects of TTA be analysed individually without compromising the necessary eventual integration of the individual analyses.

A natural way to structure the framework is to create several formal models according to the architectural design of a TTA system. In TTA, host computers that implement the given application are connected to a logical bus via so-called TTA controllers; the combination of a host and its controller is called a *node*. Nodes communicate via replicated shared media, called *channels*. Access to the communication media is granted to the nodes in a Time-Division Multiple-Access (TDMA) scheme, i. e., each node owns predefined periods of time, called *slots*, where it has exclusive rights to write to the bus. Consequently, the formal modelling

framework for TTA consists of two major classes of models: first, generic models that describe the general behaviour of the nodes in a TTA system and, second, generic models that focus on the fault-tolerant properties of the communication network of a TTA system. Both of these classes of models provide suitable abstract interfaces and thus provide the basis for the formal models for the analysis of the fault-tolerant properties of the particular protocol algorithms of TTP/C.

Finally, the formalisation of the actions that can be taken by the environment is what constitutes the fault model. Here, one has to describe the assumptions about what kind of faults can possibly occur to a node, and how the messages sent by a node relate with the messages a node can receive at the other end. In the formal modelling framework, this latter point is not included in the generic models for the nodes' behaviour, but rather treated separately in the formal models about the communication network of TTA. This has the advantage that the algorithms executed by the nodes can be analysed with respect to different fault models, and also independently of a particular network topology. The following subsections briefly describe the main aspects of these two building blocks of the formal modelling framework.

1.1.1.1 Generic models for node behaviour

The models that describe the general behaviour of the nodes in a TTA system are intended to form the common ground for the analysis of the most critical properties of TTP/C. The models abstract from various details in the execution of the TTP/C protocol. In essence, the behaviour of a node is reduced to describing the actions it takes in a single slot of a TDMA schedule, that is, the sending and receiving of messages, and the update of its internal state according to the message received. Therefore, there are three entities that describe the behaviour of a TTA node: its current internal state, the message it sends in a slot, if any, the message it receives in a slot. The values of these entities at a given time are determined on the one hand by the protocol algorithms the node is executing and, on the other hand, by the environment. While a node can control its own internal state and the messages it is sending, the environment is responsible for the messages a node can receive. Moreover, the environment can cause a node to become faulty. Consequently, two objects are sufficient to model what it means for a node to execute (an algorithm of) the Time-Triggered Protocol: a state-transition function and a message-generation function, which become parameters of a generic model for describing a node's internal behaviour. To formally model a given protocol algorithm such as, for instance, group membership, one has to provide concrete values to these two parameters.

With regard to the concrete modelling of the actions a node takes, that is, its state transitions and the generation of messages, one has to decide at which granularity these actions should be modelled. A more abstract description usually facilitates formal analysis, while a more detailed model allows for the analysis of different kinds of properties that might not be expressible within the abstract model. For the analysis of the protocol algorithms of TTP/C, two levels of abstraction have proved adequate: the un-timed synchronous systems model and the timed, synchronized system model. The first assumes that all nodes execute their state transitions synchronously in a lock-step fashion. At this level of abstraction, the state transitions of nodes can be modelled by a simple recursive function on the slot numbers. However, not all properties are expressible at this level. In particular, this refers to everything that specifically involves time, such as a clock synchronisation service, since the timing aspect is abstracted away in the synchronous system model. Hence, one also needs a more detailed description to model the timing behaviour of nodes. In comparison to the model for the un-timed, synchronous level the timed model additionally introduces entities that formalise the local clocks of the nodes. The state updates and generation of messages are now described in a more fine grained way on the level of the ticks of these clocks, rather than in terms of slots. Again, the functions

describing the state transitions and the message generation are only parameters of the model thus providing a generic interface for various concrete algorithm instances.

Obviously, if different aspects of the communication protocol TTP/C are modelled and analysed at different levels of abstraction, using different formal models, the question arises how the different model layers relate to each other. In fact, the un-timed synchronous system model is a sound abstraction of the time-triggered system model provided that the clocks of the processors are synchronised within a small bound. This relationship can be captured generically [Rushby 1999], i. e. it is independent of the particular algorithm that is modelled at the synchronous level. Rushby's proof allows properties, which have been established on the more abstract level of synchronous systems, to be transformed, or refined, to the timed system level. Moreover, it should be noted that the validity of this generic refinement proof depends on the presence of a clock synchronisation service, because only if the clocks of the node are synchronised tightly enough can the internal states of the nodes as expressed on the abstract level be faithfully translated onto the timed system level. Depending on the precision of the clock synchronisation, one obtains bounds on the instant at which a node must begin its message broadcast at the latest, and on the length of the interval during which a node must wait for messages to arrive. These qualitative constraints relating the various timing entities involved can in fact be extracted from the formal refinement proof.

1.1.1.2 Models for the communication network

The modelling of the communication network describes the actions taken at a node to send and receive messages, and the functionality of the communication channels and their guardians. The overall goal of modelling the communication network is to provide a concise description of the arguments that support the following three main correctness properties of the TTP/C communication:

- *Validity*: If a correct node transmits a correct frame, then all correct receivers will accept the frame.
- *Agreement*: If any correct node accepts a frame, then all correct receivers do.
- *Authenticity*: A correct node accepts a frame only if it has been sent by the scheduled sending node of the given slot.

Once these properties are established, they can be exploited in subsequent analyses of protocol algorithms. This is preferable, since it is generally more feasible to base an analysis on properties of a supporting model or theory, rather than on the mere definitions of the model itself. In order to facilitate the deduction, the formal proofs of these properties are decomposed into a series of smaller steps, and a hierarchy of corresponding models has been developed. Each of the single models focuses on a particular aspect of the communication. Altogether, four suitable model layers have been identified **[Pfeifer and von Henke 2006]**:

- General specification of the reception of frames.
- Channels without guardians, requiring a strong fault hypothesis.
- Channels with guardians, requiring only a weaker fault hypothesis.
- Different network topologies: local bus guardians and central guardians.

Each of the models contributes a small step towards proving the desired correctness properties. The steps themselves are each based on a set of assumptions or preconditions. In essence, on each model layer *i* one establishes a theorem of the form $assumptions_i \Rightarrow properties_i$. The idea is to design the different models in such a way that the properties on one level establish the assumptions on the next. Ultimately, the models are integrated and the reasoning is combined, yielding a chain of implications.

$$assumption_0 \Rightarrow properties_0 \Rightarrow assumptions_1 \Rightarrow properties_1 \Rightarrow assumptions_2 \Rightarrow \ldots \Rightarrow properties_f$$

The final properties, $properties_f$, correspond to the desired main correctness properties of the TTP/C communication as specified above, while the initial assumptions, $assumptions_0$, describe what constitutes the basic fault hypothesis. The hierarchic arrangement of the models for the communication network allows for a concise description of the dependencies of the three main correctness properties. On the basic level the fundamental prerequisites are described that are necessary for the desired correctness properties to hold, while the subsequent levels express what must be assumed from the nodes and guardians, respectively, to satisfy these prerequisites. In particular, the treatment precisely explains the benefits of introducing guardians into the communication network. Moreover, the genericity of the guardian models serves as an interface to the analyses for particular algorithms of the TTP/C protocol. In these analyses one can abstract from the concrete realisation of the guardians and the topology of the communication network, and instead use the properties established for the generic guardian. Thus, correctness arguments for a given protocol algorithm can be instantiated for either guardian variant.

1.1.2 Formal Analysis of TTA

This section describes how the generic modelling framework can be employed to analyse various aspects of the protocol TTP/C in a way that allows connecting the correctness arguments developed for the individual protocol algorithms.

1.1.2.1 Analysing central algorithms within the general framework

The formal modelling framework for TTA provides a set of basic models that allow algorithms be modelled at different levels of abstractions, such as the un-timed synchronous level and the timed-synchronized level. These models offer suitable interfaces to allow for the integration of analyses of fundamental algorithms of the Time-Triggered Protocol, such as group membership and clock synchronisation. For instance, the group membership algorithm is modelled at the un-timed, synchronous system level by instantiating the state transition function of the synchronous system model with a definition of the TTP/C membership algorithm. The clock synchronisation was analysed at the more concrete timed system level.

The mentioned analyses of critical protocol algorithms have been carried out using a number of abstractions. For instance, the description of the sending and receiving of messages by the nodes has been kept abstract. Particularly, entities such as a bus guardian have not been modelled explicitly in these models. However, the definitions and proved facts for the basic formal models of the TTA architecture can easily be re-used in the analyses of the protocol algorithms by providing concrete values for the parameters of the basic models. Thus, the analyses of particular protocol aspects can be coupled to the developments concerning the overall architecture. Moreover, previous analyses have been carried out relative to a strict fault model, and it could be argued that the assumptions of this fault model, and thus the analyses, are not very realistic. On the other hand, the functionality of the central guardians provide for such strict assumptions. Hence, integrating the analyses of the protocol algorithms with the models of the communication network, one can make use of the properties of the central guardians. In this way, the basis for the examination of the protocol algorithms is established, as the adoption of a strict fault model for the communication in the analysis of TTP/C is justified by the reasoning carried out on the guardian properties.

1.1.2.2 Integration of group membership with clock synchronisation

A closer look at the formal models reveals that clock synchronisation and group membership depend on each other. On the one hand, there is a hierarchical dependency: the verification of the group membership algorithm uses the un-timed system model, which abstracts from a synchronisation mechanism, and therefore depends on the correctness of the clock synchronisation service. On the other hand, the TTP/C clock synchronisation algorithm is totally integrated into the exchange of data messages. Since a processor only accepts messages from processors that it considers to belong to the same membership set, clock synchronisation actually depends also on group membership. These apparently circular dependencies need to be broken by combining the proofs so that the results obtained by analysing the individual services in isolation can also be established for the integrated services. We briefly describe how an integrated proof of the two algorithms is accomplished.

First, one observes that a simple combination of the two verification results fails because the levels of abstraction at which the services are modelled do not match. The clock synchronisation algorithm has been modelled at the more detailed time-triggered system level, whereas the group membership service has been dealt with at the more abstract level of the synchronous system model. As mentioned earlier, the un-timed synchronous system model is a sound abstraction of the time-triggered system model provided that the clocks of the processors are synchronised within a small bound. To accomplish an integrated proof, it is necessary to split up the analysis into a series of successive intervals, corresponding to the synchronisation intervals used in the analysis of the clock synchronisation algorithm. This is the result of the observation that there is a temporal dependency between group membership and clock synchronisation: in order for the synchronisation algorithm to adapt the processors' physical clocks correctly, and thus keeping them synchronised during the following period $i+1$, it must rely on the membership service having been available during the current synchronisation period i. However, a direct formulation of this dependency compromises the original goal of separating the analyses of the two algorithms. Therefore, the requirements on the group membership that are necessary for synchronisation are expressed abstractly, by means of un-interpreted prerequisites in the proof of clock synchronisation. Then one can prove, by induction on the synchronisation interval i, that for all intervals both the clock synchronisation property and the group membership property hold. For the inductive step in this proof it is necessary to show that the abstractions used in the individual proofs can be resolved. On the one hand, this means that the proof of group membership must be transferred from the higher level of abstraction to the level in which the clock synchronisation algorithm is modelled. On the other hand, it must be shown that group membership indeed provides an adequate interpretation of the abstract entities in the synchronisation model such that the presupposed hypotheses for clock synchronisation are satisfied.

Conclusions and Perspectives

The work on formal analysis of critical aspects of the Time-Triggered architecture particularly emphasises the importance of abstraction and genericity in the design and structuring of the formal models and proofs. Abstraction is a fundamental prerequisite for the feasibility of formal analysis, as it allows for both structuring the models by the provision of abstract interfaces and hiding details unnecessary or irrelevant for the formal analysis and the demonstration of critical properties. This is particularly important for the analysis of fault-tolerant systems; these systems are inherently difficult to analyse because of the possibly unrestricted behaviour of faulty components, which immensely increases the complexity of the analyses. It is therefore necessary to develop suitable abstractions with which to build models of different kinds of algorithms, services, and architectures. Through the provision of abstract interfaces, these models then offer

a certain degree of genericity, which enables one to express the commonalities of a range of designs in a coherent way. As for TTA, for instance, the models that describe the behaviour of the communication channels provide a generic treatment of the guardians and can be refined to either a central guardian-based view or to a model for local bus guardians. The ultimate goal would be to develop a library of formally specified and verified components, which can be re-used in the design and development of ubiquitous systems. [Pike et al. 2004] describe first steps in that direction for certain fault-tolerant algorithms deployed in fault-tolerant real-time architectures.

A key to establishing correctness proofs for two of the most critical algorithms of TTA, clock synchronization and group membership, was to perform the analyses at different levels of abstractions suitable for the particular algorithm at hand. To this end, however, it was also necessary to extract separate descriptions of the two algorithms from the overall protocol. This turned out to be a nontrivial step, because they are greatly integrated and even mutually depend on each other. The separation was accomplished by factoring out the exact dependencies, and using them as prerequisites in the individual proofs. For this approach, a sufficiently expressive formalism such as languages based on higher-order logics is indispensable so as to describe the required property of one algorithm through a set of logic formulas, and use these in the analyses of the other. While decomposition is one aspect, composing the parts is another. As has been demonstrated for the case of TTA, such a composition of individual proofs can require additional careful reasoning. Further research is necessary to investigate how the process of decomposition and composition can be applied to large and evolving systems in a systematic way.

2 – Model Checking

Introduction

Model checking is a technique for verifying that certain desired properties hold for a model of a system. Various languages are available for describing the model. Whatever the language, the model is typically translated to a finite state transition system. The properties are usually described by formulae expressed in a temporal logic (mainly CTL or LTL), though a refinement style of model checking is also possible. The check amounts to an exhaustive search through the state space of the model. If a check fails, i.e., if the desired property does not hold for the model, the checking process usually returns some information that can be used to localize the problem; in this sense model checking also serves as a testing and debugging technique.

Powerful algorithms have been developed to perform model checking automatically (i.e., without user interaction) and often very efficiently, sometimes allowing systems with very large state spaces to checked. This makes model checking very attractive. The technique has been used very successfully, initially in the area of hardware verification but increasingly in other areas also. ReSIST members have applied model checking successfully in several areas, as illustrated by the variety of applications described below.

Although model checking can be very effective, all model checkers suffer from the so-called *state explosion problem*. This refers to the tendency for the number of states of a model to increase exponentially as the size of the model increases, measured in terms of the number of interacting components. Since model checkers aim to explore complete state spaces, this can lead to models rapidly becoming intractable to model check within practical memory and time constraints (i.e., attempts to model check can exhaust the available memory, or not complete within an acceptable time). Techniques must therefore be found for combating the state explosion problem, to enable models of realistic size to be model checked. This is a common thread in the following descriptions of model checking application areas.

Several techniques for combating the state explosion problem are discussed. Section 2.1 focuses on abstraction approaches for model checking, studied at Technical University Darmstadt. The approaches considered depend on certain well-formedness properties (e.g., symmetry, data abstraction) of the system. The section elaborates on a novel approach for model checking synchronous protocols. Section 2.2 summarises research at Pisa University in the use of process algebras and action-based model checking. In this type of model checking, the system under study is described using a process algebra and properties to be checked are described using ACTL (an action-based temporal logic in the style of CTL). An approach to formalizing fault-tolerant systems is outlined, and it is explained how certain key characteristics of typical fault-tolerant systems can be exploited to combat the state explosion problem. Section 2.3 discusses how model checking can be used to explore the effects of system faults in the case of distributed real-time systems. It outlines *exhaustive fault simulation*, which provides a "dial" that allows a single model to be used both for rapid exploration with respect to few faults and for exhaustive verification with respect to many faults. Particular consideration is given to model checking of the Time Triggered Architecture at Ulm University. Finally, Section 2.4 summarises several applications by QinetiQ of refinement-style model checking to the analysis of distributed algorithms and protocols relevant to ubiquitous computing. This style of model checking uses a process algebra to describe the system and the properties to be checked, and a refinement checker to check that all possible behaviours of the former are allowed by the latter.

2.1 Abstraction in Model Checking

2.1.1 Introduction

This section elaborates various approaches for reducing the state space that must be explored by a model checking engine. To deal with the exponential growth of state spaces, it is necessary to scale the problem down, i.e. non-relevant details must be omitted from the model in order to make model checking feasible. In this section we survey those abstraction approaches (with respect to model checking) that assume the system exhibits certain well-formedness properties (e.g. symmetry, synchronous constraints, etc.) and we propose (partly) automated solutions intended to reduce the size of the state space. We also elaborate on a novel approach to model checking synchronous protocols.

Temporal logic model checking is a technique for determining whether a temporal logic formula is valid in a finite state system *M = (S, R, L)*, where *S* is the state space, *R* is the state transition relation, and *L* is a function that labels states with sets of atomic propositions[1]. Such a structure is usually called a Kripke structure and may entail enormous number of states because of the state explosion problem. A system with *n* Boolean state variables can have a state space which is exponential in *n*.

In [Kesten and Pnueli 2000] the authors introduce two special classes of systems for which the combination of compositionality and abstraction can effectively reduce the complexity of the model checking problem. We will classify the various approaches that address the reduction of the state space in model checking. The first class of systems is where the complexity of the problem results from the parallel composition of multiple finite-state processes (each described as a Kripke structure). For such systems complexity stems from the parameterized nature of the system architecture, i.e., the number of composing processes might be large (or even unbounded). Techniques addressing this aspect to reduce state space are described as *control abstraction*. Another typical scenario which makes the state space grow exponentially is where system variables range over extensive (or even infinite) domains (e.g. integers). Approaches which aim at tackling complexity of this kind are referred to as *data abstraction*.

2.1.2 Parameterized Networks

The technique of parameterized networks [Kurshan and McMillan 1995] is a control abstraction for infinite systems. It addresses the following verification problem: given a family $F = \{P_i\}_{i=1}^{\infty}$ of systems P_i and a temporal formula *f*, verify that each state transition system in the family *F* satisfies *f*. Many existing frameworks propose solutions to the problem of parameterized networks. [Clarke and Jha 1995] provides an approach for families of systems derived by so-called *network grammars*. According to this approach (and similarly to computational linguistics), a system P_i is a valid word (a string of terminals) of the language generated by applying grammar rules. The advantage of such a grammar is that it is a finite (and usually small) representation of an infinite family of finite systems (referred to as the language of the grammar). As an example one may think of a token ring of processing nodes, e.g. implementing mutual exclusion, without imposing a bound on the number of participants. In this case, a particular grammar (i.e. set of production rules) that produces rings with one process Q and at least two processes P is given by the set of

[1] Usually, model checkers accept models defined in higher level specification languages which, however, will be translated into Kripke structures.

rules $S \to Q \,||\, A, A \to P \,||\, A, A \to P \,||\, P$. The symbols S and A are non-terminals (S is called the starting symbol), that is, further rules need to be applied, while the processes Q and P are given as Kripke structures. The operator || defines a composition scheme which, in turn, results in a Kripke structure (state transition system, STS). A possible word resulting from this grammar could be of the form $Q \,||\, P \,||\, P \,||\, \cdots \,||\, P$. The terminal symbols represent the STSs related to the different processors. The grammar is used to specify how these STSs are composed to form the overall system, allowing the specification and verification of potentially unbounded systems. In the example, properties can be verified for systems with any number of P processes (as long as there are at least two of them).

After defining the network grammar and the properties under verification, the main idea for solving the verification problem of parameterized networks defined by grammars is to choose an appropriate *representative* for each symbol of the grammar, where each representative is a valid system that can be derived by the production rules. The verification method then model checks the representative chosen for the starting symbol; if this system satisfies the property then so does every system of the family. The soundness of the verification method is assured by the fact that all systems that can be derived from a symbol are simulated by the representative (of that particular symbol). Note that the model checking task can be performed by traditional model checkers. To make the approach more efficient an abstraction function is also proposed that maps the systems (i.e. Kripke structures) into smaller abstracted counterparts. Accordingly, the verification method consists of verifying whether the abstract representatives satisfy the property. Solutions to parameterized networks are usually less intuitive and often need considerable expert guidance to capture the symmetry of the problem in terms of the proper framework. In particular, finding feasible representatives might be a cumbersome task, since there is no deterministic way to do this (a heuristic is proposed in the work referred to above).

2.1.3 Abstract Interpretation

There has been extensive study of the use of data abstraction techniques, mostly based on the notions of *abstract interpretation* [Cousot and Cousot 1977]. Section 3 discusses abstract interpretation and the (more general) technique of symbolic execution. A general application of abstract interpretation in combination with model checking is presented in [Clarke et al. 1994]. The authors define a framework that allows temporal logic properties (CTL*) to be verified in the abstracted system using traditional model checking. In contrast to other works that likewise aim at static analysis of the system, this model checking framework allows observation of the (abstract) system variables during the entire execution of the (abstract) program.

2.1.4 Symmetry Reduction

Symmetry reduction [Clarke et al. 1993] is a control abstraction for finite concurrent systems exhibiting considerable symmetry. This approach aims at reducing the size of the state space that must be explored by the model checker by capturing the symmetry in terms of permutations of the states of the Kripke structure. The orbit relation $\theta(s)$ is an equivalence relation partitioning the set of states *S*, so that $\theta(s)$ is obtained from *s* by applying proper permutations. The idea of abstraction is that instead of modeling each state explicitly, one representative state for each orbit will be chosen (along the representative relation) and, accordingly, the abstract transition relations will be defined over these selected representative states. As an example we can consider an arbiter containing modules, each of them having a priority number. While competing for the bus, modules with higher priority numbers are given preference, and a winner is decided non-deterministically among competing modules with the same priority number. Here, the idea of a valid

abstraction could be that two modules with the same priority number can be permuted without changing the behavior of the system. Accordingly, the orbit relation partitions two global states into the same orbit if they can be transformed into each other simply by permuting the local states of the modules with the same priority number. As representatives, those states can be picked where e.g. the first module of the particular priority set is given the bus access.

2.1.5 A Novel Abstraction Approach

Amongst verification and validation (V&V) approaches, formal verification is seeing increased usage and acceptance. One popular technique is model checking that allows for automated verification (without user guidance) by performing exhaustive simulation on the model of the system. However, model checking faces the problem of state space explosion.

Abstraction is a general method to reduce the level of detail in order to reduce the complexity of analysis. Researchers at TU Darmstadt are working on a new technique that is tailored to the characteristic of a large class of distributed protocols **[Serafini et al. 2006]**. The basic observation is that many protocols exhibit a significant level of symmetry, which arises when all processes execute the same program and communication is synchronous and broadcast-based. Furthermore, most of the time the formulae to be verified are "plane", i.e., they have the form "all correct processes enjoy a certain property".

The straightforward modeling of a distributed protocol entails modeling the behavior of a single process and then composing the processes together. As a consequence, the state of the system is represented as a tuple of states, one for each process. Our approach is instead to consider all the possible states that any correct process can assume at each round. At each round, we calculate the set of all the possible states of any correct process due to symmetric message exchange with the other correct processes and to the presence of faults. This allows representation of the state of the system as a set instead of as a tuple of states, leading to a considerable reduction of the size of the state space. The verification of global properties is then carried out by checking if the desired properties hold for all the states that any correct process can have.

The key challenge is to prove that the abstraction process loses information but preserves certain properties. The abstraction is proved to be sound with respect to plane properties, i.e., if we can prove a plane property in the abstracted model then this also holds for the non-abstracted one. As we represent the set of states of any correct process, it is intuitively clear that our approach allows verification of properties that apply to each process individually, e.g., "each process shall have a variable within a given range". In this case it is sufficient to visit all the possible states of a correct process and to verify that they satisfy the property of interest. It is more complicated to prove *consistency* properties, e.g., "all processes shall decide the same value". For this purpose we partition the set of possible states of a process into *consistency sets*. At each round, it is guaranteed that if any two correct processes can have different states, these states belong to the same consistency set. The non-determinism in the evolution of the global system states determines the possibility of having several alternative consistency sets. Consistency properties are then required to hold only within each consistency set but not across them, e.g., "within each consistency set, all states shall have the same decision value".

For the sake of simplicity, the approach focuses on the verification problem for core fault tolerant protocols of frame-based, synchronous systems (e.g. consensus, atomic broadcast, diagnosis, membership). The approach, however, can be extended to model-checking of round-based asynchronous protocols.

Let us consider a scenario where such a protocol is symmetric (i.e., where each process performs the same operations) and there are no faults. If all processes start from the same initial state, they will perform the same state transitions at each round. Therefore, the state of the system could be modeled by representing only the common state of any process. Its evolution throughout rounds can thus be seen as a single thread. Each of these states is supposed to satisfy the desired properties of the protocol (safety and liveness).

We can expand this model by considering the presence of different, asymmetric initial process states. The state of a correct process will now evolve as a tree, where at each round the state of each process at round i is derived according to the state of the other processes at round $i - 1$. Each combination of states at round $i - 1$ results in different branches of the tree at round i. At each round, a correct process will find itself in one state among a set of multiple possible states; these are the children of a common parent node. These sets are called consistency sets. Similarly to alternative subtrees, consistency sets are mutually exclusive. As in the previous case, all the possible states at each round, i.e., all the states within each consistency set, are supposed to enjoy the specified properties.

When model checking fault tolerant protocols, however, the number of possible communication patterns and state transitions explodes due to the "extended" behavior of faulty nodes, particularly if the fault model allows them to communicate asymmetrically (e.g. under the Byzantine or the receive-omission fault assumption). This greatly increases the number of possible states and considerably reduces the symmetry of the system state. In our modeling approach, these further asymmetries are also handled using consistency sets. In fact, if the protocol is fault tolerant then the desired properties shall hold in spite of faults as well as of asymmetries in the possible initial states.

To understand the concept of consistency set we can consider a very simple example of a synchronous system consisting of three processors, where one of the two can be faulty and symmetrically send incorrect messages. Assume that the domain of the messages is {0, 1, 2}. Say that all correct processes start from state S1 and produce the message 0 as initial message in the first round of the protocol. There are three possible states of a correct process after the first communication round: S2, if both processes are correct and send the message 0; S3 or S4, if one of the processes is faulty and symmetrically sends the message 1 or 2 respectively. In this case we would have two consistency sets: {S2, S3} and {S2, S4}. Consistency sets are mutually exclusive and reflect the non-determinism given by the possible presence of faulty messages. By construction, they allow us to deduce that we will never have a situation where one correct process has the state S3 and another correct process has the state S4 (although a correct process can assume both the states S3 and S4).

Typically, abstraction techniques are defined considering general state models, for example Kripke structures. Such techniques are generally proved to be sound and complete and can be applied to model check any kind of specification, independent of the specific problem under examination. On the other hand, many papers reporting experiences on model checking distributed protocols introduce abstractions which are tied to the particular protocol and whose soundness and completeness are not shown but rather assumed as "sensible" (see for example [Steiner et al. 2004, Ramasamy et al. 2002]). Such an approach undermines the value of the results obtained by model checking. A contribution of our work is to introduce an abstraction scheme that, while specific for a class of distributed protocols, is rigorous enough to prove its properties mathematically. To achieve this, we first illustrate a modeling framework to express a distributed protocol (modified from [Dolev et al. 1986]). This somehow represents a higher level language if compared to normal specification languages used by model checkers. Then we show how to construct abstracted and non-abstracted models starting from the given specifications. Finally we prove that our abstracted models preserve soundness with respect to their corresponding non-abstracted models.

Using a specific protocol specification language, both the models are rigorously defined, allowing soundness and completeness to be proved. In contrast, much previous work on using abstractions to model check distributed protocols focused on defining the abstracted model. We claim that this approach should generally be applied when using domain specific abstraction, i.e. abstractions that are specific to a certain class of problems.

Currently, we are running *experiments* to measure the gain of the abstraction (in terms of state space size and proof execution time). One of our running case studies is a consensus protocol to solve *interactive (binary) consensus* [Lamport et al. 1982].

2.2 Process algebras and action-based model checking applied to fault-tolerant systems

2.2.1 Overview

In [Bernardeschi et al. 2000], process algebras and action-based model checking have been used for the formal specification and verification of fault tolerant systems. Process algebras have been recognized as useful for modelling the behaviour of a system. In process algebras, a system is seen as a set of communicating processes, and each process is characterised by the actions performed in the time by the process. The operational semantics of process algebras are Labelled Transition Systems (LTSs). The global behaviour of the system is expressed in terms of an LTS in which the transitions between states are labelled by actions. Action-based model checking is based on an action-based temporal logic (ACTL) [De Nicola and Vaandrager 1990] and ACTL models are LTSs. CTL, the branching time temporal logic most used in model checking, is based on predicates on the states. CTL models are Kripke Structures, that is transition systems where the states are labelled by a set of predicates [Clarke et al. 1986].

In [Bernardeschi et al. 2000], the formalization of fault tolerant systems is based on the following points:

- A system is modelled as a set of processes which communicate with each other and interact with the environment by executing actions;
- Faults are modelled directly by actions of the processes themselves. For each fault action, the relative failure mode is also specified. For example, a crash fault in a state extends the behaviour of the system by allowing a crash to occur in that state; when the system executes the action corresponding to the fault, a new state is reached which describes the failure mode;
- The fault tolerant technique is formally specified;
- Assumptions regarding the occurrence of faults are included in the specification by defining *ad hoc* fault assumption processes. Fault assumption processes allow the behaviour of the fault tolerant system to be studied under different fault scenarios – for example, a fault assumption could specify that at most one replica fails – and they realize a form of guided injection of faults into the system.
- Fault tolerance properties are specified as temporal logic formulae.

The formalization of a fault tolerance technique requires the use of the parallel composition, restriction and relabelling operators in order to conveniently express composition of redundant replicas and additional components. The addition of replicas increases the number of states of the system.

The main challenge when using model checking in practice is the number of states generated during the analysis, which challenges even the most advanced model checking tools.

The following characteristics of fault tolerant systems provide opportunities to limit *a priori* the state explosion problem:

- **Phased structure** of fault tolerant systems and algorithms. Indeed, a system employing architectural redundancy is composed of a number of identical modules which compute the same results. In today's embedded systems, such modules are often independent processors at the architecture level. These modules have to synchronize periodically in order to maintain their consistency, and the synchronizations are usually combined with some comparison or voting operation intended to detect or mask errors. An architecturally redundant system is therefore a distributed system that uses specialized interaction protocols. Usually such protocols have to ensure some notion of consistency even in the presence of faults, and trigger appropriate corrective actions when faults occur. The formal verification of such protocols is therefore an important step in the establishment of the overall correctness and safety of the system even in the presence of faults. Let S_i be the subsystem obtained by the composition (interleaving) of phase i of all the replicas. Due to the phased structure, the upper bound on the size of the state space for a system of n replicas is given by the maximum size of the state space of S_i, for any given phase i, which is significantly lower than the interleaving of the complete state space of the n replicas.
- **Regular structure** of a redundant system. Due to system replication, the regular structure of a redundant system may be exploited to contain state explosion with the help of established techniques, such as symmetries and reduction preorders.
- **Static configuration parameters**. Generally, the original semi-formal specification of an embedded system takes into account some attributes that can be considered as static configuration parameters and describe the particular type of entity that is controlled. As an example, a typical attribute for a level crossing entity says whether it is on the mainline or not. The semi-formal specification considers these attributes as if they were variables. This would contribute unnecessarily to the growth of the number of states of the model. Therefore, in the development of the formal specification we can take one configuration at a time. A safety property is satisfied if the property is verified in all possible configurations.

Model checking of a temporal logic formula, which gives an abstract notion of correct behaviour, can be applied to verify the correctness of a fault tolerant system design. When dealing with the correctness of the design, many situations arise in which it is enough to satisfy some “partial” aspects of the behaviour of the system. Temporal logic formulae represent a natural way to extract particular aspects of the system correctness.

The usability of the approach in real case studies is supported by the availability of automatic tools. The proof of temporal logic properties is based on the model checker in [Fantechi et al. 1998]. The approach has been proved usable for the verification of a railway interlocking system and of fault tolerant mechanisms [Bernardeschi et al.2002].

2.2.2 A railway interlocking system verification

The approach has been applied to a part of a railway signalling interlocking control system developed by Ansaldo Trasporti [Bernardeschi et al. 1998]. The system operates within a complex environment, interacting with a number of different actuators, sensors, and human operators. Sensors convey data concerning the physical status of the environment, actuators allow for the control of the operations and the

status of the external environment. An operator may interact with the system by sending commands and selecting modes of operation. The central Safety-nucleus is based on a Triple Modular Redundancy (TMR) configuration of computers implementing a two out of three voting scheme, with automatic exclusion of the unit in disagreement with the other two. The objective of this control system is to permit a safe passage of trains by adjusting the setting of signals on the railway line. The control system is represented by a set of communicating processes, modelling logical and physical entities. The control of the entities is realised by operations which act on variables. Often, variables represent signals whose domain of values is very limited or whose values of interest are limited in number.

The translation from the semi-formal specification (given in terms of variables and events) to the formal specification was straightforward. Using *ad hoc* techniques for testing signal values and combining the replicas of the TMR configuration, researchers at Pisa University have obtained a model of the behaviour of the system (consisting of one replica) of about one million of states. The static parameters allowed a reduction in the number of states of this global LTS from about one million states to 77294 states. Note that a reduction of the model of the system is also useful for the standard model checking approach.

2.2.3 Fault tolerant mechanisms verification

This framework has been used in the project GUARDS (Generic Upgradable Architecture for Real-Time Dependable Systems) to validate the Inter-Consistency mechanism which is the basis of *ad hoc* defined fault tolerance mechanisms [Bernardeschi et al. 2001]. The mechanism is a composition of transmitter and receiver protocols. The algorithm is modelled as a network of communicating processes, each modelling one of the nodes. Moreover, the algorithm has a phased structure: each of the previous processes is described by a network of communicating processes modelling the different phases of the algorithm and the local variables. The classical Agreement and Validity properties can be expressed by temporal logic formulae. In the following, a network composed of four nodes is considered. The size of the state space of the single node is shown, as is that of the network under various fault assumptions. The fault assumptions have been modelled by means of specific processes which constrain the occurrences of faults.

Model of:	**states**
A single non faulty node	428
Network of 4 non faulty nodes	3479
Network with an arbitrarily faulty node and a symmetric faulty node	109613
Network with an arbitrarily faulty node, and authentication violation	122767

The table above clearly shows that the size of the state space of the system with 4 nodes is much less than the fourth power of the size of the state space of a single node, which shows that the framework is effective at combating the state explosion problem. Fault assumptions have been used to reduce the state space.

2.3 Model Checking for Exhaustive Fault Simulation

Design of fault-tolerant distributed real-time algorithms is notoriously difficult and error-prone: the combinations of fault arrivals, interleaving of concurrent events, and variations in real-time durations lead to a case explosion that taxes the intellectual capacity of human designers. These difficulties are compounded when optimizing numerical parameters — e.g., when seeking to determine a minimum safe timeout, or the time required to stabilize after an upset. In an idealized world, algorithms are derived by a systematic process guided by formal correctness arguments. However, in contemporary reality, designers generally have an

informal argument in mind and develop the final algorithm and its parameters by mentally exploring local variations against that argument and against scenarios that highlight tricky cases. Exploration against scenarios can be partially automated using a simulator or rapid prototype and such automation may increase the number of scenarios that can be examined and the reliability of the examination.

Automated examination of scenarios can be taken still further using model checking. In model checking, the case explosion problem is transformed into one of state explosion—meaning that the time and space required to run the model checker grows rapidly and eventually becomes infeasible as the size of the model grows, so that abstraction, or consideration of only limited numbers of fault cases and real-time delays, must be employed. When using model checking in the design loop, the challenge is to cover a usefully large number of scenarios in a very short time (say a few minutes), so that the designers can perform an interactive exploration of the design space without losing concentration or patience. As a design becomes consolidated, attention shifts from exploration to verification and the challenge for model checking becomes one of covering a truly exhaustive set of scenarios for a realistically accurate model in reasonable time (say, overnight).

The increasing performance of modern model-checking tools offers high potential for the computer-aided design of fault-tolerant algorithms. Instead of relying on human imagination to generate taxing failure scenarios to probe a fault-tolerant algorithm during development, the fault behaviour of a faulty process can be defined at its interfaces to the remaining system in order to enable the use of model checking to automatically examine all possible failure scenarios. In [Steiner et al 2004], this approach is called "exhaustive fault simulation": it provides a "dial", so that a single model can be used in model checking for both rapid exploration and exhaustive verification. The authors illustrate exhaustive fault simulation using a new startup algorithm for the Time-Triggered Architecture (TTA) and show that this approach is fast enough to be deployed in the design loop.

The TTA startup algorithm is fairly subtle and must cope with many kinds of fault and timing behaviours. Model checking provides a way to explore these behaviours in an automatic way, but faces certain difficulties. First, the algorithm involves time in an essential way and the most realistic formal model for the algorithm will be one in which time is treated as a continuous variable. Timed automata provide a suitable formalism of this kind, and are mechanized in model checkers such as Kronos and UPPAAL. However, model checking for timed automata is computationally complex, in fact sufficiently complex that the case/state explosion caused by considering a large number of fault scenarios rapidly makes the model computationally infeasible. Initial experiments did use timed automata and the authors of [Steiner et al. 2004] were unable to consider more than a very few simple kinds of fault.

The utility of model checking for exploration and verification of fault-tolerant algorithms critically depends on considering a large number of different kinds of fault; ideally, one would like the fault model to be *exhaustive*, meaning that one describes every kind of fault one can think of, and lets the model checker inject these in all possible ways. Since this is impracticable in a model that uses continuous time, the authors of [Steiner et al. 2004] looked for an abstraction employing discrete time.

Nodes executing the startup algorithm measure time by counting off slots in a Time Division Multiple Access (TDMA) schedule (cf. Sect. 1.1.1). Although slots have duration and may be offset at different nodes, one can think of them as indivisible units: it is irrelevant by how much the slots at different nodes are offset, just whether they overlap at all (so that a collision can occur). Thus, one can use a discrete notion of time and can model the collective behaviour of a cluster of nodes as the synchronous composition of discrete systems. Another way to justify this modelling approach is to think of it as describing the system from the

point of view of a central guardian: each discrete instant corresponds to some real time interval at the guardian and all messages that (start to) arrive in that interval are regarded as simultaneous; the behaviour of the nodes is driven off (i.e., synchronously composed with) the discretisation provided by the central guardian.

Faults vastly increase the state-space that must be explored in model checking, and they do so in two different ways. The first way is by introducing genuinely different behaviours; in [Steiner et al 2004], a fault degree "*dial*" is introduced to parameterize the various behaviours. The idea is to use as high a degree (i.e., as many kinds) of faults as prove feasible for model checking in practice. To this end, the node model describes a faulty node as one that can send arbitrary messages in each slot. The possible outputs of such a faulty node are classified into six fault degrees. For example, a fault degree of 1 allows a faulty node only to fail silent, while fault degree 6 allows a node to send an arbitrary combination of messages with correct or incorrect semantics, noise, or nothing on each of the two TTA channels.

The second way faults increase the state space is to introduce "clutter" in the form of states that differ in irrelevant ways: for example, a faulty node can end up in one of many different states, but once the correct components have excluded this node from further consideration, its state has no effect on system behaviour. However, a model checker distinguishes all the different states of the faulty component and this needlessly complicates its task. A valuable "trick" in modelling fault-tolerant algorithms is to set the states of faulty components to fixed values once they can no longer affect the behaviour of the system. In [Steiner et al 2004], this is implemented by a mechanism called *feedback*.

The resulting models have billions or even trillions of reachable states, yet the symbolic model checker of SAL is able to examine these in a few tens of minutes (for billions of states) or hours (for trillions). The combination of an effective modelling approach and an efficient tool allowed the authors of [Steiner et al. 2004] to use model checking over an exhaustive fault model in the design loop for the algorithm, and also helped them establish the worst-case startup times. Thus, the approach extends previous experiments in model-checking fault-tolerant algorithms by vastly increasing the number of scenarios considered, while achieving performance that allows the method to be used in design exploration as well as for verification.

Possible future formal methods studies might explore the use of infinite-bounded model checking techniques, which combine a SAT solver with decision procedures for theories including real arithmetic, to develop and analyze models that use continuous time, while still allowing rich fault models. Furthermore, theorem proving tools such as PVS could be used to formally verify the algorithm and its fault hypothesis in their most general forms.

2.4 Case Studies in refinement-style Model Checking

This section summarizes several verification case studies performed by QinetiQ using the process algebra "Communicating Sequential Processes" (CSP) [Roscoe 1998] and the refinement-style model checker FDR [FDR 2006] for CSP.

Where used by particular analyses, specific techniques for combating state explosion are mentioned below. For most of the analyses, various techniques were used to model the systems in ways that avoid unnecessary duplication of states when compiling a CSP model to an internal transition system. Many of these techniques are discussed in [Roscoe 98] and are well-known within the CSP/FDR research community, so they are not described below.

2.4.1 Verification of Selected MAFTIA protocols

This subsection summarises refinement-style model checking of selected mechanisms and protocols developed by the EU project MAFTIA [MAFTIA, **Veríssimo et al. 2006**]. This verification work was undertaken by QinetiQ in collaboration with IBM Zurich Research Laboratory and the University of Saarland. In all cases, the refinement-style model checker FDR [FDR 2006] was applied to CSP models. Section 5 describes IBM's research, within MAFTIA and subsequently, on combining cryptographic and formal techniques. Detailed information regarding the MAFTIA protocols is provided in the "Arch" part of this document.

2.4.1.1 Contract-Signing Protocols

Using CSP modelling and refinement-style model checking, QinetiQ partly verified the Contract-Signing protocols adopted by the EU project MAFTIA [Creese et al. 2003]. These protocols enable two parties – the 'sender' and 'receiver' – to agree irrefutably to a contract using the services of an independent 'verifier' and Trusted Third Party (TTP). There are two versions of the protocol, depending on whether the communication links are considered asynchronous with eventual delivery or fully synchronous. Both versions run under various threat models.

The modelling of the protocols used quite standard CSP modelling techniques: each of the four main roles of the protocol was modelled as a separate process and these were put in synchronised parallel with a process representing an integrated 'spy' and communications network. The final version of the CSP model revealed that stated desired safety properties of the protocol were fulfilled; this was done using CSP's traces model. One of the requirements not verified was an availability requirement in the asynchronous version of the protocol under the assumption of 'eventual delivery' of 'success' or 'abort' messages. Analysis of eventual delivery is described below.

2.4.1.2 Trusted Timely Computing Base (TTCB)

The Trusted Timely Computing Base (TTCB) is a security kernel composed of a number of distributed, trusted components (local TTCBs) connected via a secure, synchronous 'control network'. QinetiQ verified two of the TTCB's security services – the Local Authentication Service (LAS) and the Trusted Block Agreement (TBA) – as reported in [Adelsbach et al. 2003]. Just as for the Contract-Signing protocols, the TTCB was modelled as a number of separate CSP processes, in this case representing the kernel, the local TTCB, and the 'spy' integrated with the payload network.

No attacks were found on the security properties stated of the LAS and TBA. However, the analysis did reveal an issue regarding the sending of an 'Entity ID' unprotected to the entity by the local TTCB as part of the LAS. In the short term, the spy could exploit this to cause repeated mismatches and thus a denial-of-service. The issue was raised with the authors of the LAS who suggested amendments to protect the Entity ID.

2.4.1.3 Synchrony Models and the Modelling of 'Eventual Delivery'

MAFTIA recognised four main synchrony models, namely: fully synchronous, fully asynchronous, hybrid (as in the TTCB synchronous control network + asynchronous payload network) and asynchronous with eventual delivery. The last of these synchrony models assumes that all messages sent between 'honest'

parties are delivered 'eventually' relative to any other discrete events. QinetiQ studied this synchrony model in some depth and a CSP model was developed that allowed for the verification of protocols running under the model. This, model, however, was expensive in terms of state space, which limited its use within MAFTIA.

Although eventuality can be specified by CSP's failures-divergences model, one cannot model explicitly a communications medium in which messages are guaranteed to arrive 'eventually', where 'eventuality' is relative to the occurrence of other discrete events [Leuschel et al. 2001]. It was therefore necessary to formulate the problem strictly in terms of the protocol, not its environment. A novel feature of the approach was the reformulation of the question, weakening it to the point where it can be checked of a finite state model. The following alternative question was posed: is there anything intrinsic in the design of the protocol that would prevent satisfaction of its success criteria? This means: is the protocol guaranteed to meet its success criteria in an environment where the communications medium cannot indefinitely delay messages? This was achieved using watchdog processes. It may be possible to achieve the same effect by using FDR's fair divergence checking capability. See [Creese et al. 2003] for further details.

2.4.1.4 Miscellaneous Other Work within MAFTIA

In addition to the main verification items listed above, QinetiQ undertook other CSP analyses for MAFTIA, including analysis of a probabilistic asynchronous Byzantine Agreement protocol and of an Ideal System for Group Key Agreement (GKA); these met with mixed success. An important obstacle was the size of the models that could be analysed by FDR, and a notable success here was a technique for collecting an arbitrarily large number of votes data independently (where the properties of interest depend only on the proportions of votes cast). See [Creese et al. 2003] and [Adelsbach et al. 2003] for further details.

2.4.2 Formal Assessment of Bluetooth

This section is a summary of an analytical assessment of Bluetooth security mechanisms present in the Bluetooth v1.2 standard involving formal analysis of selected Bluetooth protocols. This work was performed by QinetiQ's Systems Assurance Group as part of the FORWARD project [FORWARD], in collaboration with project partners and QinetiQ's penetration testing team. It used the process algebra Communicating Sequential Processes (CSP) [Roscoe 1998], the security protocol analysis front-end Casper [Lowe 1998, Lowe 1999], and the refinement-style model checker FDR [FDR 2006].

2.4.2.2 Analysis of Bluetooth using Casper

Casper was used to verify the Bluetooth pairing procedure. Casper is a CSP front-end designed primarily for the analysis of two-party security protocols. Casper input files use the 'security protocol language', a formal grammar based on a *de facto* 'standard notation' used in academic literature. Each input file describes: (i) a security protocol defined in terms of the 'roles' played by agents; (ii) a 'system' of honest agents running that protocol in the presence of an Intruder who tries to 'break' the protocol with respect to a set of required security properties; and (iii) a formal specification in terms of the set of security properties that should hold of the system. Casper compiles 'security protocol language' files to files containing CSP processes representing the system (including the Intruder) and the security properties to be checked. These processes can then be compared using FDR [FDR 2006].

There are five distinct stages in the pairing procedure. In turn, these are: generation of the (initial) *link*, *unit* and *combination keys,* authentication of the current *link key*, and generation of an *encryption key.*

Also, the use cases of the pairing procedure vary according to a number of parameters, including: the assignment of agents to roles in any particular stage of the protocol (none of the stages are symmetric in the roles), the number of protocol runs per role, the session variables including the Intruder's initial knowledge at the beginning of each stage of the protocol, and whether the protocol is run in parallel or sequentially.

As a result, there are many possible use cases to be considered. To help manage the complexity, QinetiQ developed a tool called Caspose, which is essentially an intelligent Casper pre-processor; it takes a number of small Casper scripts representing the individual stages of the pairing procedure and amalgamates them, outputting multiple self-contained Casper scripts corresponding to different use cases. Although written for the Bluetooth pairing procedure, Caspose is applicable to the Casper analysis of general protocols or protocol stacks that can be broken up into a number of individual stages.

Results of the analysis

The analysis was question driven. Bluetooth v1.2 does not formally assert any properties of the Link-Layer security. Instead, the formal analysts drafted properties based on a number of questions informed in part by a separate penetration testing phase. Some examples of these questions follow:

- Is it possible for an initial link key to be deduced by the Intruder without any initial information? If the answer is 'no', then what information would be required by the Intruder, and when, for him to be able to deduce the key?
- What knowledge is required by the Intruder to cause the authentication of a link key to fail?
- Is it possible that an encryption key can ever to be deduced by the Intruder?
- Can the Intruder spoof one party in the pairing process?

Using Caspose, a sizeable, but not complete, subset of about forty use cases of the pairing procedure were considered in relation to the above questions and others. These use cases covered most of what could reasonably be considered as the 'normal usages' of the procedure. No definite protocol attacks were discovered that did not assume that the Intruder possessed some prior information. However, it was shown that, with some capabilities in addition to those which the Intruder would routinely be expected to have in practice, the Intruder may be able to break or otherwise compromise the protocol. These additional capabilities include the ability to make an effective (verifiable) guess on a shared PIN, or to spoof the identity of a Bluetooth device. These are capabilities that can be attained quite feasibly, according to the findings of the analysis undertaken by the penetration testers, and they can lead to failures of secrecy and authentication, some quite subtle. See Section 4 of [Azeem et al. 2004] for more details.

2.4.3 Formal Analysis of the Grid Routing Protocol

As part of the UK DTI-funded FORWARD project [FORWARD], analysis methodologies suited to verification of key properties of mobile routing protocols were developed. We summarise the methodologies developed by QinetiQ, in collaboration with project partners, for black-and-white correctness analysis (involving properties that are predicate, so either true or false) and for performance analysis (involving properties that hold with some probability other than 0 or 1). We outline the application of these methodologies to the Grid routing protocol.

2.4.3.1 Grid

Grid [Aguayo et al. 2001] is a routing protocol intended to provide scalable routing for large ad-hoc mobile networks. It requires no fixed infrastructure and is designed to be deployed rapidly. It was designed as part of the MIT Oxygen Project (oxygen.ai.mit.edu), focussed on enabling pervasive human-centred computing through a combination of specific user and system technologies.

Grid uses the Grid Location Service (GLS), a mechanism by which nodes learn the positions of others, together with Geographic Forwarding (GF), a simple technique for routing messages to nodes at geographic locations specified by the senders. GLS consists of a bootup phase followed by a query/reply phase.

GLS bootup aims to establish distributed knowledge of node locations, according to a static subdivision of 2-D space into a hierarchical 'grid' of squares (4 top-level squares are each composed of 4 smaller squares, and so on down to some depth). It operates in a series of rounds; each round is intended to establish location servers one step further away in the grid hierarchy from the nodes whose locations they store, compared to the location servers established in the previous round.

The query/reply phase uses and maintains the knowledge established during bootup. A query is sent when a node wants to learn the current location of some target node. Queries are forwarded to neighbouring nodes that (are believed by the forwarding nodes to) know the desired location more accurately, until a node is reached that knows the location accurately; a reply is then sent back to the original requester, again forwarded suitably.

2.4.3.2 Correctness analysis

The correctness analysis methodology is described in [Bolton et al. 2003]. Separate CSP models of GLS and GF were written, and these were analysed using the FDR refinement-style model checker. The GLS analysis considered both the bootup phase and the query/reply phase.

Briefly, the approach taken was to model generic 'GLS' and 'GF' node processes, and rename these to represent GLS or GF functionality of the particular nodes of a notional small network. Using CSP's process operators, the renamed nodes were then composed in parallel, synchronised suitably on query and reply messages. Notable techniques employed in the modelling include: the use of configuration parameters for easy comparison of alternative algorithms; arbitrary static modelling to enable verification of properties for *all* static configurations of a system; use of a natural recurrence relation to express global correctness properties of GF; and use of CSP's multiway synchronisation to enable efficient modelling of complex patterns of communication.

The Grid correctness analysis assumed ideal operating conditions: no node movement and no message loss. (These shortcomings were addressed in the performance analysis, described in the next section.) Since it was possibilistic in nature, it would have been difficult to assume anything else and still obtain useful results. It yielded the following results for small static networks under this assumption:

- it characterised in precise terms when GF successfully delivers packets;
- it identified an ambiguity in the published description of Grid [Aguayo et al. 2001], demonstrating that key properties were not satisfied by one possible interpretation;
- it proved correctness of both phases of GLS when an alternative interpretation is used.

Overall, the analysis demonstrated that refinement-style model checking can provide insight into how systems work and can verify desired black-and-white correctness properties of those systems.

2.4.3.3 Performance analysis

The performance analysis methodology is described in [Moffat et al. 2005]. It addressed the task of assessing how environmental factors affect "correctness" and "efficiency". The latter covers issues such as the relative speed of restabilisation after network changes and the quantification of the overhead cost associated with attempting to maintain a correct view of the network topology in spite of network changes/message loss.

This methodology was specifically designed to complement that for black-and-white correctness analysis. It makes use of probabilistic models derived from the possibilistic CSP models used in the correctness analysis outlined above. Accordingly, this methodology allowed more realistic characterisation of, for example, GLS bootup. In particular, it characterised GLS performance by calculating accurate minimum and maximum bounds on the probability of a specified condition being satisfied.

The methodology includes 'bolt-on' guidance for obtaining probabilistic models systematically and easily, starting from the possibilistic models. The guidance describes how probabilistic annotations can be added to the (possibilistic) CSP models of GLS produced for the black-and-white correctness analysis. This yields models expressed in a probabilistic extension of machine-readable CSP, called pCSP_M. Subsequent steps were described for analysing such models, using a specially-built extension of the FDR model-checker to translate the models into the PRISM language and the PRISM tool. (The pCSP_M language and the probabilistically extended version of FDR were themselves developed as part of the FORWARD project; see [Goldsmith and Whittaker 2005].)

The PRISM tool [Hinton *et al.* 2006] is a probabilistic model checker. Probabilistic model checking is an algorithmic procedure for deriving formal quantitative properties of a given probabilistic system. These properties include, for example, "what is the worst-case probability of a system failure occurring within the first two hours of operation?" and "what is the maximum expected power consumption of a particular component before all messages are successfully delivered?". It can therefore be viewed as quantitative verification; it is related to evaluation, which is addressed in the "Eval" part of this document.

The Grid performance analysis allowed various forms of message loss, and obtained limited results for dynamic networks. The results obtained for GLS validated the approach. They made sense intuitively and agreed with performance predictions QinetiQ had made based on hand calculations. Importantly, the approach is not restricted to the simple cases for which predictions had been made.

2.4.4 Formal Analysis of OceanStore elements using ModelWorks

Here we summarise QinetiQ's verification of selected elements of the OceanStore distributed storage mechanism. This work was performed as part of the FORWARD project [FORWARD].

2.4.4.1 OceanStore

OceanStore [Kubiatowicz et al. 2000] is an architecture for a global-scale, persistent distributed storage mechanism. It was designed to be constructed from largely untrusted infrastructure, to be resilient to server crashes and to avoid information leakage to third parties. OceanStore supports nomadic data, i.e., data that is allowed to flow freely.

2.4.4.2 Dependability Library and ModelWorks IDE

ModelWorks is the graphical front-end to QinetiQ's dependability library [Simmonds and Hawkins 2004, Simmonds and Hawkins 2005]; it is an evolving framework developed to support the analysis and verification of dependable distributed systems. It aims to deskill the modelling of such systems to a large degree and, in so doing, reduce costs. The main elements of ModelWorks are: (i) a graphical notation for describing the network topology of systems; (ii) simple process description languages for describing the processing of information at individual nodes; (iii) support for various forms of specification, including assumption-commitment style reasoning; and (iv) translators that take models described and specified in the languages of (i)-(iii) and output models in various target languages, currently CSP.

2.4.4.3 The Modelling

The OceanStore elements modelled were the Update and Query Algorithms. These are the mechanisms by which nodes in OceanStore update when a new piece of data is added and perform location queries on data. Both mechanisms use attenuated Bloom filters [Bloom 1970] to associate with each node the probabilities of finding any given piece of data through neighbouring nodes. Bloom filters are efficient but 'lossy'.

The Update and Query algorithms were modelled for small three-node loop topologies with various distinct arrangements of data in the system. Under certain assumptions, including that the Bloom filter hash functions were collision-free, it was shown that the algorithms terminate successfully [Simmonds and Hawkins 2004, Simmonds and Hawkins 2005]. These papers include detailed technical discussion regarding how ModelWorks could be improved to support 'structural induction' techniques, by which results such as those above for the small three-node topologies could be extrapolated to arbitrarily large systems.

Conclusions

Model checking is currently an important verification technique, and we expect it to be useful for verifying the resilience of current and future ubiquitous computing systems. ReSIST partners (specifically Pisa University, Technical University of Darmstadt, Ulm University, and QinetiQ) have extensive experience in the area of model checking. The range of analyses summarised in this section indicates that model checking can be useful for a wide variety of applications relevant to ubiquitous computing.

The state space explosion problem is the principal limitation of model checking: time and space (computer memory) resources are the main obstacles when attempting to analyse problems of realistic size.

Further efforts to address the state explosion problem are required. For example, the Grid correctness analysis (Section 2.4.3.2) was performed assuming ideal operating conditions: no message loss and no node movement. Although the subsequent performance analysis (Section 2.4.3.3) allowed various forms of message loss, and obtained some results for dynamic networks, the state explosion problem meant that these analyses produced results of limited use, because only a small number of nodes could be considered. The current technology is far from capable of addressing large ad-hoc mobile networks. Similar limitations can be expected when model checking is applied to future ubiquitous computing systems.

Ubiquitous systems will be large and widely distributed. The sheer numbers of components and their interactions are likely to make these systems very complex. Any techniques that abstract away from insignificant differences between components, or insignificant details of the components themselves, are likely to be helpful. Several of the techniques discussed above can be expected to alleviate state explosion in

future applications of model checking. These techniques include exploitation of symmetry, the parameterized networks technique, data abstraction, and use of data independence.

3 – Symbolic Execution and Abstract Interpretation

Introduction

Symbolic execution is used to describe a range of techniques for analysing computer programs. Symbolic execution techniques represent elements of program state by symbolic representation of their values. These are then manipulated in ways that reflect the behaviour of the program on corresponding program states. A narrow definition of symbolic values would insist that they represent the manner in which the values have been derived.

A key challenge in symbolic execution is to arrive at a suitable abstraction – i.e., to know which aspects of the program to abstract away. This obviously depends on the program properties of interest. Pragmatic, heuristics-based approaches exist for determining the granularity of abstraction to apply. For example, one could start by considering a detailed model and give up if analysis does not terminate within some resource (space or time) limit; if this does not succeed, one could use a more abstract model, and so on. There is no general guarantee that this simple strategy will succeed; more sophisticated approaches are needed in practice.

Abstract Interpretation is a general theory that can be used to define correct abstractions and/or to study properties of abstractions. Abstract interpretation of imperative programs was introduced by Sintzoff [Sintzoff 1972]. A seminal paper by [Cousot and Cousot 1977] placed the approach on a firm footing by generalising several ad hoc data flow analysis techniques into a unified framework. This was further developed in [Cousot 1981], [Cousot and Cousot 1992] and [Nielson 1982, 1989].

Abstract interpretation allows all possible behaviours of a program to be predicted approximately. Prediction is approximate in the sense that some extra behaviour, not actually possible for the program, might be included. This provides a way of verifying and deriving properties of interest, since properties that hold of all predicted states also hold of all actual states. A very simple example is to consider only whether integer variables have odd or even values; one can then analyse the program while only maintaining this ‘parity’ information. Results obtained will be valid for the original program, though much information will obviously have been lost. Approximation is, of course, a necessary price to pay for analysis to be guaranteed to terminate.

The following sections describe work by ReSIST partners concerning techniques for program analysis related to symbolic execution and applications of abstract interpretation to verification problems. These techniques may be viewed as supporting verification.

3.1 Program Slicing

3.1.1 Overview

Program slicing is a decomposition technique that extracts from programs those statements that are relevant to a particular computation. Informally, a slice is the answer to one of the following questions: “What program statements (potentially) affect the computation of variable v at statement S?” or “What program statements are (potentially) affected by variable v at statement S?”.

This technique can be used as a precursor to other types of analysis, e.g. program debugging, testing, parallelization, integration, software safety, understanding, software maintenance, software metrics. It can allow subsequent analyses to focus on only those parts of the program that matter to the properties they are investigating; thus, one may hope that program slicing can reduce the effort needed to perform the subsequent analyses.

A slice is taken with respect to a slicing criterion. In its simplest form, a slicing criterion specifies a single location (statement S) and variable (v) of interest. More generally, a slicing criterion can specify a collection of statement/variable pairs.

Three major ways of classifying program slices are as follows:

- Executable vs. Non-executable
  An executable slice forms an executable program. Most techniques yield executable slices.
- Backward vs. Forward
  A backward slice is computed by following the edges of a dependence graph (or control flow graph) backwards. A forward slice is computed by following edges forwards. Strictly, the adjectives forward and backward apply to the manner in which the slice is computed, rather than to intrinsic properties of the slices.
- Static vs. Dynamic
  Static slices are calculated without knowing particular input values. Dynamic slices are computed for given fixed inputs. As above, strictly, the adjectives static and dynamic apply to the manner of calculation. These words are used because up-front static analysis does not have particular run-time values available, whereas run-time dynamic analysis does.

Program slicing algorithms are generally of two sorts: those based on control flow graphs (CFGs), and those based on program dependence graphs (PDGs) or system dependence graphs (SDGs). These terms are defined below.

3.1.2 Slicing using a control flow graph

A control flow graph is a directed graph with nodes representing statements and arcs representing control flow between them. Each node n is labelled by the variables referenced at (the statement represented by) n, and the variables defined at n.

Computation of a slice from a CFG is done in three steps: (1) create the CFG; (2) compute data flow information as sets relevant(n) of variables at each node n; and (3) extract the slice [Binkley & Gallagher 1996].

3.1.3 Slicing using a program/system dependence graph

A program dependence graph is a directed graph with nodes representing assignment statements or control predicates, and arcs representing control or flow dependence explicitly (which are explained below).

Control dependence edges are each labelled by TRUE or FALSE. Such an edge from u to v means that whenever the predicate represented by u is evaluated, and its value matches the label, the program component represented by v will eventually execute if the program terminates. A flow dependence edge

from u to v means that the program's computation might be changed if the relative order of the components represented by u and v were changed.

PDGs are well-suited to intra-procedural slicing; once the PDG has been generated, intra-procedural slicing amounts to a simple reachability problem over the PDG. To slice across procedure boundaries, system dependence graphs are used, as explained next.

A system dependence graph is a generalisation of a PDG that represents multi-procedure programs. Compared to a PDG, an SDG additionally represents procedure calls, procedure entry, parameters and parameter passing. These are represented using 'summary edges' between actual-in and actual-out parameter nodes.

3.1.4 Current Applications

Program slicing has several applications [Binckley & Gallagher 1996]: *Differencing, Program Integration*, *Software Maintenance*, *Testing*, *Debugging, Software Quality Assurance*, *Reverse Engineering,* and *Functional Cohesion* (measuring the 'relatedness' of program components).

Of these, the most relevant to ReSIST are testing and software quality assurance. Program slicing can be used to reduce the amount of regression testing needed after modification of a program. Regarding software quality assurance, slicing can be used for locating safety-critical code (locating all code that may contribute to the values of variables in components deemed safety-critical), defending against common mode errors by validating functional diversity (verifying that there are no interactions of safety-critical components with non-safety-critical components), highlighting program slices for further analysis/inspection/testing, simplifying the auditing task by dynamic slicing (using particular, hopefully representative, input values) and then auditing the resulting relatively simple slices.

3.2 Type Based Analysis

3.2.1 Overview

Type based analysis uses either an explicit or an inferred type system of the source language to provide assumptions (properties) that can be used to simplify the modelling of program effects. This was the initial motivation for the study of type and effect systems, and it has since been shown that using this approach can improve the efficiency of some algorithms [Palsberg 2001].

A type system essentially ensures that parts of a program's state always satisfy some invariant. An effect system essentially models some aspect of the program's execution, such as whether execution of an operation can cause an exception to be raised, or whether it can cause some (external) resource to be updated. Type and effect systems combine these two approaches, by annotating the typed state transitions. Having said this, the literature contains annotated type systems that are similar to effect systems, such as the "Annotated type constructors" of [Nielson et al. 1999, Section 1.6.1]. So, in practice, it is hard to cleanly separate annotated type systems from effect systems.

Type systems are typically formalised in terms of logical type judgment rules, which provide a mechanism for determining the type of a valid expression. In this context type checking amounts to ensuring that each expression has the expected (required) type.

Some combination of unification and constraint checking algorithms underpins most type checking systems. For some simple languages the unification algorithm amounts to equality testing and the constraint checking amounts to the success (or failure) of unification. However, most modern programming languages require more sophisticated unification techniques, such as those based on partial orders. For example, subtyping judgment rules essentially construct a partial order over types. In this context, unification and constraint algorithms can be configured to check whether a candidate subtype is allowed.

Semantic properties of a program can sometimes be inferred along with the more usual type information; in such cases, suitably annotated types can be inferred by application of extended type inference rules. An example of this approach is the use of index types (a variant of dependent types), as shown by [Xia and Hook 2000].

To cope with issues such as recursive types, the above approaches can be justified by appealing to a fixed-point theory.

It has been shown that type systems can be characterised (defined in terms of) abstract interpretations [Cousot 1997]. It has also been shown that there are some non pure-type analyses where type analysis is as powerful as abstract interpretation. For example, [Jensen 2002] demonstrates that his type-based strictness analysis is equivalent to the standard abstract interpretation strictness analysis. Pragmatically, being able to show equivalences between different formal analysis frameworks has the potential to enable techniques and tools applied in one to be used for demonstrating that a given property holds in the other framework (because they exist, are more efficient, etc.).

3.2.2 Current Applications

The current applications of type-based analysis include: control flow analysis, method in-lining, application extraction, redundant load elimination, encapsulation checking, race detection, memory management, side effect analysis, (information) flow analysis, security flow analysis, exception analysis, and region analysis [Palsberg 2001].

3.3 Abstract interpretation

3.3.1 Overview

Abstract interpretation is a method for analyzing programs in order to collect approximate information about their run-time behaviour.

An abstract interpretation is defined as a non-standard, approximated semantics (abstract semantics) obtained from the detailed standard semantics (concrete semantics) by substituting the actual (concrete) domains of computations and their basic operators with abstract domains and corresponding abstract operators.

Abstract domains are representations of some properties of interest of values in the concrete domain, while abstract operations simulate the behaviour of the corresponding concrete operations. The notion of approximation is encoded by suitable partial orderings on objects in the domains. The orderings on the concrete and abstract domains describe the relative precision of domain values. Concrete and abstract universes are related by a concretization function and an abstraction function.

Using this approach different analyses can be systematically defined. Moreover, the proof of the correctness of the analysis can be done in a standard way. Correctness assures that the properties encoded in the abstract domains checked on the abstract executions are guaranteed to be true of real executions.

3.3.2 Exception Analysis

QinetiQ's MALPORTE tool investigates the means by which software, written in C, C++ or Ada, can raise run-time exceptions or have undefined behaviour; it then derives system constraints that would prevent these occurring. The approach is to consider how sets of ranges of values can be used as an abstraction for use in exception analysis [Whiting and Hill 1999].

An advantage of abstract interpretation is that it does not generally require any code annotations. MALPORTE does not require code annotations, but it is able to make use of optional user-specified 'restrictions' on intermediate values (e.g. a certain value communicated by one component to another might be restricted to some particular range). Restrictions can improve the precision of analysis results, sometimes significantly. Importantly, restrictions are checked by MALPORTE for intermediate values that are output by an analysed software component. (If a restriction is claimed for an intermediate value output by a hardware component, then a separate justification would be required.)

MALPORTE is based on abstract interpretation of programs written in an intermediate language: the Architecture Neutral Distribution Format (ANDF). The approach is therefore applicable to the range of languages for which translators to ANDF are available.

A number of other tools adopt generally similar approaches, including Polyspace [Polyspace 2006, Hote 2004, Hote 2005] and Cousot et al.'s ASTREE tool [ASTREE 2006, Cousot et al. 2005]. The ASTREE tools web site [ASTREE 2006] claims two significant results: first, that it was able to automatically prove the absence of runtime exceptions in the fly-by-wire flight control software of the Airbus A340; and second, that the analysis was extended to include electric flight control codes of the A380.

3.3.3 Secure Information Flow Analysis

In [Barbuti et al. 2002] abstract interpretation was applied to the analysis of secure information flow of programs [Sabelfeld and Myers 2003] written in a simple high level language. The work assumes systems implementing a multilevel security policy, where the security levels form a lattice. The approach is based on abstract interpretation of the operational semantics [Schmidt 1996]. Values carry a security level, which may change dynamically during the execution of the program. Moreover, implicit flows are modelled by an environment, upon which the instructions are executed. The environment contains information on the open implicit flows: it is updated when an implicit flow begins, and it is restored on termination of the implicit flow. An enriched semantics was defined which handles the flow of information among variables, in addition to execution aspects. The semantics was proved to correctly describe the actual information flow within programs. An abstract semantics able to ensure security of all executions was defined. Both semantics are operational and are based on the same set of rules: each one is obtained by a suitable redefinition of the domains on which the rules operate.

[Barbuti et al. 2004] studied secure information flow of programs written in a stack based assembly code. This security problem for machine code is characterised by the following points: 1) since the instructions operate on the operand stack, explicit flow of information must keep track of the security level of values pushed on and popped off the operand stack; 2) The sequential control flow of the program is modified by

conditional and unconditional branching instructions, jumping to any program point. This makes it more complicated to find the scope of implicit flows with respect to structured languages, where the scope of a conditional branching instruction coincides with the syntactic scope of the instruction itself; 3) the operand stack may be manipulated in different ways by the branches of a conditional instruction: they can perform a different number of pop and push operations, and with a different order. Therefore the influence of the implicit flows onto the operand stack must also be taken into account; 4) the length and the contents of the stack when the program terminates may be a means by which security leakages can occur.

Finally, in [Bernardeschi et al. 2004] concurrent programs consisting of a number of independently executing sequential processes with private memory are analysed. Communications between processes are synchronous: processes synchronize by explicit sending and receiving primitives. Open systems that can accept inputs from the environment by means of a set of input channels and produce outputs to the environment by means of a set of output channels are considered. Each input or output channel is associated with a security level. The absence of information flow is checked from the input channels to output channels having a lower security level. In addition to explicit information flow caused by the assignment command, explicit information flow due to input and output commands must be considered: for example, the command *output(x)* represents an information flow from the variable *x* to the external environment, while the command *input(x)* represents an information flow from the external environment to the variable *x*. The main point of the analysis is that the execution of the instruction in a process may depend on the branch taken by the conditional instruction in another process, due the communication between the two processes. In order to take account of implicit flows when computing the security level of a value, a function associating a *dependency set* with each instruction was defined, where the dependency set is the set of branching instructions (conditional and iterative) on which the execution of the instruction depends. Note that all the instructions that follow a loop can be executed or not according to termination/non-termination of the loop. In the work, a concrete and abstract semantics of the language is given. The concrete semantics is proved adequate to check secure information flow and a theorem states that if each state of the abstract semantics of the program is different from *error* then the program satisfies the secure information flow property.

Being semantics based, abstract interpretation allows a wide class of programs to be accepted as secure (wider, for example, than typing approaches allow). At the same time, being rule based, abstract interpretation can be fully automated.

3.3.4 Type checking in embedded systems

The Java Platform includes a software module, called the bytecode Verifier, that checks the bytecode for type correctness. For example, it ensures that instructions do not forge pointers, e.g. by using integers as object references. The verification of the code is performed by executing abstractly the instructions over types instead of over actual values. In particular, the Verifier performs a data-flow analysis on the bytecode. A bytecode verification algorithm is presented in [Lindholm and Yellin 1996], and almost all existing bytecode verifiers implement that algorithm. The verification is expensive both in computation time and memory space, since the algorithm evaluates all possible execution flows and maintains the state of each instruction that is either the target of a branch or the entry point of an exception handler [Leroy 2003]. So, even though bytecode verification is a key point of the security chain of the Java Platform, verification is made optional in many embedded devices, like Java cards, since the memory requirements of the verification process are frequently too high.

In [Bernardeschi et al. 2005] a verification algorithm, called multipass verification, is proposed. This algorithm remarkably reduces the use of the memory by performing the verification during multiple

specialized passes. Each pass is dedicated to a particular type. The amount of memory required to save the states is reduced since, during each pass, the abstract interpreter needs to know only if the type (saved in a register or in a stack location) is *compatible* with the pass or not. In this case, only one bit is needed to specify the type of the data saved in registers and stack locations, rather than the three bytes used in the standard verification approach. The type compatibility is given by the types hierarchy and it can be computed statically. The analysis is performed on the whole bytecode for every pass. The number of passes depends on the instructions contained in the method. One pass is needed for each type used by the instructions. Additional passes are needed to check the initialization of objects. The multipass verification is feasible since bytecode instructions are typed and the number of types is limited: basic types (int, byte, ...), reference types (the ones listed in the constant pool) and the return address type.

A prototype tool has been developed. The prototype shows that multipass verification can be performed directly on small memory systems. Besides reducing the size of memory required for the analysis, the multipass approach is also efficient in time. The time needed for a complete multipass analysis is proportional to the number of types used by instructions and, after analyzing a large number of methods, we found that the number of types actually used in each method was on average only 5.8 (22%). Moreover, since multipass verification computes the states by comparing bits while standard verification usually needs to traverse the class hierarchy when computing results, each pass of the multipass verification is much simpler than the standard one. Thus, despite the multiple passes, the time needed for a complete multipass verification is comparable to the time needed for a standard verification.

Conclusions

Symbolic execution and abstract interpretation are verification techniques that allow systems to be modelled and analysed at different levels of abstractions. ReSIST partners (specifically Pisa University and QinetiQ) have applied them in the design of automatic tools for static program analysis.

The complexity of future ubiquitous computing systems makes abstraction a necessary feature for the successful application of formal verification to this kind of systems. Moreover, we expect that abstraction could also be useful to deal with the scalability issues relevant to ReSIST.

Abstract interpretation, in addition to program analysis, can be applied in many other different areas of computer science like in model checking, type inference and security to study properties of abstractions. This open the possibility of fruitful interactions between the different researchers in ReSIST.

4 – Robustness Testing

Introduction

Robustness may be defined as a specialized dependability attribute, characterizing a system reaction with respect to external faults [Avizienis et al. 2004]. Accordingly, robustness testing involves testing a system in the presence of faults or stressful environmental conditions. This definition covers a large spectrum of approaches, which can be classified according to two viewpoints.

The first viewpoint determines the input domain of interest. Indeed, the characterization of the input domain plays a major role in the definition of robustness testing. The input domain can be split into two main dimensions: the activity (*workload*) and the faults (*faultload*). The workload and faultload can be given more or less emphasis, depending on the approaches. Workload-based approaches extend usual testing efforts by submitting the system to higher load tests. Faultload-based approaches focus on the fault dimension and the behaviour of the system subjected to a given set of faults. Obviously, the two dimensions of the input domain may combine their effects on the system: mixed workload- and faultload-based approaches explicitly consider such combined effects.

The second viewpoint is the classification of approaches according to the pursued objective: testing may be performed for verification purposes, or for evaluation. The verification of robustness is most often on the lineage of classical testing approaches, where a model of the system (e.g., a behavioural model) is used as a guide for selecting test cases (e.g., transition coverage is required). The evaluation of robustness rather builds on fault injection and load testing approaches, for which the first-class citizens are models of the input domain. For example, the workload is selected according to a probabilistic model of the operational profile, and the faultload is based on a model of faults that are deemed representative of actual faults in operation. Recent effort to standardize this kind of evaluation-oriented testing has yielded the emergence of the concept of dependability benchmarking.

Note that the associations and oppositions *verification/model-of-the-system* versus *evaluation/model-of-the-input-domain* are somewhat too schematic. Some evaluation-oriented approaches take a model of the system into account, so as to address the issue of how to synchronize the workload and the faultload. Similarly, work on verification may consider a model of the input domain to improve the design of testing. Indeed, this chapter strongly supports the view that verification- and evaluation-oriented testing work may face common problems.

This is why this chapter does not adopt a pure verification-centric view, and also mentions work that is related to evaluation[2]. The first classification is used to build the structure of the chapter, presenting workload-based (Section 4.1), faultload-based (Section 4.2) and mixed (Section 4.3) robustness testing approaches. Conclusions and perspectives are provided in Section 4.4.

---

2 More pointers on the evaluation topic are to be found in the **Eval** part of the Deliverable, specifically in the section addressing **dependability benchmarking** approaches.

4.1. Workload-based approaches

Load testing can be used for both verification and evaluation purposes. In the first case, the aim is to verify that the system is still able to fulfil its functional requirements when submitted to a peak load. In the second case, the aim is to infer performance measures from the observed behaviour.

Performance benchmarks constitute a form of evaluation-oriented load tests whose target is a family of systems, typically Off-The-Shelf (OTS) products implementing the same interface standard [Gray 1993]. There are hundreds of performance benchmarks for computer systems and components (microprocessors, OSs, communication protocols, transactional systems, etc.). In addition to nominal performance, the behaviour of the target system when submitted to stressful activity is of great interest. Indeed, in many critical systems, a stable behaviour in presence of extra workload is preferred to very high performance in presence of nominal workload. Although dependability properties are also felt important, they are usually not covered by such benchmarks. For example, in the case of transactional systems, it is assumed that the ACID properties of the transactions are supported by the target system: the benchmarks do not include any procedure to confirm that the recovery mechanisms are working properly or to measure the impact of the recovery activations.

An example of work more related to dependability properties is [Avritzer and Weyuker 1995], that proposes an approach to study the impact of improper resource management at the applicative level. The target applications are typically telecommunication software systems. Load testing is intended to reveal design faults causing the loss of calls even though there are ample resources available, and to evaluate the gradual degradation of system capacity when such faulty states are encountered. Hence, both verification and evaluation issues are addressed. The activity is modelled by Markov chains, calibrated by operational profile data. Test cases are then automatically generated using deterministic graph algorithms that traverse only states and transitions whose steady-state probability is higher than a given threshold. This approach was applied to real systems and proved effective at revealing subtle problems.

In [Waeselynck and Thévenod-Fosse 1999], LAAS authors apply conformance testing methods with extra consideration for reusability in stressful environments. The aim is to check whether reusable concurrent objects are robust enough to satisfy their interface specification under various solicitation rates. The test inputs are generated by considering both the value and time domains. As regards the value domain, the generation is based on coverage criteria for the specified object interface models. The time intervals between successive input events are then generated according to three load profiles: (i) long delays compared to the reaction time of the objects under test; (ii) short delays; and (iii) a mix of short, long and intermediate delays. Experimental results obtained on a case study show that these load profiles are complementary in terms of their revealing power, as they induce different failure patterns.

4.2. Faultload-based approaches

These approaches concern controlled experiments that explicitly consider the various types of faults that may affect the system in operation. They span physical, design or human interaction faults. Usually the process of selecting the workload to conduct the experiments is not the main focus.

In such approaches, it is seldom the case that actual faults are applied. Most of the time, the faults are emulated: the underlying objective is to induce errors and failures that are similar to those provoked by real faults. The focus is then on the representativeness of the emulated faultload.

Testing with respect to physical faults (Section 4.2.1) is in the tradition of the fault-injection community and mainly addresses evaluation issues. Testing with respect to invalid inputs (Section 4.2.2) can be seen as a form of fault-based, verification-oriented approach coming from the software testing community. In Section 4.2.3, we show that both categories of approaches are used in recent work characterizing the failure modes of Off-The-Shelf (OTS) components.

4.2.1 Testing with respect to physical faults

Let us consider the case of external radiations that may alter the internal behaviour of ICs. Heavy-ions are a typical example of such kind of radiations. They can flip bits of the memory elements that compose a digital target system (e.g., a "zero" value stored by a memory element is modified to a "one"). Until recently, such effects were mostly observed in the case of digital devices operating in space. However, recent studies have shown that effects of radiations are becoming perceptible at modest altitudes and at ground level as well. This is the consequence of the reductions in size of the elementary devices. In the study reported in [Shivakumar et al. 2002], the authors predict that the error rates of logic circuits will increase nine orders of magnitude, from 1992 to 2011.

In addition to approaches applying actual radiations, voltages or interferences [Arlat et al. 2003], the bit-flip model has become very common and well accepted in many studies. The most popular fault injection technique that supports this fault model is the so-called SoftWare Implemented Fault Injection (SWIFI, in short) technique, see e.g. [Carreira et al. 1998]. Using this technique, memory cells (encompassing bits in memory space or even in microprocessor registers) can be selectively corrupted at run time with limited temporal intrusiveness. The thread of execution is interrupted or diverted so as to flip the value of selected bits. Single or multiple bits can be flipped at once. In practice, in most reported studies, the memory words (or registers) as well as the bits that are targeted are selected on a (pseudo)random basis.

This form of testing is usually performed for evaluation purposes. For example, the aim may be to get an estimate of the coverage [Bouricius et al. 1971] provided by the fault tolerance mechanisms included in the system, or to measure the error latency within the system as exemplified by a LAAS study [Arlat et al. 1993].

4.2.2 Testing with respect to invalid inputs

Invalid inputs may originate from human-interaction faults (e.g., an operator enters a wrong value), or from design faults residing in systems calling the services of the system under test. Testing is used to verify that the system adequately handles such invalid inputs (e.g., an appropriate error message is returned).

The decomposition of the input domain into valid and invalid classes, and the coverage of the invalid classes during testing, is common practice (see e.g., one of the earliest reference books on software testing: [Myers 79]). For example, given an input parameter $x \in [0..10]$, two invalid classes to be tested could be $x<0$ and $x>10$. In the field of protocol testing, it is also common to consider incorrect message frames, as well as inopportune events that the system should ignore or explicitly refuse at some point of the execution sequence.

The definition of invalid inputs may be more or less sophisticated, depending on the underlying input model. We present below some examples of sophisticated approaches, based on the notion of input syntax, or on the notion of type constructors.

Command-driven software, GUI, or web applications are typical examples where the inputs can be defined using a formal syntax. In such cases, syntax testing [Beizer 90] can be used for both conformance and robustness testing. This technique has been applied by [Ghosh et al. 1998] to test Windows NT software. The inputs were randomly generated from a grammar specification, allowing the production of syntactically correct usages of system commands combined with anomalous data. [Offutt et al. 2004] consider invalid input patterns for web applications. They notice that a common technique in web forms is to perform input validation on the client side (e.g., using JavaScript), but that users can bypass this validation. The defined invalid patterns include value-level patterns (e.g., the data value has not the right type), parameter level patterns (e.g., to account for inter-value constraints of pairs of parameters), and control-flow ones (e.g., to account for the user breaking the normal execution sequence by backward and forward navigation). These patterns are used to produce requests to the server.

In object-oriented software, the generation of inputs may have to account for the complex underlying type system. For example, if a class method needs an input parameter of type T, testing this method requires knowing how to construct instances of T. This problem has been addressed by [Csallner and Smaragdakis 2004] for testing the robustness of Java classes. The proposed approach starts by automatically analyzing the source code to compute a graph structure determining the various ways to construct an object of a given type. For example, a parameter of type T may be created by using the null value, by calling a constructor of T or any subtype of T, or by using any intermediate object having a T-returning method, etc. Primitive types (e.g., int) can be created with pre-set values (e.g., 0). Then, input generation proceeds by randomly traversing the graph to create different parameter combinations for the target method to be tested.

Note that the improper treatment of invalid inputs has been shown to be a major cause of failure for executive supports [Koopman and DeVale 1999]. Moreover, it is well-known that such flaws may constitute security vulnerabilities for the affected systems.

4.2.3 Testing the robustness of OTS components

Recently, many studies have been devoted to the characterization of the robustness of software OTS components. These studies were mainly motivated by the increasing tendency of using such components into computer systems, including systems devoted to control critical applications or infrastructures (e.g., transportation, telecommunications, etc.). Software OTS components may exhibit high functionality and quality as exemplified by their use in a wide range of operational contexts. But they are not necessarily developed according to the requirements imposed for critical applications. Accordingly, the objective of characterization of the behaviour of these components in the presence of faults is of high interest since upper software layers have to deal with the resulting failure modes. One difficulty is that OTS components are generally provided as “black-box” components, i.e., with limited visibility on their structure and no access to their source code. Experimental assessment provides a pragmatic way to obtain objective insights on the robustness and failure modes of a target software component.

Let us take the example of executive supports (microkernels, operating systems, middleware). The threats may come from several origins: from the software environment (applicative layer, drivers added to the executive) or from the hardware layer. Hence, a mix of test approaches can be used, building upon both the software testing technology and the fault injection one.

Flipping bits in the code or data segment of the executive is a popular technique for assessing robustness with respect to physical faults [Arlat et al. 2003]. Exercising the executive with invalid input parameters is common practice for emulating threats coming from the software interface (either the API or the driver

interface). We have already mentioned work using a formal grammar to generate commands for Windows NT software [Ghosh et al. 1998]. The most popular approach is to use a list of substitution values for each parameter type of functions at the interface. For example, a string parameter may be substituted by a null string, a long string, etc. The identification of parameter types is not as heavy as it might seem at a first glance. For example, [Koopman & DeVale 1999] identified only 20 data types for testing 233 POSIX functions. The principle of selective substitution is now commonly used for testing the robustness of software executives [Shelton et al. 2000] [Pan et al. 2001] [Albinet et al. 2004] [Kanoun et al. 2005]. This fault model can be further enriched to account for state-based functions. Then, a sequence of calls may be considered with multiple parameter corruption and modification of the order of the calls, as proposed by BUTE researchers for high availability middleware systems in **[Micskei et al 2006]**. Note that, by analogy with the fault models used by the fault-injection community, some authors have also proposed to flip bits of the parameters. Such an approach does not account for the semantics of parameters. Its representativeness with respect to software faults is often questioned in spite of the results obtained in [Madeira et al. 2000], which provided some partial support to this regards. At LAAS-CNRS, experimental results comparing the bit-flip and selective substitution models [Jarboui et al. 2003] were in favour of selective substitution.

A less used approach, also coming from the software testing technology, is to use mutations. Mutations [DeMillo et al. 1978] [Offutt and Untch 2000] [Chevalley and Thévenod-Fosse 2003] consist of elementary modifications of the software code, creating different versions of a program. Each mutant version differs from the original one by a single injected modification. Standard mutation environments work with the source code, but [Durães and Madeira 2002] also proposed an approach to emulate source-level mutations at machine-code level. They used this approach to study the robustness of operating systems with respect to faulty drivers [Durães and Madeira 2003].

All these test approaches are mostly applied for evaluation purposes: the aim is to determine the distribution of failure modes, to study the propagation of errors, or to measure the impact of the injected faults on the temporal behaviour. Due to the fact that the target OTS component is often seen as a “black box”, or a “grey box” at best, the test outcomes are observed at a coarse grain. From a verification perspective, this would make it difficult to detect erroneous behaviour patterns more sophisticated than simple crash/hangs. Note however, the work reported in [Rodriguez et al. 2003] that proposed a temporal logic framework for specifying and automatically implementing both fault injection and fine-grain observation mechanisms. This framework was developed at LAAS to test real-time microkernels. Although the aim was mostly related to evaluation, one verification-related result was that physical faults in the memory of the kernel may considerably impair the timing predictability of applications running on top of it (some predicted worst case values were exceeded) [Rodriguez et al. 2002].

More information on the robustness testing of OTS components is to be found in the **Eval** part of the Deliverable, section on **dependability benchmarking**.

4.3 Mixed Workload- and Faultload-based Approaches

In practice, the insights gained from the tests are not optimal when the design of experiments is focused only on one dimension of the input domain (either the faultload or the workload). For faultload-based approaches, it is quite common to report a large proportion (about 30% or even more in some cases) of fault injection experiments that do not result in any perceived effect. Accordingly, from a testing point of view, those experiments can be considered as “non significant tests”. Combining explicitly the faultload and workload should clearly improve this situation. The aim may be to increase controllability (the tests should be able to

sensitize the injected faults), and/or to reduce the length of the test campaign without reducing the insights that can be gained from it.

For these objectives to be achieved, it is usually necessary that some knowledge on the target system be available. Hence, faultload-and-workload-based approaches are rarely pure input-domain-modelling approaches: they also make use of a model of the system. In case no prior model of the system is available, it may be necessary to acquire knowledge through on-line observation of the system during testing.

We present below examples of mixed approaches addressing:

- the verification of resilience requirements (§4.3.1), building on model-based test selection approaches,
- the reduction of the number of "non significant tests", by selecting the injected faults based on the effect the workload has on the system (§4.3.2),
- the identification of faults classes that should be equivalent for a given system and given workload, so as to design a minimal set of experiments (§4.3.3),
- the search for test scenarios that are stressing with respect to critical requirements (§4.3.4),
- system security testing (§4.3.5)

As will be seen, effective combination of the workload and the faultload remains a difficult problem, for which the proposed solutions are limited in scope.

4.3.1 Model-based test approaches for the verification of resilience

Resilience mechanisms are specified to ensure some desired properties in the presence of a target set of threats. Verifying that the mechanism fulfils its specification can be seen as a conformance testing problem. Compared to traditional conformance testing, the only difference is the explicit fault dimension in the input domain, since faults are key inputs that the resilience mechanism is expected to deal with. Note that the fault dimension has a strong impact on the implementation of testing: the experiments may necessitate the development of complex test platforms to be able to inject the faults, synchronize them with the activity, and observe their effect.

Apart from implementation issues, the test selection methods used in that context are quite similar to methods used for verifying traditional software. Specifically, there are many examples of work using behavioural models to guide the design of testing. Four example contributions originating from LAAS-CNRS are mentioned below:

- In [Avresky et al. 1996], the authors propose a formalism to represent a fault-tolerance (FT) mechanism by a set of assertions. This formalism enables an execution tree to be generated. Test sequences are then selected to cover proper sets of paths from the root to a leaf node of the tree. Applied to an FT protocol that had been partly validated using formal methods, the application of the approach allowed two deficiencies to be revealed.

- In [Arlat et al. 1999], two types of models are considered: i) data flow graphs are used to describe the dependencies between the FT mechanisms, yielding a determination of a test order; ii) continuous time Markov chains are used to describe the dynamic behaviour of the individual FT mechanisms. Test data are randomly selected according to input profiles ensuring a balanced coverage of the transitions of these behavioural models.

- [Arazo and Crouzet 2001] provide a global framework that features a hierarchical set of viewpoints to model COTS-based FT architectures. In this framework, the behaviour of the FT mechanisms is described by means of communicating automata and formulas in temporal logic. The approach to test generation consists in mutating the automata and the formulas, and in exploiting the ability of model checkers to construct counterexamples.

- [Lussier and Waeselynck 2004] consider both behavioural models and partial proofs of properties (using theorem proving) associated with these models. The principle is to guide testing toward the pending parts of the proofs, so as to reveal flaws that would prevent the required resilience properties from being fulfilled.

As regards other work, it is worth mentioning the early contribution of [Echtle and Leu 1995] which specifies FT protocols by a Petri net, from which a reachability graph is constructed. A semi-automated procedure is defined to generate test data so as to ensure coverage of paths of this graph. More recently, the work reported in [Fernandez et al. 2005] builds upon formal methods for the conformance testing of protocols. It starts by applying syntactic transformations (mutations) to a behavioural model, according to an abstract fault model. Then, it is shown how existing approaches based on IOLTS (Input/Output Labelled Transition Systems) can be applied for producing robustness tests. A discrepancy with respect to standard approaches is that the aim is not to verify conformance with respect to the (mutated) specification. Rather, the aim is to check whether the implementation may violate a robustness requirement expressed as a Rabin automaton. The produced tests correspond to violation scenarios at the specification level.

As can be seen, robustness testing has benefited from advanced model-based approaches developed by the software testing community. It is expected that the connection will become even tighter as mature testing tools are now available. Indeed, the model-based testing community has produced numerous tools to support test generation [Hartman 2002], which reached the efficiency to be applied to industrial-scale models also, like in the experiment reported by BUTE researchers in **[Micskei and Majzik 2006]**. Such tools may offer support to robustness testing, as exemplified by the recent work mentioned above that builds upon an existing validation environment [Fernandez et al. 2005].

It is worth noting, however, that model-based approaches have mainly been used to test individual mechanisms and protocols. Testing the robustness of complex systems (such as executive supports) remains out of their scope. Also, these approaches can only handle faults that are at the same level of abstraction as the behavioural model under consideration. For example, low-level details like dynamic usage of memory, registers, etc. are usually not represented. Still, such details are relevant pieces of reality for studying the impact of bit-level physical faults.

4.3.2 Stress-based and path-based approaches for selecting faults

As already said, a major problem with faultload-based evaluation approaches is the large number of non significant tests (having no perceived effect). Effective combination of the workload and the faultload is difficult without some knowledge on the system behaviour. When robustness testing cannot be based on a behavioural model, a pragmatic approach may be to acquire knowledge on system behaviour by means of on-line observation.

The work reported in [Tsai et al. 1999] provides two complementary approaches based on this principle. In the so-called “stress-based” technique, fault injection is controlled by measuring the activity of the target system submitted to a target workload: basically, three main components (processor, memory and I/O) are

distinguished, and faults are injected on components whose activities are maximum, or that are above a given threshold. Whereas the results show that this simple technique allows fault activation to be significantly increased, it does not guarantee that all faults will be actually activated. The second technique ("path-based" fault injection) provides such a guarantee, but requires a more elaborated procedure. Path-based fault injection relies on a pre-injection analysis in order to monitor the execution paths activated by the workload, and the resources used along the paths. The information is then used to select faults that are guaranteed to be activated by this workload. The fault accounted for are essentially control-flow faults (i.e., that directly affects the execution of branches and jumps). As mentioned by the authors, there is an obvious link with white-box approaches used for software testing.

Clearly the stress-based technique is not optimal compared to path-based injection. But it is much easier to conduct. It can take advantage of monitoring facilities already present in most operating systems, and should be more amenable to tackle complex systems.

4.3.3 Fault collapsing

In some cases, it may be possible to simply skip some fault injection experiments, because their result is trivially known in advance, or because distinct faults yield the same consequences. This process is also known as "fault collapsing" and is widely used for hardware testing. It exploits information on the system design to identify dominances or equivalences in the set of faults to be injected [Pomeranz and Reddy 2004].

The work reported in [Berrojo et al. 2002] provides an example of the use of workload-dependent fault collapsing. It addresses the simulation of VLSI circuits, described as executable VHDL models. The workload-dependent analysis is based on a preliminary run of the model with no fault injected. The recorded trace allows the identification of useless faults (e.g., it is useless to corrupt a bit before a write operation) and of equivalent ones (e.g., in the time window between the last access to a bit and a subsequent read operation, all activation dates of bit corruption are equivalent). Experimental evidence gained on an industrial case study suggests that workload dependent fault collapsing may enable pruning up to 80% of the total number of faults. The remaining 20% were sufficient to exhaustively analyze the circuit behaviour with respect to the whole set of faults.

4.3.4 Search-based testing applied to robustness issues

Metaheuristic search techniques, like genetic algorithms or simulated annealing, have proven useful to solve complex optimization problems. Their generality makes them capable of very wide application. Specifically, they have gained growing attention in the field of software testing to automate the generation of test data [McMinn 2004]. The underlying testing problems are as diverse as structural testing, mutation testing, pre- and post-condition testing, determination of worst case execution time, …, and robustness testing. Whatever the testing problem, it is first reformulated as an optimization problem with respect to some objective function called *fitness function* (usually for maximization problems) or *cost function* (for minimization problems). Then, a search run involves a series of test executions. Each execution allows a candidate solution (i.e., a generated test case, or test sequence) to be evaluated with respect to the objective function. Evaluation results are used to guide the generation of new candidate solutions in the run.

Search-based testing has been used to test the robustness of control systems in [Schultz et al. 1995] [Abdellatif-Kaddour et al., 2003b]. The authors consider control systems tested in connection with a simulator of their physical environment accounting for the physical devices, the physical laws governing the controlled process, as well the possible occurrence of faults (e.g. faults affecting sensors and actuators). The

aim is to search for scenarios that would violate high-level requirements related to the most critical failure modes of the system. The search is guided by objective functions allowing the on-line evaluation of the "dangerousness" of the generated scenarios.

In [Schultz et al. 1995], the search employs genetic algorithms. A test scenario is composed of two main parts, initial conditions and fault rules. The fault rules are guarded by triggers that condition the instantiation of the fault in the simulation. For part of the experiments, the rules were integrated into a learning system, allowing the use of sophisticated learning operators to speed up the search.

In [Abdellatif-Kaddour et al. 2003a], LAAS researchers propose a stepwise construction of scenarios: each step explores possible continuations of the dangerous scenarios found at the previous step. Exploration at each step is first performed using random sampling, and the strategy is enhanced by applying simulated annealing search in [Abdellatif-Kaddour et al. 2003b].

Genetic algorithms and simulated annealing are sophisticated search techniques. But it must be stressed that their efficiency does not come for free. Some effort is always required to tune the metaheuristics to the problem to be solved (e.g., for genetic algorithms, choose an encoding of individuals, a fitness function for them, mutation and cross-over operators, the size of the population, the strategy to produce the next population from the current one). Investigating alternative design choices, and calibrating the corresponding parameters, can indeed be quite costly. The problem is discussed at length for simulated annealing search in a LAAS publication **[Waeselynck et al. 2006]**.

4.3.5 Security penetration testing

Penetration testing is a form of robustness testing with respect to malicious environments. It consists of posturing a penetrator to discover ways to breach the system's security controls. The so-called Flaw Hypothesis Methodology is a widely used penetration approach that consists of four stages [Weissman 1995]:

- Flaw Generation developing an inventory of suspected flaws;
- Flaw Confirmation assessing the flaw hypotheses as true, false or untested;
- Flaw Generalisation analysing the generality of the revealed security weakness, so that a confirmed flaw becomes a new flaw hypothesis generator;
- Flaw Elimination.

Testing may be used in the second stage. It is usually focused on the flaw hypotheses that cannot be confirmed by paper and pencil analysis. Past experience with similar systems is a good starting point for generating flaw hypotheses. There exist several tools, called security scanners, that automate testing with respect to a list of typical vulnerabilities. However, they are likely to be insufficient for a thorough validation of security. This is why the penetration approach potentially exploits any source of information available for the target system: security policy model, system specification, architecture and design, source code, operational practices… Further knowledge may be acquired by on-line observation of the system, with the aid of tools (e.g., port scan tools, to determine ports accepting connection). As an example of complete study combining disparate sources of information, one can mention [Daniels et al. 1999] that was addressing the assessment of an office machine. Note that software design faults constitute vulnerabilities than can be actively searched by the penetration team: many successful attacks are based on the exploitation of bugs. Actually, even physical faults may have an impact on security. In [Xu et al. 2001], the authors revealed that simple bit-flips could significantly increase the vulnerability of the protections supported by an operating

system. In [Chen et al. 2004], it was further shown that bit-flips in the code segment of a firewall could result in illegal packets to penetrate.

For such holistic robustness problems, the design of testing has to rely on *ad hoc* guidelines. As a result, the achieved efficiency heavily depends on the expertise of the penetration team.

Conclusions and Perspective

This chapter has presented a survey of work on robustness testing. It has been shown that the contributions came from two communities, the fault injection community (mostly addressing evaluation issues), and the software testing community (mostly addressing verification issues), with opportunities for cross-fertilization.

Robustness testing had to cope with major changes in the target computing systems, like the move from simple, dedicated systems to more complex, OTS-based systems. This yielded to enlarge the set of faults of interest, and to consider fault models at different levels of abstraction (e.g., memory faults at a concrete level, and invalid software inputs at an abstract one). In order to improve the combined selection of the faultload and workload, the design of test experiments had to account for information on the system behaviour, either under the form of a model at the appropriate level of abstraction (e.g., a VHDL model, or a transition system model), or under the form of collected on-line observation data (as in stress-based testing, or in search-based testing).

The next challenge to be faced is with the emergence of the ubiquitous computing paradigm, specifically in mobile settings. Two of the main characteristics of mobile systems are mentioned below:

- High error rates. The movement of mobile devices causes disconnections. As a result, network topology changes frequently. Moreover, limited capacity of mobile device, scarce bandwidth in wireless connection, etc. make the computation more difficult. The population of users may range from users with low level of expertise to expert but malicious ones. This all means that errors are likely to occur during the operation of mobile systems.
- Context dependency. The context includes any detectable and relevant attribute of a device, of its interaction with other devices and of its surrounding environment at an instant of time. The context is continuously evolving, so that mobile applications have to be aware of, and adapt to, the induced changes.

The development of appropriate testing technologies has been seldom explored so far.

As mobile-based applications have to cope with unstructured, evolving and uncertain environments, it is expected that input domain models will remain first-class citizens for robustness testing. A difficult issue is the richness of the context to consider, which may impact the modelling of both the faultload and the workload. As regards models of the target systems, it is worth noting that adequate specification approaches are still to be invented. Currently, from the verification side, it seems that the typical practice is to select test cases manually from informal requirements. Finally, the technological problems posed by mobile testing should not be underestimated: performing controlled experiments may require complex test platforms to possibly account for a variety of equipments (mobile devices, fixed infrastructure), communication between them, mobility models, 3D-models of geographical areas, etc.

Once again, there will be opportunities for cross-fertilization, as modelling issues and technological ones will have to be addressed for both verification and evaluation purposes.

Acknowledgements

The view developed in this chapter has benefited from pre-ReSIST work, in the framework of a French CNRS project on robustness testing (Action Spécifique no. 23), and then in the European IST ASSERT project (Automated Proof-based System and Software Engineering for Real-Time Applications, IP FP6 no. 4033, http://www.assert-online.net/).

5 – Verification of Systems Containing Cryptography[3]

Introduction

Security-critical systems are an important application area for formal methods such as verification. However, such systems often contain cryptographic subsystems. The natural definitions of these subsystems are probabilistic and in most cases based on computational restrictions. Hence it is not obvious how one can treat cryptographic subsystems in a sound way with typical formal methods, in particular if one does not want to encumber the entire proof of an overall system by probabilities and computational restrictions due only to its cryptographic subsystems (scalability).

We survey the progress of integrating cryptography into formal methods, in particular IBM Research's work based on the notion of reactive simulatability (RSIM), a refinement notion suitable for linking cryptography and more abstract specification. We also present the underlying system model which unifies a computational and a more abstract presentation and allows generic distributed scheduling. We show the relation of RSIM and other types of specifications, mention to what extent the techniques have actually been integrated into tools, and clarify to what extent the classical abstractions from cryptography by term-algebras (Dolev-Yao models) are sound.

5.1 Secure Channels as an Example of Cryptography within Larger Systems

Imagine you are using formal methods such as deductive theorem proving or model checking (both with mechanized tool support) to prove the correctness of a distributed system with respect to an overall specification. This system uses SSL/TLS for secure communication between its components; this is a widely used standard for cryptographically protecting messages on otherwise insecure channels [Dierks and Allen 1999]. What does the use of SSL mean for the overall proof?

Clearly, the nicest solution would be if you would not need to bother that SSL is a cryptographic subsystem, but could simply abstract from it by a secure channel as if that channel were realized by a dedicated and protected wire. Most formal description techniques for distributed systems used as a basis for theorem proving or model checking, have a notion of secure channels as a basic communication mechanism, or can easily specify one. Essentially, a unidirectional secure channel correctly delivers messages from one specific sender to one specific recipient and to no other party. Hence simply using such a secure-channel abstraction for SSL would be perfect for the overall system proof. In fact, this is what many people already do. For instance, the work reported in Section 3.3.3 (secure information flow analysis) assumes that parties have private channels; in real life those might be implemented by cryptography. Or one could imagine the communication in the OceanStore example in Section 2.4.4 being performed over such channels.

However, is this abstraction sound? SSL is realized with cryptographic primitives, such as Diffie-Hellman key exchange and symmetric encryption and authentication. These systems are not perfectly unbreakable;

---

[3] This chapter is based on and contains text parts from joint work with Michael Backes (Saarland University, Saarbrücken, Germany, backes@cs.uni-sb.de), and Michael Waidner (IBM Software Group, Somers, NY, USA, wmi@zurich.ibm.com), in particular [Backes et al. 2006a]. Both performed the main part of this work while at IBM Research, Rueschlikon.

e.g., an adversary with sufficient computing time can deterministically insert forged messages or learn the messages sent by honest participants. Hence the simple secure-channel abstraction that we discussed is certainly not sound in the standard absolute sense. Nevertheless, it seems "essentially" correct in the sense that for adversaries with reasonably bounded computational power, and if one ignores very small probabilities, the differences seem to disappear. Hence the two main questions are:

- Can one rigorously define a soundness notion in which a cryptographic realization like SSL can be a refinement of a non-cryptographic specification like simple secure channels?
- Does SSL have features that differ so much from the secure-channel abstraction that they are not abstracted away by this potential soundness notion, and if yes, can the secure-channel abstraction be slightly modified to accommodate these features?

As an answer to the first question, IBM Research Zurich introduced in [Pfitzmann and Waidner 2000] the notion of *reactive simulatability (RSIM)*, which we survey below. It is not only applicable to the example of secure channels, but very broadly for abstractions of cryptographic systems. As to the second question, SSL indeed has such features (imperfections). For instance, an adversary who observes the underlying communication lines can easily see who communicates when with whom, and even the length of the communicated messages, because typical encryption leaves this length more or less unchanged. The adversary can also suppress messages.

5.2 Generalizing the Example

Similar to this example, we have proposed the following overall approach when proving a system with cryptographic subsystems: First, find a good "natural" abstraction of the cryptographic subsystem. Secondly, investigate whether the natural abstraction has to be extended by certain imperfections such as leaking traffic patterns and message lengths. Thirdly, prove the real cryptographic system sound with respect to this modified abstraction in the RSIM sense. During the third step, one often also changes the cryptographic implementation a little because typical classical realizations concentrated on specific security properties and did not aim at realizing an overall abstraction. The third step is typically a manual cryptographic reduction proof at present, done once and for all for each interesting abstraction.[4] Proofs *using* the abstractions can then be made with formal methods as before, except that the imperfections of the concrete abstractions have to be encoded in the tools. Note from the SSL example that even with the imperfections the abstraction is deterministic and free of computational aspects, and thus within the scope of current tools.

5.3 Reactive Simulatability

Reactive simulatability (RSIM) is a notion for comparing two systems, typically called real and ideal system [Pfitzmann and Waidner 2000, Pfitzmann and Waidner 2001]; in this context they are the abstraction and the implementation. In terms of the formal-methods community one might call RSIM an implementation or refinement relation, specifically geared towards the preservation of what one might call secrecy properties compared with functional properties. In Figure 5.1, the ideal system is called TH (trusted host), and the

[4] It might turn out that the proof can be modularized using abstractions of lower-level primitives; then it may already be automated.

protocol machines of the real system are called M_1, ..., M_n.[5] The ideal or real system interacts with arbitrary so-called honest users, collectively denoted by a machine H, and an adversary A, who is often given more power than the honest users. In real systems A typically controls the network and can manipulate messages on the bitstring level. The option for A and H to communicate directly corresponds to known- and chosen-message attacks.

Reactive simulatability between the real and ideal system essentially means that for every adversary A on the real system there exists an adversary A' on the ideal system, such that arbitrary honest users H cannot distinguish whether they interact with the real system and A, or with the ideal system and A'. Indistinguishability of families of random variables, here the two families of views of the honest users, is a well-known cryptographic notion from [Yao 1982]. There exist stronger universal and blackbox versions of this definition, depending mainly on the quantifier order [Pfitzmann and Waidner 2000].

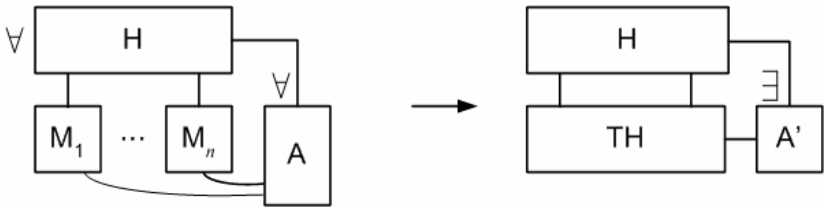

Figure 5.1 Overview of reactive simulatability (RSIM)

In the SSL example, the ideal system would be the simple secure channel with the added output of traffic data to the adversary (the imperfections), while the real machines would be actual SSL protocol machines.

The RSIM notion is based on simulatability definitions for secure (one-step) function evaluation [Goldreich et al. 1987, Goldwasser and Levin 1990, Beaver 1991, Micali and Rogaway 1991, Canetti 2000]. It is also highly related to the observational equivalence notions for a probabilistic π-calculus from [Lincoln et al. 1998]. A notion very similar to RSIM was later also called UC [Canetti 2001].

The RSIM notion fulfils important properties that one expects of refinement, in particular a composition theorem and transitivity. These properties allow the promised overall proof techniques: One can analyse the overall system using the abstraction from cryptography (and thus with standard verification tools), and obtains automatically that the overall system with the real cryptography plugged in fulfils the same properties in a well-defined sense of cryptographic soundness.

5.4 System Model

In principle, RSIM can be defined over many system models by regarding the boxes in Figure 5.1 as (possibly probabilistic) I/O automata, Turing machines, CSP or π-calculus processes, etc.. Even large parts of most theorems and proofs about RSIM are on this box-level and could be instantiated in many ways. For the rigorous definitions, we have used a system model with probabilistic I/O automata, and with a well-defined computational realization by Turing machines for computational aspects. Two important aspects are:

[5] This case that the ideal system (abstraction) is just one machine and the real system (implementation) is distributed is very common, but the model does not prescribe this.

- It is usually not sufficient that individual transitions of the automata are polynomial-time ("transaction poly"), and not even that the runtime of each automaton is polynomial in the entire inputs so far ("weakly poly"), because these notions do not compose. One usually needs overall polynomial-time restrictions in the initial inputs.
- We allow generic distributed scheduling for the asynchronous case. This means that for each part of the distributed computation that may be scheduled separately, we can designate an arbitrary other machine as the scheduler. This allows us to define adversarial scheduling with realistic information as well as, e.g., adversary-scheduled secure channels, local distributed algorithms not under control of the adversary, and specific fair schedulers. A more detailed overview of this scheduling and other nondeterminism in these models can be found in **[Backes et al. 2006b].**

Recall that while the adversary and the honest users are also polynomial-time probabilistic I/O automata in this model, they are all-quantified in the RSIM definition. This captures that the adversary is malicious and may choose the worst possible feasible behavior, and the honest users are also allowed to do whatever they want.[6]

5.5 Individual Security Properties

Reactive simulatability is great once it has been proved for a pair of an abstraction and a cryptographic realization, because then arbitrary larger systems can be proved with the abstraction and the results are also true with the realization. However, it is a strong property, and sometimes the consequences for the realization are not desired. Hence it is important to also have notions of individual security properties, such as the integrity of certain messages or the absence of information flow between certain parties. These properties can be given similar links between abstract formulations usual in formal methods and cryptographic realizations; see, e.g., [Pfitzmann and Waidner 2000, Backes and Pfitzmann 2003, Backes and Pfitzmann 2005]. Let us mention two more examples that relate to assumptions in previous sections.

An authenticity assumption as in Section 1.1.1.2 will often be realized cryptographically in practice.[7] Then it will in reality only hold under cryptographic assumptions, and with small error probabilities, and we have the same questions about soundness of the abstraction as for SSL above. This abstraction is best specified as an integrity property, and can be shown sound without additional imperfections.

Information flow in distributed systems, as discussed in Section 3.3.3, often relies on encryption. Sometimes one might not want to use just a secure-channel abstraction, but argue about the encryptions in more detail. Then a program might contain an assignment $c := \mathsf{E}(k, m)$ where m is a message to be protected ("high" in information flow terms), k a key, and E the encryption operator. The standard information-flow analysis would find that information flows from m into c. However, we used encryption precisely to be able to send the ciphertext c safely over insecure ("low") channels. Hence a desirable abstraction would be that we make E a sub-module with known information flow (such sub-modules will likely be allowed in any scalable

---

6 Sometimes concrete restrictions on the users have to be made, corresponding to a user model. Then these restrictions must be verified if the users are another protocol, or communicated as user education if the users are human. This could be related to the studies of socio-technical systems.

7 However, in the concrete TTA architecture reported in Section 1.1.1.2 it is realized in a weaker but more efficient way that does not tolerate malicious adversaries.

information-flow analysis) and define that *m* does not flow into *c*. Now, again, we have the question in which sense this is sound because an adversary with unlimited computational power might still extract *m* from *c*. This is answered by our analysis of the soundness of information-flow properties.

5.6 Dolev-Yao Models

In the past, formal methods have usually abstracted from cryptographic operations by term algebras. For instance, a ciphertext built by successively encrypting a message *m* with two keys *k* and *k'* would be represented by the term $\mathsf{E}(k, \mathsf{E}(k', m))$, and such terms would not be evaluated to bitstrings, only certain cancellation rules such as $\mathsf{D}(k, \mathsf{E}(k, m)) = m$ for symmetric encryption. This is also called symbolic cryptography or Dolev-Yao models after the first such publication [Dolev and Yao 1983]. For instance, the proofs of the MAFTIA contract signing protocol in Section 2.4.1.1 and of the Bluetooth pairing procedure in Section 2.4.2 use these abstractions. Justifying these abstractions is non-trivial because the term algebras are used as what is called initial models in the theory of term algebras. For the modelling of cryptography, this means in particular that one assumes that terms that an adversary cannot derive by the explicit cancellation rules from terms that it saw are perfectly secret from the adversary. This assumption is at first glance not at all close to actual cryptographic security definitions.

To bridge these gaps, IBM Research has shown that a Dolev-Yao model of several important core cryptographic primitives, with small extensions (imperfections) similar to those we mentioned for SSL above, can indeed be implemented soundly in the sense of blackbox RSIM (BRSIM) under standard cryptographic assumptions, see in particular [Backes et al. 2003]. Extensions of this result to more cryptographic primitives were presented in [Backes et al. 2003a, Backes and Pfitzmann 2004] and uses in protocol proofs (abstract but manual) in [Backes and Pfitzmann 2004a, Backes 2004, Backes and Dürmuth 2005, Backes and Pfitzmann 2006]. This extension was recently encoded into the theorem prover Isabelle and a first protocol proved in a mechanized way [Sprenger et al. 2006]. There is also an encoding into a type system for secrecy properties [Laud 2005].

However, we have also shown recently that extending these results to other primitives like hashing or XOR is not possible, at least not in the same very strong sense (BRSIM) and with similar generality as our positive results [Backes and Pfitzmann 2005a, Backes et al. 2006d]. An overview is given in **[Backes et al. 2006c]**. There are weaker notions of soundness for Dolev-Yao models by other authors that sometimes allow simpler abstractions and implementations than BRSIM soundness; the trade-off remains to be fully explored in particular for these additional primitives, both in the sense of what is possible in which soundness notion, and what larger proofs are possible with the other soundness notions, as they do not have such clear composition theorems as BRSIM.

In the context of larger systems, we see Dolev-Yao models as a tool on a middle level, useful for proving protocols that use standard cryptography in a blackbox way, but still rather explicitly. Within proofs of large overall systems, we believe that even more abstract specifications of cryptographic subsystems, such as entire secure channels or entire secure payment systems, are more suitable. Those may, however, be formally proved using the middle-level Dolev-Yao abstractions.

Conclusion

We have given an overview of a soundness notion that allows one to make simple, deterministic abstractions from cryptographic systems. This is vital for the scalability of sound proofs of systems involving cryptography, because prior to such abstraction options, the only sound way would have been to prove the

entire system by reduction proofs to the security definition of the cryptographic primitives. Now, in contrast, one can proceed with the abstractions and apply all the techniques described in the previous chapters.

A particular area where we can see formal methods for cryptography gaining more industrial relevance is web services security, at least if the current trend continues to make web services security specifications, like all web services specifications, highly extensible and configurable, so that one may not be able to prove all standards-compatible realizations in advance. A first analysis of how formal-methods considerations and cryptographic considerations play together in this context can be found in [Backes et al. 2006]. For purely formal considerations based on Dolev-Yao models we also refer to [Bhargavan et al. 2003, Bhargavan et al. 2004].

References

[Abdellatif-Kaddour et al. 2003a] O. Abdellatif-Kaddour, P. Thévenod-Fosse and H. Waeselynck, "Property-oriented testing: a strategy for exploring dangerous scenarios", in Proc. 18th ACM Symposium on Applied Computing (SAC'2003), Melbourne, USA, pp.1128-1134, ACM Press, 2003.

[Abdellatif-Kaddour et al. 2003b] O. Abdellatif-Kaddour, P. Thévenod-Fosse and H. Waeselynck, "An empirical investigation of simulated annealing applied to property-oriented testing", in Proc. ACS/IEEE Int. Conf. on Computer Systems and Applications (AICCSA'2003), Tunis, Tunisia, 2003.

[Adelsbach et al. 2003] A. Adelsbach and S. Creese (editors), "Final Report on Verification and Assessment", January 2003, http://www.maftia.org/deliverables/D22.pdf.

[Aguayo et al. 2001] D. Aguayo, D. De Couto, W. Lin, H. Lee, and J. Li, "Grid: Building a robust ad hoc network", In Proceedings of the 2001 Student Oxygen Workshop. MIT Laboratory for Computer Science, 2001.

[Albinet et al. 2004] A. Albinet, J. Arlat and J.-C. Fabre, "Characterization of the Impact of Faulty Drivers on the Robustness of the Linux Kernel", in Proc. IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN-2004), Florence, Italy, pp.867-876, IEEE CS Press, 2004.

[Arazo and Crouzet 2001] A. Arazo and Y. Crouzet, "Formal Guides for Experimentally Verifying Complex Software-Implemented Fault Tolerance Mechanisms", in Proc. 7th Int. Conference on Engineering of Complex Computer Systems (ICECCS'2001), Skövde, Sweden, pp.69-79, IEEE CS Press, 2001.

[Arlat et al. 1993] J. Arlat, A. Costes, Y. Crouzet, J.-C. Laprie and D. Powell, "Fault Injection and Dependability Evaluation of Fault-Tolerant Systems", IEEE Transactions on Computers, vol. 42, no. 8, pp.913-923, 1993.

[Arlat et al. 1999] J. Arlat, J. Boué and Y. Crouzet, "Validation-based Development of Dependable Systems", IEEE Micro, vol. 19, no. 4, pp.66-79, 1999.

[Arlat et al. 2003] J. Arlat, Y. Crouzet, J. Karlsson, P. Folkesson, E. Fuchs and G. H. Leber, "Comparison of Physical and Software-Implemented Fault Injection Techniques", IEEE Transactions on Computers, vol. 52, no. 8, pp.1115-1133, 2003.

[ASTREE 2006] ASTREE static analyzer web site, http://www.astree.ens.fr/.

[Avizienis et al. 2004] A. Avizienis , J-C. Laprie, B. Randell and C. Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing", IEEE Transactions on Dependable and Secure Computing, vol. 1, no. 1, pp.11-33, 2004.

[Avresky et al. 1996] D. Avresky, J. Arlat, J.-C. Laprie and Y. Crouzet, "Fault Injection for the Formal Testing of Fault Tolerance", IEEE Transactions on Reliability, vol. 45, no. 3, pp.443-455, 1996.

[Avritzer and Weyuker 1995] A. Avritzer and E.J. Weyuker, "The Automatic Generation of Load Test Suites and the Assessment of the Resulting Software", IEEE Transactions on Software Engineering, vol. 21, no. 9, pp. 705-716, 1995.

[Azeem et al. 2004] K. Azeem, P. Beechy, S. Creese, M. Goldsmith, R. Harrison, A. Hood, C. Pygott, J. Roach, W. Simmonds, P. Whittaker, "Analytical Assessment of Bluetooth Security Mechanisms", First issue January 2004. Public version, September 2004, http://www.forward-project.org.uk/PDF_Files/D4.pdf.

[Backes 2004] M. Backes, "A cryptographically sound Dolev-Yao style security proof of the Otway-Rees protocol", in Proc. 9th ESORICS, volume 3193 of LNCS, pp. 89-108. Springer, 2004.

[Backes and Dürmuth 2005] M. Backes and M. Dürmuth, "A cryptographically sound Dolev-Yao style security proof of an electronic payment system", in Proc. 18th IEEE CSFW, pp. 78-93, 2005.

[Backes and Pfitzmann 2003] M. Backes and B. Pfitzmann, "Intransitive non-interference for cryptographic purposes", in Proc. 24th IEEE Symp. on Security and Privacy, pp. 140-152, 2003.

[Backes and Pfitzmann 2004] M. Backes and B. Pfitzmann, "Symmetric encryption in a simulatable Dolev-Yao style cryptographic library", in Proc. 17th IEEE CSFW, pp. 204-218, 2004.

[Backes and Pfitzmann 2004a] M. Backes and B. Pfitzmann, "A cryptographically sound security proof of the Needham-Schroeder-Lowe public-key protocol", in Journal on Selected Areas in Communications, 22(10):2075-2086, 2004.

[Backes and Pfitzmann 2005] M. Backes and B. Pfitzmann, "Relating symbolic and cryptographic secrecy", in IEEE Transactions on Dependable and Secure Computing, 2(2):109-123, 2005, http://csdl.computer.org/dl/trans/tq/2005/02/q0109.pdf (preliminary version http://eprint.iacr.org/2004/300/).

[Backes and Pfitzmann 2005a] M. Backes and B. Pfitzmann, "Limits of the cryptographic realization of Dolev-Yao-style XOR", in Proc. 10th ESORICS, volume 3679 of LNCS, pp. 178-196. Springer, 2005.

[Backes and Pfitzmann 2006] M. Backes and B. Pfitzmann, "On the cryptographic key secrecy of the strengthened Yahalom protocol", in 21st IFIP SEC, May 2006.

[Backes et al. 2003] M. Backes, B. Pfitzmann, and M. Waidner, "A composable cryptographic library with nested operations", in Proc. 10th ACM CCS, pp. 220-230, 2003, http://doi.acm.org/10.1145/948109.948140. Long version IACR ePrint Archive, Report 2003/015, Jan. 2003, http://eprint.iacr.org/2003/015/.

[Backes et al. 2003a] M. Backes, B. Pfitzmann, and M. Waidner, "Symmetric authentication within a simulatable cryptographic library", in Proc. 8th ESORICS, volume 2808 of LNCS, pp. 271-290. Springer, 2003.

[Backes et al. 2006] M. Backes, S. Mödersheim, B. Pfitzmann, and L. Viganò, "Symbolic and cryptographic analysis of the Secure WS-ReliableMessaging Scenario", in Proc. 9th FOSSACS, volume 3921 of LNCS, pp. 428-445. Springer, 2006.

[Backes et al. 2006a] M. Backes, B. Pfitzmann, and M. Waidner, "Formal Methods and Cryptography"; FM 2006: Formal Methods, LNCS 4085, Springer-Verlag, Berlin 2006, 612-616.

[Backes et al. 2006b] M. Backes, B. Pfitzmann, and M. Waidner, "Non-Determinism in Multi-Party Computation"; Workshop on Models for Cryptographic Protocols (MCP 2006), Aarhus, July-August 2006; abstracts as report of ECRYPT (European Network of Excellence in Cryptology, IST-2002-507932), http://www.zurich.ibm.com/security/publications/2006/BaPW06c-Nondet-survey.MCP.pdf.

[Backes et al. 2006c] M. Backes, B. Pfitzmann, and M. Waidner, "Soundness Limits of Dolev-Yao Models"; Workshop on Formal and Computational Cryptography (FCC 2006), Venice, July 2006 (no formal proceedings), http://www.zurich.ibm.com/security/publications/2006/BaPW06b-Impossibility-survey.FCC.pdf.

[Backes et al. 2006d] M. Backes, B. Pfitzmann, and M. Waidner, "Limits of the Reactive Simulatability/UC of Dolev-Yao Models with Hashes", accepted for 11th ESORICS, 2006, to appear in Springer-Verlag; preliminary version Cryptology ePrint Archive, Report 2006/068, http://eprint.iacr.org/, 2006.

[Barbuti et al. 2002] R. Barbuti, C. Bernardeschi, N. De Francesco, "Abstract Interpretation of Operational Semantics for Secure Information Flow", Information Processing Letters, Vol. 83, No. 2, 2002.

[Barbuti et al. 2004] R. Barbuti, C. Bernardeschi, N. De Francesco, "Analysing Information Flow Properties in Assembly Code by Abstract Interpretation", Computer Journal, Vol. 47, No. 1, 2004.

[Beaver 1991] D. Beaver, "Secure multiparty protocols and zero knowledge proof systems tolerating a faulty minority", in Journal of Cryptology, 4(2):75-122, 1991.

[Beizer 90] B. Beizer, Software Testing Techniques, 2nd ed., New York: Van Nostrand Reinhold, 1990.

[Bernardeschi et al. 1998] C. Bernardeschi, A. Fantechi, S. Gnesi, S. Larosa, G. Mongardi, D. Romano, "A Formal Verification Environment for Railway Signalling System Design", Formal Methods in System Design, No. 12, 1998.

[Bernardeschi et al. 2000] C. Bernardeschi, A. Fantechi, L. Simoncini, "Formally Verifying Fault Tolerant System Designs", Computer Journal, Vol. 43, No. 3, 2000.

[Bernardeschi et al. 2001] C. Bernardeschi, A. Fantechi, S. Gnesi, "Formal Validation of Fault-tolerance Mechanisms", Guardsm Reliability Engineering and System Safety, No. 71, 2001.

[Bernardeschi et al. 2002] C. Bernardeschi, A. Fantechi, S. Gnesi, "Model checking fault tolerant systems", Software Testing, Verification and Reliability, No. 12, 2002.

[Bernardeschi et al. 2004] C. Bernardeschi, N. De Francesco, G.Lettieri, "Concrete and Abstract Semantics to Check Secure Information Flow in Concurrent Programs", Fundamenta Informaticae, Vol. 60, No. 1-4, 2004

[Bernardeschi et al. 2005] C. Bernardeschi, G. Lettieri, L. Martini, P. Masci, "A Space-Aware Bytecode Verifier for Java Cards", in Proc. of Bytecode2005, Edinburgh (UK), April 2005.

[Berrojo et al. 2002] L. Berrojo, F. Corno, L. Entrena, I. González, C. Lopez, M. Sonza Reorda and G. Squillero, "An Industrial Environment for High-Level Fault-Tolerant Structures Insertion and Validation", in Proc. 20th VLSI Test Symposium (VTS-2002), Monterey, USA, pp.229-236, IEEE CS Press, 2002.

[Bhargavan et al. 2003] K. Bhargavan, C. Fournet, A. Gordon, and R. Pucella, "TulaFale: A security tool for web servics", in Proc. 2nd Intern. Symp. on Formal Methods for Components and Objects (FMCO03), volume 3188 of LNCS, pp. 197-222. Springer, 2004.

[Bhargavan et al. 2004] K. Bhargavan, R. Corin, C. Fournet, and A. Gordon, "Secure sessions for web services", in ACM Workshop on Secure Web Services (SWS). ACM Press, to appear, 2004.

[Bloom 1970] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors", In Communications of the ACM, 13(7):422-426, 1970.

[Bolton et al. 2003] Christie Bolton, Sadie Creese, Michael Goldsmith, Gavin Lowe, Nick Moffat, Helen Roscoe and Paul Whittaker, "Correctness of Routing in Wireless Communications Environments", October 2003, http://www.forward-project.org.uk/PDF_Files/D3.pdf .

[Bouricius et al. 1971] W. G. Bouricius, W. C. Carter, D. C. Jessep, P. R. Schneider and A. B. Wadia, "Reliability Modeling for Fault-Tolerant Computers", IEEE Trans. on Computers, C-20 (11), pp.1306-1311, 1971.

[Binckley & Gallagher 1996] Binkley, D. & Gallagher, K., "Program Slicing", *Advances in Computing* **43**. 1996.

[Canetti 2000] R. Canetti, "Security and composition of multiparty cryptographic protocols", in Journal of Cryptology, 3(1):143-202, 2000.

[Canetti 2001] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols", in Proc. 42nd IEEE FOCS, pp. 136-145, 2001.

[Carreira et al. 1998] J. Carreira, H. Madeira and J. G. Silva, "Xception: A Technique for the Experimental Evaluation of Dependability in Modern Computers", IEEE Trans. on Software Engineering, vol. 24, no. 2, pp.125-136, 1998.

[Chen et al. 2004] S. Chen, J. Xu, Z. Kalbarczyk, R. Iyer and K. Whisnant, "Modeling and evaluating the security threats of transient errors in firewall software", Performance Evaluation, vol. 56, no. 1-4, 2004.

[Chevalley and Thévenod-Fosse 2003] P. Chevalley and P. Thévenod-Fosse, "A mutation analysis tool for Java programs", Int. Journal on Software Tools for Technology Transfer (STTT), vol. 5, no. 1, pp.90-103, 2003.

[Clarke and Jha 1995] E.M. Clarke and S. Jha, "Symmetry and induction in model checking", Computer Science Today: Recent Trends and Developments, 1000:455–470, 1995

[Clarke et al 1986] E.M. Clarke, EA Emerson, and A.P. Sistla , "Automatic verification of finite state concurrent systems using temporal logic specifications", ACM Transaction on Programming Languages and Systems, Vol. 8, No. 2, 1986.

[Clarke et al. 1993] E.M. Clarke, Th. Filkorn, and S. Jha, "Exploiting symmetry in temporal logic model checking", In CAV '93: Proceedings of the 5th International Conference on Computer Aided Verification, pages 450–462, London, UK, 1993, Springer-Verlag

[Clarke et al. 1994] E.M. Clarke, O. Grumberg, and D. E. Long, "Model checking and abstraction" ACM Transactions on Programming Languages and Systems, 16(5):1512–1542, 1994

[Csallner and Smaragdakis 2004] C. Csallner and Y. Smaragdakis, "Jcrasher: an automatic robustness tester for Java", Software – Practice and Experience, vol. 34, no. 11, pp. 1025-1050, 2004.

[Cousot 1981] P. Cousot, "Semantic Foundations of program analysis", in Program Flow: Analysis, Theory and Applications, S.S. Muchnik & N.D. Jones, eds., Prentice Hall, pp. 303-342, 1981.

[Cousot 1997] P. Cousot, "Types as Abstract Interpretations", in 24th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Programming Languages, ACM Press, pp. 316-331, 1997.

[Cousot and Cousot 1977] P. Cousot and R. Cousot, "Abstract Interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints", in Proc. 4th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pages 238–252, New York, NY, USA, 1977, ACM Press.

[Cousot and Cousot 1992] P. Cousot and R. Cousot, "Abstract interpretation frameworks", Journal of Logic and Computation, Vol. 2, No. 4, 1992.

[Cousot et al. 2005] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Mine, D. Monniaux, and X. Rival, "The ASTREE analyser", in ESOP 2005 — The European Symposium on Programming, Springer, M. Sagiv, ed., pp. 21-30, 2005.

[Creese et al. 2003] S. Creese and W. Simmonds (editors), "Specification and Verification of Selected Intrusion Tolerance Properties in CSP", June 2003, http://www.maftia.org/deliverables/D7.pdf

[Daniels et al. 1999] T. E. Daniels, B. A. Kuperman and E. H. Spafford, "Penetration analysis of a Xerox Documenter DC 230ST: Assessing the Security of a Multipurpose Office Machine", CERIAS, Tech. Report, N°99-09, 1999.

[DeMillo et al. 1978] R. A. DeMillo, R. J. Lipton, and F. G. Sayward. "Hints on test data selection: help for the practicing programmer", IEEE Computer, vol. 11, no. 4, pp. 34-41, 1978.

[De Nicola and Vaandrager 1990] R. de Nicola and FW Vaandrager, "Actions versus state based logics for transition systems", in Proc. Ecole de Printemps on Semantics of Concurrency, Lecture Notes in Computer Science 469, Springer, Berlin, 1990.

[Dierks and Allen 1999] T. Dierks and C. Allen, "The TLS Protocol Version 1.0, 1999", Internet RFC 2246.

[Dolev and Yao 1983] D. Dolev and A. Yao, "On the Security of Public Key Protocols", IEEE Transactions on Information Theory, vol. 29, no. 2, pp. 198-208, 1983.

[Dolev et al. 1986] D. Dolev, N. A. Lynch, S. S. Pinter, E. W. Stark, and W. E. Weihl. "Reaching approximate agreement in the presence of faults". Journal of the ACM, 33(3):499-516, 1986.

[Durães and Madeira 2002] J. Durães and H. Madeira, “Emulation of software faults by selective mutations at machine-code level”, in Proc. 13th IEEE Int. Symp. on Software Reliability Engineering (ISSRE 2002), Annapolis, MD, USA, pp. 329-340, 2002.

[Durães and Madeira 2003] J. Durães and H. Madeira, “Multidimensional Characterization of the Impact of Faulty Drivers on the Operating Systems Behavior”, Special Issue on Dependable Computing of the Transactions of IEICE (Institute of the Electronics, Information and Communication Engineers) Journal, Japan, December 2003.

[Echtle and Leu 1995] K. Echtle and M. Leu, “Test of Fault Tolerant Distributed Systems by Fault Injection”, in Fault-Tolerant Parallel and Distributed Systems (D. Pradhan and D. Avresky, Eds.), pp.244-251, IEEE CS Press, Los Alamitos, CA, USA, 1995.

[Fantechi et al. 1998] A. Fantechi, S. Gnesi, R. Pugliese, E. Tronci, “A symbolic model checker for ACTL”, in Proc. FM-Trends'98: International Workshop on Current Trends in Applied Formal Methods, Lecture Notes in Computer Science 1641, 1998.

[FDR 2006] Formal Systems (Europe) Ltd, “Failures-Divergences Refinement: FDR2 Manual”. 2006. Further information available at http://www.fsel.com

[Fernandez et al. 2005] J.C. Fernandez, L. Mounier and C Pachon, “A Model-based Approach for robustness testing”, in Proc. 17th IFIP Int. Conference on Testing of Communicating Systems (TESTCOM 2005), Montreal, Canada, pp. 333-348, LNCS 3502, Springer, 2005

[FORWARD] “A Future of Reliable Wireless Ad-hoc networks of Roaming Devices”, part of the UK Department of Trade & Industry (DTI) “Next Wave Technologies and Markets” programme, 2006, http://www.forward-project.org.uk/.

[Ghosh et al. 1998] A.K. Ghosh, V. Shah and M. Schmid, “An approach for Analysing the Robustness of Windows NT Software”, in Proc. 21st National Information Systems Security Conference, Crystal City, VA, USA, pp. 383-391, 1998.

[Goldreich et al. 1987] O. Goldreich, S. Micali, and A. Wigderson, “How to play any mental game - or - a completeness theorem for protocols with honest majority”, in Proc. 19th ACM STOC, pp. 218-229, 1987.

[Goldsmith and Whittaker 2005] M. Goldsmith and P. Whittaker, “A CSP Frontend for Probabilistic Tools”, June 2005, http://www.forward-project.org.uk/PDF_Files/D14.pdf

[Graf and Saidi 1997] S. Graf and H. Saıdi, “Construction of abstract state graphs with PVS”, In CAV ’97: Proceedings of the 9th International Conference on Computer Aided Verification, pages 72–83, London, UK, 1997, Springer-Verlag

[Goldwasser and Levin 1990] S. Goldwasser and L. Levin, “Fair computation of general functions in presence of immoral majority”, in Proc. CRYPTO '90, volume 537 of LNCS, pp. 77-93. Springer, 1990.

[Gray 1993] J. Gray (Ed.), The Benchmark Handbook for Database and Transaction Processing Systems, Morgan Kaufmann Publishers, San Francisco, CA, USA, 1993.

[Hartman 2002] A. Hartman, “Model-based test generation tools”, Agedis Consortium, URL: http://www.agedis.de/documents/ModelBasedTestGenerationTools_cs.pdf, 2002.

[Hinton et al. 2006] A. Hinton, M. Kwiatkowska, G. Norman and D. Parker. "PRISM: A Tool for Automatic Verification of Probabilistic Systems". In H. Hermanns and J. Palsberg (editors) Proc. 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06), volume 920 of LNCS, pages 441-444, Springer. March 2006.

[Hote 2004] C. Hote, “Run-time Error Detection through Semantic Analysis: A Breakthrough Solution to Today's Software Testing Inadequacies”, Technical report, PolySpace Technologies Inc. 2004 (original version 2001).

[Hote 2005] C. Hote, “Advanced Software Static Analysis Techniques That Provide New Opportunities for Reducing Debugging Costs and Streamlining Functional Tests”, Technical report, PolySpace Technologies Inc. 2005.

[Jarboui et al. 2003] T. Jarboui, J. Arlat, Y. Crouzet, K. Kanoun and T. Marteau, “Impact of Internal and External Software Faults on the Linux Kernel”, IEICE Transactions on Information and Systems, E86-D (12), pp.2571-2578, 2003.

[Jensen, 2002] T. Jensen, “Types in program analysis”, in The Essence of Computation: Complexity, A nalysis, Transformation (Essays Dedicated to Neil D. Jones), T. Mogensen; D. Schmidt & I. H. Sudborough, eds., Springer, pp. 204-222. 2002.

[Kanoun et al. 2005] K. Kanoun, Y. Crouzet, A. Kalakech, A. E. Rugina and P. Rumeau, “Benchmarking the Dependability of Windows and Linux using Postmark Workloads”, in Proc. 16th Int. Symp. on Software Reliability Engineering (ISSRE 2005), (Chicago, IL, USA), pp.11-20, IEEE CS Press, 2005.

[Kesten and Pnueli 2000] Y. Kesten and A. Pnueli, “Control and Data Abstraction: The Cornerstones of Practical Formal Verification”, In International Journal on Software Tools for Technology Transfer, pages 328-342, 2000

[Koopman and DeVale 1999] P. Koopman and J. DeVale, “Comparing the Robustness of POSIX Operating Systems”, in Proc. 29th Int. Symp. on Fault-Tolerant Computing (FTCS-29), Madison, WI, USA, pp.30-37, IEEE CS Press, 1999.

[Kubiatowicz et al. 2000] John Kubiatowicz, David Bindel, Yan Chen, Patrock Eaton, Dennis Geels, Ramakrishna Gummadi, sean Rhea, Hakim Weatherspoon, Westly Weimer, Christopher Wells, ben Zhao, “Oceanstore: An architecture for global-scale persistent storage”, Proceedings of ACM ASPLOS, ACM November 2000.

[Kurshan and McMillan 1995] R. P. Kurshan and K. L. McMillan, “A structural induction theorem for processes”, Information and Computation, 117(1):1–11, 1995

[Lamport et al. 1982] L. Lamport, R. Shostak, M. Pease, “The byzantine generals problem”, ACM Transactions on Programming Languages and Systems, 4(3), 1982.

[Laud 2005] P. Laud, “Secrecy types for a simulatable cryptographic library”, in Proc. 12th ACM CCS, pp. 26-35, 2005.

[Leroy 2003] X. Leroy, “Java bytecode verification: algorithms and formalizations”, Journal on automated reasoning, Vol 30, 2003.

[Leuschel et al. 2001] M. Leuschel, T. Massart and A. Currie, "How to Make FDR Spin: LTL Model Checking of CSP by Refinement", in Proc. FME'01, LNCS vol. 2021, 2001.

[Lincoln et al. 1998] P. Lincoln, J. Mitchell, M. Mitchell, and A. Scedrov, "A probabilistic poly-time framework for protocol analysis", in Proc. 5th ACM CCS, pp. 112-121, 1998.

[Lindholm and Yellin 1996], T. Lindholm and F. Yellin, "The Java Virtual Machine Specification", Addison-Wesley Publishing Company, 1996.

[Lowe 1998] G. Lowe, "Casper: A Compiler for the analysis of security protocols", in Journal of Computer Security, 6:53-84, 1998.

[Lowe 1999] G. Lowe, "Casper: A Compiler for the analysis of security protocols – User Manual and Tutorial", Department of Mathematics and Computer Science, University of Leicester, LE1 7RH, UK, 1.3 edition, July 1999.

[Lussier and Waeselynck 2004] G. Lussier and H. Waeselynck, "Deriving test sets from partial proofs", Proc. 15th IEEE Int. Symposium on Software Reliability Engineering (ISSRE'2004), Saint-Malo, France, pp. 14-24, IEEE CS Press, 2004.

[Madeira et al. 2000] H. Madeira, D. Costa and M. Vieira, "On the Emulation of Software Faults by Software Fault Injection", in Proc. Int. Conference on Dependable Systems and Networks (DSN-2000), New York, NY, USA, pp.417-426, IEEE CS Press, 2000.

[MAFTIA] "Malicious and Accidental-Fault Tolerance for Internet Applications", EU Framework Five Program, http://www.maftia.org/

[McMinn 2004] P. McMinn, "Search-Based Software Test Data Generation: A Survey", Software Testing, Verification & Reliability, vol. 14, no. 2, pp. 105-156, 2004.

[Micali and Rogaway 1991] S. Micali and P. Rogaway, "Secure computation", in Proc. CRYPTO '91, volume 576 of LNCS, pp. 392-404. Springer, 1991.

[Micskei and Majzik 2006] Z. Micskei and I. Majzik, "Model-based Automatic Test Generation for Event-Driven Embedded Systems using Model Checkers," in Proc. of Dependability of Computer Systems (DepCoS '06), Szklarska Poręba, Poland, pp.192-198, IEEE CS Press, 2006.

[Micskei et al. 2006] Z. Micskei, I. Majzik and F. Tam, "Robustness Testing Techniques For High Availability Middleware Solutions," in Proc. Int. Workshop on Engineering of Fault Tolerant Systems (EFTS 2006), Luxembourg, Luxembourg, 12 - 14 June, 2006.

[Moffat et al. 2005] N. Moffat, M. Goldsmith and S. Creese, "A Methodology for Assessing Performance", March 2005, http://www.forward-project.org.uk/PDF_Files/D21.pdf

[Myers 79] G.J. Myers, The art of software testing, John Wiley & Sons, 1979.

[Nielson 1982] F. Nielson, "A denotational framework for data flow analysis", in *Acta Informatica* **18**, 165-187, 1982.

[Nielson 1989] F. Nielson, "Two-level semantics and abstract interpretation", in Theoretical Computer Science 69(2), 117-242. 1989.

[Nielson et al. 1999] F. Nielson, H.R. Nielson, and C. Hankin, “Principles of Program Analysis”, Springer, 1999.

[Offutt and Untch 2000] J. Offutt and R. Untch, “Mutation 2000: Uniting the Orthogonal”, in Proc. Mutation 2000: Mutation Testing in the Twentieth and the Twenty First Centuries, pp. 45-55, San Jose, CA, October 2000.

[Offutt et al. 2004] J. Offutt, Y. Wu, X. Du and H. Huang, “Bypass Testing of Web Applications”, in Proc. 15th IEEE International Symposium on Software Reliability Engineering (ISSRE '04), Saint-Malo, Bretagne, France, pp. 187-197, 2004.

[Palsberg 2001] J. Palsberg, "Type-based analysis and applications", http://citeseer.ist.psu.edu/palsberg01typebased.html. 2001.

[Pan et al. 2001] J. Pan, P. J. Koopman, D. P. Siewiorek, Y. Huang, R. Gruber and M. L. Jiang, “Robustness Testing and Hardening of CORBA ORB Implementations”, in Proc. Int. Conference on Dependable Systems and Networks (DSN-2001), Göteborg, Sweden, pp.141-150, IEEE CS Press, 2001.

[Pfeifer et al. 1999] H. Pfeifer, D. Schwier, and F. von Henke, “Formal Verification for Time-Triggered Clock Synchronization”, in Proc. of Dependable Computing for Critical Applications 7, pp. 207–226. IEEE Computer Society, January 1999.

[Pfeifer 2000] H. Pfeifer, ”Formal Verification of the TTP Group Membership Algorithm”, in Proc. of FORTE XIII / PSTV XX, pp. 3–18. Kluwer Academic Publishers, October 2000.

[Pfeifer and von Henke 2006] H. Pfeifer and F. von Henke, “Modular Formal Analysis of the Central Guardian in the Time-Triggered Architecture”, Reliability Engineering & System Safety, Special Issue on Safety, Reliability and Security of Industrial Computer Systems, Elsevier Ltd., 2006, to appear.

[Pfitzmann and Waidner 2000] B. Pfitzmann and M. Waidner, “Composition and integrity preservation of secure reactive systems”, in Proc. 7th ACM CCS, pp. 245-254, 2000, http://www.acm.org/pubs/contents/proceedings/commsec/352600/ (personal version http://www.zurich.ibm.com/security/publications/2000/PfiWai2000aCompositionIntegrity-ACMCCS.pdf).

[Pfitzmann and Waidner 2001] B. Pfitzmann and M. Waidner, “A model for asynchronous reactive systems and its application to secure message transmission”, in Proc. 22nd IEEE Symp. on Security & Privacy, pp. 184-200, 2001, http://computer.org/proceedings/s%26p/1046/10460184abs.htm, personal version http://www.zurich.ibm.com/security/publications/2001/PfiWai2001AsyncCompositionOaklandl.pdf.

[Polyspace 2006] Polyspace web site, http://www.polyspace.com/ 2006.

[Pomeranz and Reddy 2004] I. Pomeranz and S. M. Reddy, “Properties of Maximally Dominating Faults”, in Proc. 13th IEEE Asian Test Symposium (ATS-2004), Kentin, Taiwan, pp.448-453, IEEE CS Press, 2004.

[H.V. Ramasamy et al. 2002] H. V. Ramasamy, M. Cukier, and W. H. Sanders. “Formal specification and verification of a group membership protocol for an intrusion-tolerant group communication system.” In PRDC '02: Proceedings of the 2002 Pacific Rim International Symposium on Dependable Computing, page 9, Washington, DC, USA, 2002. IEEE Computer Society.

[Rodriguez et al. 2002] M. Rodriguez, A. Albinet, J. Arlat, "MAFALDA-RT: a tool for dependability assessment of real-time systems", in Proc. Int. Conference on Dependable Systems & Networks (DSN'2002), Washington, USA, pp.267-272, IEEE CS Press, 2002.

[Rodriguez et al. 2003] M. Rodriguez, J.C. Fabre and J. Arlat, "Building SWIFI tools from temporal logic specifications", in Proc. Int. Conference on Dependable Systems and Networks (DSN'2003), San Francisco, USA, pp.95-104, IEEE CS Press, 2003.

[Roscoe 1998] A.W. Roscoe, "The Theory and Practice of Concurrency", Prentice Hall Series in Computer Science, 1998.

[Rushby 1999] J. Rushby, "Systematic Formal Verification for Fault-Tolerant Time-Triggered Algorithms", IEEE Transactions on Software Engineering, Vol. 25, No. 5, pp. 651–660, September 1999.

[Sabelfeld and Myers 2003] A. Sabelfeld and C. Myers, "Language-based Information flow Security", IEEE Journal on Selected Areas in communications Vol. 21, No. 1, 2003.

[Schmidt 1996] D.A. Schmidt, "Abstract interpretation of small-step semantics", in Proc. 5th LOMAPS Workshop on Analysis and Verification of Multiple-Agent Languages, 1996.

[Schultz et al. 1995] A.C. Schultz, J.J. Grefenstette and K.A. De Jong, "Learning to Break Things: Adaptive Testing of Intelligent Controllers", in Handbook on Evolutionary Computation, Chapter G3.5, IOP Publishing Ltd. and Oxford University Press, 1995.

[Serafini et al. 2006] M. Serafini, P. Bokor and N. Suri, "On Exploiting Symmetry to Verify Distributed Protocols", Fast Abstract, International Conference on Dependable Systems and Networks (DSN) 2006.

[Shelton et al. 2000] C. Shelton, P. Koopman and K. Devale, "Robustness Testing of the Microsoft Win32 API", in Proc. Int. Conference on Dependable Systems and Networks (DSN'2000), New York, NY, USA, pp.261-270, IEEE CS Press, 2000.

[Shivakumar et al. 2002] P. Shivakumar, M. Kistler, S. W. Keckler, D. Burger and L. Alvisi, "Modeling the Effect of Technology Trends on the Soft Error Rate of Combinatorial Logic", in Proc. Int. Conf. On Dependable Systems and Networks (DSN-2002), Washington, DC, USA, pp.389-398, IEEE CS Press, 2002.

[Simmonds and Hawkins 2004] W. Simmonds and T Hawkins, "A CSP Framework for Analysing Fault-Tolerant Distributed Systems", June 2004, http://www.forward-project.org.uk/PDF_Files/D9.pdf

[Simmonds and Hawkins 2005] W. Simmonds, T. Hawkins, "The Modelling and Analysis of OceanStore Elements Using the CSP Dependability Library", Trustworthy Global Computing (TGC) 2005, 230-247.

[Sintzoff 1972] M. Sintzoff, "Calculating properties of programs by valuations on specific models", *in*'ACM Conference on Proving Assertions about Programs, pp. 203-207. 1972.

[Sprenger et al. 2006] C. Sprenger, M. Backes, D. Basin, B. Pfitzmann, and M. Waidner, "Cryptographically sound theorem proving", in Proc. 19th IEEE CSFW, pp. 153-166, 2006. Preliminary version IACR ePrint Archive, Record 2006/047, February 2006, http://eprint.iacr.org/2006/047/.

[Steiner et al. 2004] W. Steiner, J. Rushby, M. Sorea, and H. Pfeifer. "Model Checking a Fault-Tolerant Startup Algorithm: From Design Exploration To Exhaustive Fault Simulation", in Proc. of the International Conference on Dependable Systems and Networks (DSN 2004), pp. 189-198, June 2004.

[Tsai et al. 1999] T. K. Tsai, M.-C. Hsueh, Z. Kalbarczyk and R. K. Iyer, "Stress-Based and Path-Based Fault Injection", IEEE Transactions on Computers, vol. 48, no. 11, pp.1183-1201, 1999.

[Veríssimo et al. 2006] P. Veríssimo, N. Neves, C. Cachin, J. Poritz, Y. Deswarte, D. Powell, R. Stroud and I. Welch, "MAFTIA Intrusion-Tolerant Middleware: The Road to Automatic Security", IEEE Security and Privacy, 4(4), July-August 2006.

[Waeselynck and Thévenod-Fosse 1999] H.Waeselynck and P. Thévenod-Fosse, "A case study in statistical testing of reusable concurrent objects", in Proc. 3rd European Dependable Computing Conference (EDDC-3), Prague, Czech Republic, LNCS 1667, Springer Verlag, pp.401-418, 1999.

[Waeselynck et al. 2006] H. Waeselynck, P. Thévenod-Fosse and O. Abdellatif-Kaddour, "Simulated annealing applied to test generation: landscape characterization and stopping criteria", to appear in *Empirical Software Engineering*, 2006.

[Weissman 1995] C. Weissman, "Security Penetration Testing Guidelines - Chapter 10", in Navy Handbook for the Computer Security Certification of Trusted Systems, Technical Memorandum 5540:082A, Naval Research Laboratory, USA, 1995.

[Whiting and Hill 1999] E. Whiting and M. Hill, "Safety Analysis of Hawk in Flight Monitor", in Proceedings of ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering. 1999.

[Xia and Hook 2000] S. Xia and J. Hook, "Dependently typing JVM method invocation", available at http://citeseer.ist.psu.edu/xia00dependently.html 2000.

[Xu et al. 2001] J. Xu, S. Chen, Z. Kalbarczyk and R. K. Iyer, "An Experimental Study of Security Vulnerabilities Caused by Errors", in Proc. IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN-2001), Göteborg, Sweden, pp.421-430, IEEE CS Press, 2001.

[Yao 1982] A. C. Yao, "Theory and applications of trapdoor functions", in Proc. 23rd IEEE FOCS, pp. 80-91, 1982.