HAL
archives-ouvertes.fr

Conceptual Models for Assessment & Assurance of Dependability, Security and Privacy in the Eternal CONNECTed World

Antonia Bertolino, Silvano Chiaradonna, Gabriele Costa, Felicita Di Giandomenico, Antinisca Di Marco, Paul Grace, Valérie Issarny, Marta Kwiatkowska, Fabio Martinelli, Paolo Masci, et al.

▶ **To cite this version:**

Antonia Bertolino, Silvano Chiaradonna, Gabriele Costa, Felicita Di Giandomenico, Antinisca Di Marco, et al.. Conceptual Models for Assessment & Assurance of Dependability, Security and Privacy in the Eternal CONNECTed World. [Technical Report] 2010. inria-00465221

HAL Id: inria-00465221
https://hal.inria.fr/inria-00465221

Submitted on 19 Mar 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

CONNECT

Emergent Connectors for

Eternal Software Intensive Networked Systems

ICT FET IP Project

Deliverable D5.1

Conceptual Models for Assessment & Assurance of Dependability, Security and Privacy in the Eternal CONNECTed World

SEVENTH FRAMEWORK PROGRAMME

http://www.connect-forever.eu

INRIA · DOCOMO Euro-Labs · LANCASTER UNIVERSITY · THALES · Universita' dell'Aquila · technische universität dortmund · UNIVERSITY OF OXFORD · UPPSALA UNIVERSITET

Project Number	:	231167
Project Title	:	CONNECT – Emergent Connectors for Eternal Software Intensive Networked Systems
Deliverable Type	:	Report

Deliverable Number	:	D5.1
Title of Deliverable	:	Conceptual Models for Assessment & Assurance of Dependability, Security and Privacy in the Eternal CONNECTed World
Nature of Deliverable	:	R
Dissemination Level	:	Public
Internal Version Number	:	0.3
Contractual Delivery Date	:	M12
Actual Delivery Date	:	12 February 2010
Contributing WPs	:	WP5
Editor(s)	:	Antonia Bertolino
Author(s)	:	Antonia Bertolino, Silvano Chiaradonna, Gabriele Costa, Felicita Di Giandomenico, Antinisca Di Marco, Paul Grace, Valerie Issarny, Marta Kwiatkowska, Fabio Martinelli, Paolo Masci, Ilaria Matteucci, Hongyang Qu, Mark Rouncefield, Rachid Saadi, Antonino Sabetta, Romina Spalazzese, Francois Taiani
Reviewer(s)	:	Valerie Issarny, Francois Taiani

Abstract

This is the first deliverable of WP5, which covers Conceptual Models for Assessment & Assurance of Dependability, Security and Privacy in the Eternal CONNECTed World. *As described in the project DOW, in this document we cover the following topics:*

- *Metrics definition*
- *Identification of limitations of current V&V approaches and exploration of extensions/refinements/new developments*
- *Identification of security, privacy and trust models*

WP5 focus is on dependability concerning the peculiar aspects of the project, i.e., the threats deriving from on-the-fly synthesis of CONNECT*ors. We explore appropriate means for assessing/guaranteeing that the* CONNECT*ed System yields acceptable levels for non-functional properties, such as reliability (e.g., the* CONNECT*or will ensure continued communication without interruption), security and privacy (e.g., the transactions do not disclose confidential data), trust (e.g., Networked Systems are put in communication only with parties they trust).*
After defining a conceptual framework for metrics definition, we present the approaches to dependability in CONNECT*, which cover:* i) *Model-based V&V,* ii) *Security enforcement and* iii) *Trust management. The approaches are centered around monitoring, to allow for on-line analysis. Monitoring is performed alongside the functionalities of the* CONNECT*ed System and is used to detect conditions that are deemed relevant by its clients (i.e., the other* CONNECT *Enablers). A unified lifecycle encompassing dependability analysis, security enforcement and trust management is outlined, spanning over discovery time, synthesis time and execution time.*

Keyword List

CONNECT*ed System,* CONNECT*or, Dependability, Enabler, Generic Metrics, Metric Refinement, Monitoring, Networked System, Resilience, Security, Security Policy, Security-by-Contract, Soft Metrics, State-Based Stochastic Methods, Stochastic Model Checking, Trust*

Document History

Version	Type of Change	Author(s)
0.1 0.2 0.3	release for Oxford meeting complete draft for internal review final release	A. Bertolino (ed.) A. Bertolino (ed.) A. Bertolino (ed.)

Document Review

Date	Version	Reviewer	Comment
29 Jan. 2010 3 Feb. 2010	0.2 0.2	F. Taiani V. Issarny	various comments throughout various comments throughout

Table of Contents

List of Figures

List of Tables

1 Introduction

The CONNECT world envisions dynamic environments populated by technological islands which are referred to as Networked Systems. CONNECT aims at overcoming the heterogeneity barriers that prevent Networked Systems from being eternally CONNECTed and at enabling their seamless composition in spite of technology heterogeneity and evolution. The ambitious goal of the project is to have eternally functioning systems within a dynamic evolving context. This is achieved by synthesizing *on-the-fly* the CONNECTors through which Networked Systems communicate. The resulting emergent CONNECTors then compose and further adapt the interaction protocols run by the CONNECTed Systems. This abstract CONNECT architecture is depicted schematically in Figure 1.1.

Indeed, as stated in Deliverable D1.1, four actors with different roles and different characteristics can be pointed out in the CONNECT architecture: (i) Networked Systems, that use CONNECT services; (ii) CONNECT Enablers, that encapsulate the CONNECT logic to synthesise a communication bridge between heterogeneous Networked Systems; (iii) CONNECTors, i.e., the emergent bridges synthesised by CONNECT Enablers; (iv) the CONNECTed System obtained by CONNECTing different Networked Systems.

Clearly, true communication between two heterogeneous Networked Systems speaking different languages can only be achieved by sharing a communication protocol and establishing a common semantics behind the interactions between the two parties. These two ingredients (communication protocol and semantics) will form the two basic building blocks of the synthesized CONNECTor, for ensuring functional compliance in the communication between Networked Systems. They are however necessary prerequisite, but not sufficient *per se* to guarantee that the CONNECTed Networked Systems will properly cooperate.

To achieve effective communication, in fact, it is also necessary to provide guarantees of non-functional properties, such as reliability (e.g., the CONNECTor will ensure continued communication without interruption), security and privacy (e.g., the transactions do not disclose confidential data), trust (e.g., Networked Systems are put in communication only with parties they trust). Indeed, there exist many potential threats to the dependability of modern dynamic, evolving and heterogeneous systems [63].

Systems that are pervasive, adaptive, and continuously undergo changes of course are not the exclusive realm of CONNECT. A large community exists that actively addresses methodologies to assess and ensure the dependability of nowadays complex and ever-changing systems. When the emphasis is on the evolution of the considered system, the term *resilience* [96, 117] is often used in place of the more classical term “dependability”. Laprie [117] defines resilience as “the persistence of dependability when facing changes”. In this deliverable, we will use indifferently the two terms.

Several other European projects specifically focus on achieving dependability against accidental and intentional failures, both in traditional settings [4, 3], and, more recently, in evolving systems. Notably, the European Network of Excellence ReSIST [170] provides, readily available on-line, a wealthy of up-to-date publications in an organized repository. Interesting and comprehensive is also [96], which reports the results from a focused workshop on resilience engineering.

In CONNECT, we build on the existing literature and other related projects for general dependability needs descending from the dynamic nature of systems, as discussed in [64], and focus the investigation on dependability towards those aspects more specific to CONNECT. Specifically, we address the threats deriving from on-the-fly synthesis of CONNECTors: is the CONNECTor reliable?, is the CONNECTed System secure?. While of course CONNECT is not immune to other sources and kinds of failures, we are more interested here to understand what is the potential impact on system dependability, security and trust of the communication established through the CONNECT approach.

Work Package 5 (WP5) within CONNECT covers all such important concerns and therefore it obviously plays a central role in the project. In WP5, we are investigating a comprehensive approach, which combines dependability analysis, security enforcement and trust assessment, and is centred around a lightweight adaptive monitoring framework. The activity on dependability assessment is strictly related and complemented by a verification framework, including on-line verification and quantitative compositional reasoning, which is part of the foundations and verification methods for composable CONNECTors under investigation in Work Package 2 [53].

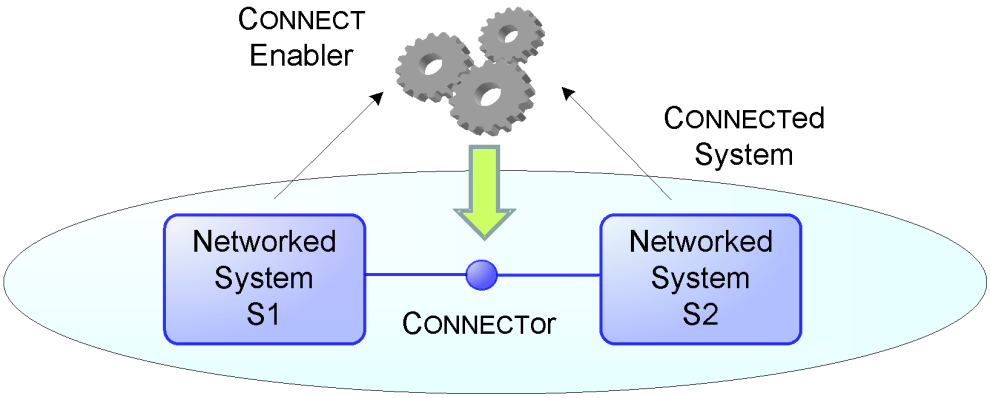


Figure 1.1: Actors in the CONNECT architecture

1.1 WP5 Tasks

WP5 activity is structured into four tasks:
Task 5.1. Dependability metrics for open dynamic systems: the aim is to revisit classical dependability metrics to account for dynamic CONNECTions in open, evolutionary networks and elicit relevant properties to be ensured.
Task 5.2. Dependability verification & validation (V&V) in evolving, adaptive contexts: this will develop approaches for quantitative verification of dependability properties, and for lightweight adaptive monitoring that is meant to detect potential problems and provide feedback to learning and synthesis activities.
Task 5.3. Security and privacy: this will adapt and extend existing techniques for security-by-contract checking and enforcement.
Task 5.4. Distributed trust management: this will develop a unifying theory for trust and a corresponding reputation scheme.

At this stage of the project, the investigation within the four tasks has taken a safe comprehensive scope, covering a wide range of approaches for off-line and on-line V&V, for establishment and enforcement of security contracts, and for measuring the trustworthiness of Networked Systems. As the project proceeds and the other WPs produce more advanced results on the CONNECT architecture and the learning&synthesis process, we will be able to get a more accurate view of needed dependability properties, so to refine the adopted dependability assurance approaches and better focus our efforts on the most relevant techniques and tools.

1.2 This Deliverable

This is the first deliverable of WP5: *Conceptual Models for Assessment & Assurance of Dependability, Security and Privacy in the Eternal* CONNECT*ed World*.

The activity of WP5 has started in Month 4. As a first step, in the elapsed months we have started with surveying and adapting models and technologies for Dependability, Security and Trust, coming from the partners background, and with eliciting at Consortium level which are the relevant dependability metrics to be assured. These metrics are addressed within Task 5.1 and are the subject of Chapters 2 and 3. We present an initial conceptual framework, which is still undergoing refinement.

Task 5.2 addresses off-line and on-line V&V and monitoring. The approaches which we intend to develop and adapt are introduced in Chapters 7 (concerning monitoring) and 3: in the latter currently we describe state-of-art model-based analysis and probabilistic model checking techniques and tools to be applied for off-line analysis. Future activity will be concerned with: on the one side, developing an effective monitor and linking it with CONNECT Enablers; on the other side, investigating the feasibility of moving some of the off-line dependability analyses to on-line.

Task 5.3 addresses security concerns by adopting and extending state-of-art approaches for contract-based security enforcement. For the moment we present in Chapter 5 a model based on some assumptions in the Networked System, future work will investigate how to release some of these assumptions, and how to integrate the model with trust management.

Task 5.4 addresses trust management. In Chapter 6 a CONNECT trust model is defined through which Enablers can safely cooperate and assess CONNECTor trustworthiness, so to provide CONNECTed Systems with the most trusted CONNECTor.

Whereas in this document the different dependability concerns are dealt with along separate tracks, in the next year we aim at developing an integrated framework. In Chapter 8 we attempt a first outline of a unified workflow of WP5 activities.

2 Dependability Metrics

In this chapter, we introduce a conceptual framework for refining and classifying dependability attributes in CONNECT. To do this, we first overview the basic concepts and definitions from the literature and then propose a refinement scheme through which classic (or generic) dependability metrics can be customised to reflect the CONNECT vision.

2.1 Terminology

The CONNECT vision requires to ensure that the CONNECTed Networked Systems inter-operate as intended, not only concerning their functionality, but also concerning non-functional properties, such as performance, security or trust. Unfortunately, there does not exist a commonly agreed taxonomy comprehensive of all such concerns that we can import as a whole. Historically, different communities have evolved different terminology and concepts, also sometime with not consistent use of terminology. An important and highly cited reference is [15], in which dependability and security concepts are reconciled, however such paper, for instance, does not consider explicitly concepts relative to performance or trust.

In this deliverable, we use the term "dependability" with two different senses: a stricter one, consistent with its definition in [15] (see Figure 2.2 below); and a broader one (such as in this chapter title) in which the term dependability is meant as a label inclusive of all the different concerns listed above (see Figure 2.1), although this broad acception is not adherent to the usage of this term in the literature. The context will make clear if we mean dependability in strict or broad sense.

Another related term that is subject to different interpretations in different fields is QoS (Quality of Service). QoS is a high-level generic concept which characterises the ability of a system to meet certain specified needs [1]. In networks, QoS generally refers to performance-related attributes (such as transmission rates and error rates). In computer systems, QoS includes performance, but is often used in generic sense to characterise the non-functional properties of a system. Therefore, it is often considered a concept similar to dependability (see, e.g., [39]), although more oriented towards the user's perspective (while dependability would be more oriented towards the designer's point of view).

In this deliverable, to prevent confusion, we avoid further using the term QoS.

2.2 Motivation

Metrics are quantitative / qualitative indicators suitable to point out the ability of a system to accomplish its intended service according to its specification or user requirements. Classical metrics of computer systems, such as dependability, performance, security and trust, need to be conveyed in the CONNECTed world, where the provided service may change in accordance with the evolution of the environment.

Indeed, as already mentioned in Chapter 1, new Networked Systems may join or leave the network, possibly in unforeseen or even unpredictable manners. Hence, CONNECT Enablers may need to synthesise CONNECTors on-the-fly, even when the knowledge on the behaviour and capabilities of some Networked Systems is still incomplete. In these cases, as illustrated later in Figure 8.2 of Chapter 8, CONNECT Enablers may initially synthesise a basic CONNECTor that permits only some elementary form of interaction, and an enhanced CONNECTor may be synthesised only in a second phase, when CONNECT Enablers have learnt the behaviour of the new Networked Systems. This iterative approach to the synthesis of CONNECTors is represented in Figure 8.2 by the various loops that through "Dependability analysis" return back to the "Synthesise CONNECTor" activity.

Due to the on-the-fly CONNECTor synthesis, it may happen that new Networked Systems may not be able to get access to a service that is already available in the network, e.g., because CONNECT Enablers take some time to synthesise a proper CONNECTor. In certain cases, some Networked Systems might never be able to get a service, e.g., because of critical functional or non-functional mismatches that cannot be properly bridged by the Enablers with the resources available in the network. Similarly, Networked Systems of an already CONNECTed System may experience service discontinuity during time since different CONNECTors might be used in different time frames.

CONNECT metrics need to take into account the above-mentioned factors. In order to specify CONNECT metrics, we define a conceptual framework that refines classical dependability metrics along a CONNECT-*dependent* dimension and a *context-dependent* dimension. In this Chapter we show how the two dimensions defined in the framework can be used to customise classical dependability metrics with respect to the four actors of the CONNECT architecture (see Figure 1.1), to the application scenario, and to the heterogeneous / evolvable aspects of the actors of CONNECT.

The presentation proceeds as follows. First, to make the document self-contained, we recall well-established definitions and concepts that are typically used to evaluate non-functional attributes of computing systems (readers that are familiar with the subject, may safely skip this part). Second, starting from the above generic definitions, a conceptual framework that can be used to derive and classify CONNECT metrics is defined. Third, an example of derivation and classification of CONNECT metrics is developed for the CONNECT scenario "*Stadium Warning System*" described in Deliverable D6.1 [54].

2.3 Basic Concepts

A computing system is characterised by a *functional specification*, i.e., a description of what the system is intended for, and a *non-functional specification*, i.e., a description of how well the system is supposed to provide its intended service. If the computing system complies with its functional and non-functional specification, then the system provides a *proper* service; otherwise, the provided service is *improper*. The transition from proper to improper service is referred to as a *failure*. *Service restoration*, on the other hand, is a transition from incorrect to correct service. The (part of the) system state that generates a failure is called *error*, and the hypothesised cause of the error is called a *fault*. The cause-effect relation between faults, errors and failures is known as the *Fault* $\rightarrow$ *Error* $\rightarrow$ *Failure* chain. Errors can be either *detected*, i.e., their presence is pointed out with proper messages and signals, or *latent*, they are present but not detected. A fault is *active* if it leads to an error, otherwise it is *dormant*. A computing system is defined *robust* when it is able to provide a proper service even in situations that exceed its specification [12]. A computing system operates in a *degraded mode* when a subset of the implemented services fails but the provided service is still acceptable since it is compliant with the specification of the service.

In order to reduce the number of failures and their severity, computing systems can be instrumented with mechanisms that break the *Fault* $\rightarrow$ *Error* $\rightarrow$ *Failure* chain. These mechanisms can be grouped in four main categories [15]: fault prevention, fault tolerance, fault removal, fault forecasting. *Fault prevention* aims at eliminating all possible faults; in real-world systems, this is generally accomplished with design and implementation choices that reduce the probability of a system failure to an acceptably low value. Fault prevention techniques can be applied during all phases of the system lifecycle. *Fault tolerance* [14] expects failures to occur and deals with mechanisms suitable to compensate failures with proper counteractions. Fault tolerance can be carried out via *error detection* (identification of errors), and *system recovery* (elimination of identified errors and prevention of a re-activation of their causes). *Fault removal* is intended to detect and remove faults introduced during all phases of the lifecycle of a computing system. During the development phase, fault removal consists in *verification* (checking system properties), *diagnosis* (identification of the fault), *correction* (fault removal). *Fault forecasting* deals with estimating present and future faults. Techniques for fault forecasting can be either *qualitative*, i.e., they identify, classify and rank events that possibly lead to service failures, or *quantitative*, i.e., they provide a probabilistic estimation of the occurrence of service failures.

2.4 Dependability, Performance, Security and Trust

The broader acception of dependability in CONNECT includes four classes of attributes: *dependability*, *performance*, *security* and *trust*. In this section we recall from the literature their main characteristics. A discussion on the implications of sociological aspects on dependability attributes is elaborated in Chapter 3.

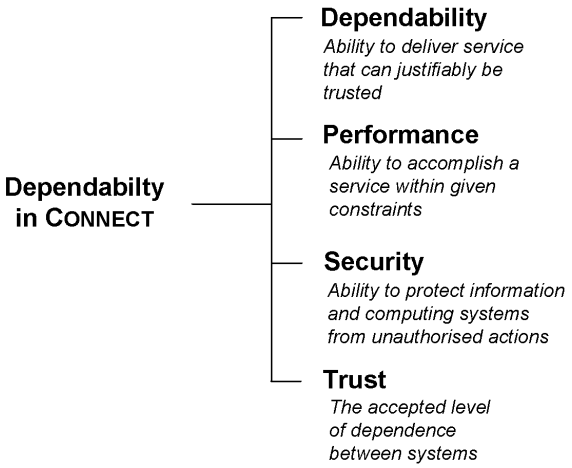


Figure 2.1: Dependability in CONNECT

Dependability

Dependability is defined as the ability of a system to provide its intended services in a justifiable way [116]. Such ability of the system is generally measured against the following attributes (see Figure 2.2): *availability*, *reliability*, *safety*, *integrity*, *maintainability*. *Availability* is defined as the readiness for correct service and is generally computed as the ratio between the up-time of the system to the duration of the considered time period. *Reliability* is defined as the continuity of correct service and is typically expressed by using the notions of mean time between failures (MTBF) and mean time to recover (MTTR) for repairable systems, and with mean time to failure (MTTF) for non-repairable systems. *Safety* is the absence of catastrophic consequences. This attribute is a special case of reliability: a safe state, in this case, can be either a state in which a proper service is provided, or a state where an improper service is provided due to non-catastrophic failures. *Integrity* is defined as the absence of improper system state alterations. *Maintainability* is the ability to undergo modifications and repairs.

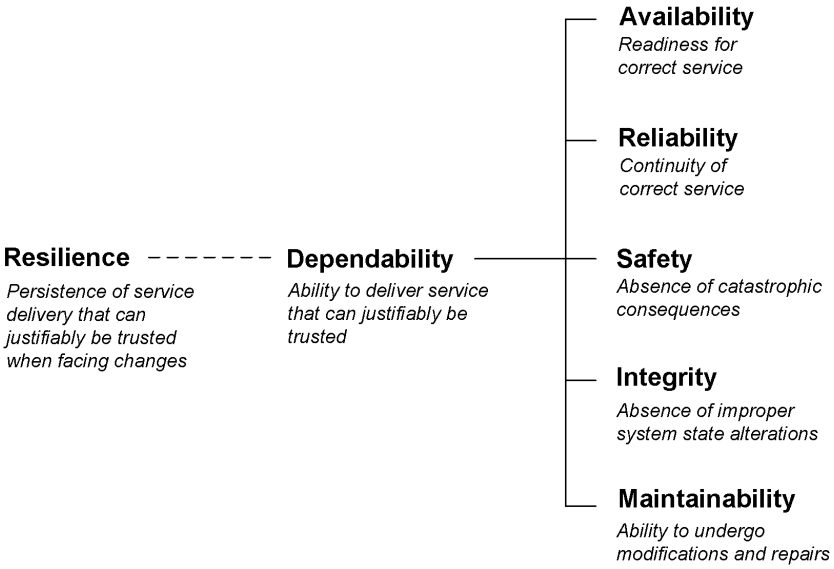


Figure 2.2: Classical dependability attributes and resilience

Resilience: Dynamic and evolvable systems generally need to cope with unanticipated conditions that might cause system failures. In these cases, the concept of dependability can be naturally extended to *Re-*

silience, i.e., the persistence of service delivery that can justifiably be trusted when facing changes [117]. Possible changes can be classified according to their *nature* (e.g., functional, environmental, technological), *prospect* (e.g., foreseen, foreseeable, unforeseen), and *timing* (e.g., shot, medium or long term).

Performance

Performance [2] is the ability of a system to accomplish its intended services within given non-functional constraints (e.g., time, memory). Typically, performance of a system can be characterised with the following attributes (see Figure 2.4): *timeliness*, *precision*, *accuracy*, *capacity* and *throughput*. *Timeliness* is the ability of the system to provide a service according to given time requirements, e.g., at a given time and within a certain time frame. *Precision* is the ability of the system to provide the same results when repeating measurements under unchanged conditions. *Accuracy* is the ability of the system to provide exact results, i.e., results that match the actual value of the quantity being measured. *Capacity* is the ability of the system to hold a certain amount of data / handle a certain amount of operations. *Throughput* is the ability to handle a certain amount of operations / data in a given time period.

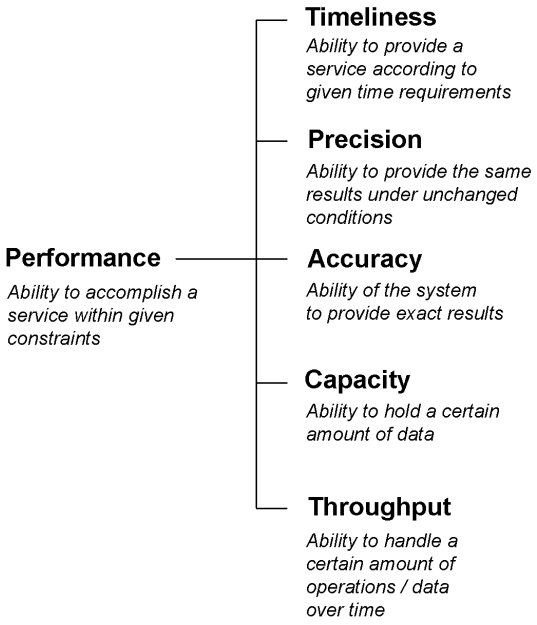


Figure 2.3: Performance attributes

Performability: This indicator combines the concepts of performance and dependability, and has been introduced to evaluate degradable systems, i.e., systems that are still able to provide a proper service when facing faults, but with degraded level of performance. *Performability* is the ability of a system to accomplish its intended services in the presence of faults over a specified period of time [140]. Performability allows to evaluate different application requirements and to assess dependability-related attributes in terms of risk versus benefit.

Security

Security is the absence of unauthorised access to, or handling of, system state [15]. The concept has been developed for many years from different communities of researchers and practitioners; for this reason, security has slightly different meanings in different fields. Nevertheless, security properties can be generally defined in terms three basic attributes (see Figure 2.5): *confidentiality* (absence of unauthorised disclosure of information), *integrity* (absence of unauthorised system state alterations), *availability* (readiness for authorised actions).

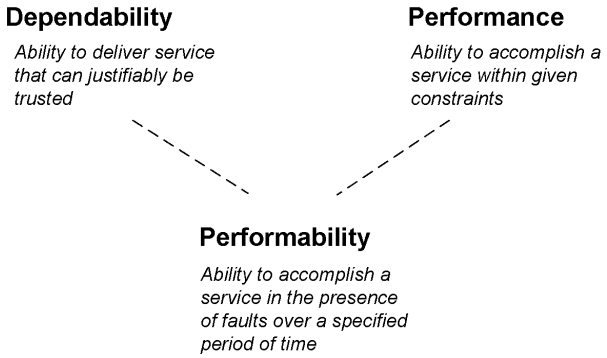


Figure 2.4: Performability and its relation with Dependability and Performance

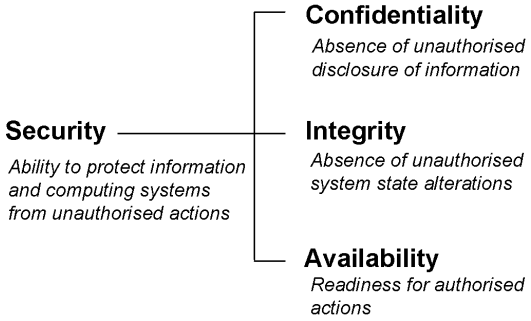


Figure 2.5: Security attributes

Trust

Trust is the accepted level of dependence between systems [15], i.e., the dependence of a *trustor* (the party that decides whether to trust) and a *trustee* (the party whose trust is being evaluated) together with a judgement that the level of dependence is acceptable [193]. Trust can be measured against the following attributes (see Figure 2.6): 3^{rd} *party*, *action*, *capability*, *competence*, *confidence*, *context*, *history*. 3^{rd} *party* links the level of trust to information provided by external entities, e.g., recommendations. *Action* takes into account what the trustee is trying to do, e.g., message relay, bank transfer. *Capability* is the ability of the trustee to perform a task; this attribute has a specialised meaning in the field of security, where trust levels may depend on access rights. *Competence* gives a trust level depending on the expertise level of the trustee. *Confidence* takes into account the variability of trust factors over time. *Context* considers the status of the system at a certain point of time when evaluating the trustee. *History* links the trust level to the past behaviour of the trustee.

2.5 CONNECT Metrics Framework

The classical dependability metrics defined in the literature and briefly surveyed in the previous section are very useful to give a conceptual classification of different concerns in assigning reliance on a system. However they are quite generic, and are not appropriate to qualify concrete metrics of dependability in CONNECT scenarios.

We discussed actively what are relevant metrics of dependability in CONNECT. To this purpose, we initially proceeded in top-down fashion by collecting from all partners of the CONNECT Consortium their expectations in terms of CONNECT dependability metrics. The results from this process are reported below in Section 2.5.3. Then we applied such metrics in a bottom-up process to the scenarios developed in Deliverable D6.1. However, we realized that our attempt to determine once and for all a fixed set of CONNECT metrics was not meaningful, because from one scenario to another we may be interested in

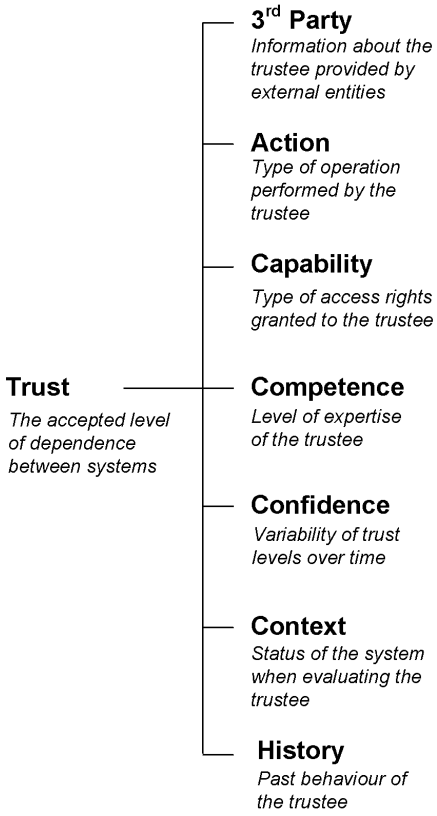


Figure 2.6: Trust attributes

guaranteeing very different properties.

Finally, from this process, we identified as a more interesting and broad-scope solution the definition of a conceptual framework through which the *generic metrics* of dependability can be, from time to time, refined in each different domain to obtain concrete and relevant metrics. Working from this idea, we propose here a first definition of this conceptual framework, which, of course, is expected to be further extended / enhanced during the project lifetime. It is developed along two dimensions that we identified as primarily useful to refine classical dependability metrics:

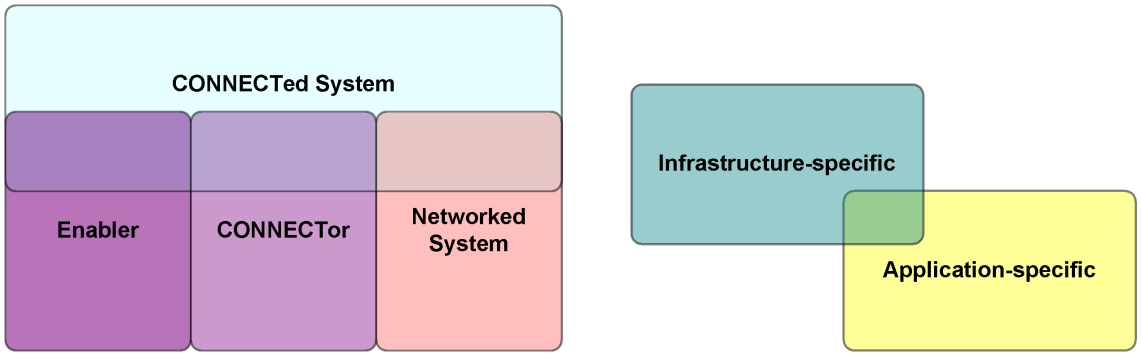
(i) a CONNECT-*dependent* dimension, which considers the four actors of the CONNECT architecture (see Figure 1.1) described in Deliverable D1.1;

(ii) a *context-dependent* dimension, which takes into account the application scenario and heterogeneous / evolvable aspects of the actors of CONNECT.

2.5.1 CONNECT-*dependent* dimension

The CONNECT-*dependent* dimension refines and classifies generic metrics according to the structural roles of the CONNECT architecture. The rationale behind this dimension is that different definitions of classical dependability metrics may be given for the different actors of the CONNECT architecture, namely the Networked Systems, the Enablers, the CONNECTors and the CONNECTed System.

With reference to the above actors, the CONNECT-dependent dimension includes three disjoint classes: *NetworkedSystem-specific*, *Enabler-specific*, CONNECT*or-specific*, plus a fourth partially overlapping class, CONNECT*edSystem-specific* (see Figure 2.7(b)). The *Enabler-specific* and CONNECT*or-specific* classes can be used to obtain "internal" CONNECT metrics, i.e., metrics suitable to assess the dependability level of the CONNECT service.



(a) CONNECT-dependent refinement (b) Context-dependent refinement

Figure 2.7: Refinement dimensions of the proposed conceptual framework

2.5.2 *Context-dependent* dimension

The *context-dependent* dimension refines and classifies generic metrics according to the application context. The rationale behind the choice of this dimension is that CONNECT metrics can be linked to a particular application scenario and / or to heterogeneous and evolutionary aspects of the different actors of the CONNECT architecture. Indeed, as reported in Deliverable D1.1, CONNECT Enablers can accept CONNECT requests from different Networked Systems, discover new Networked Systems, gather / learn information on functional and non-functional behaviour of new Networked Systems, and synthesise, at run-time, new CONNECTors which allow inter-operation among Networked Systems willing to interact.

According to the CONNECT vision, the context-dependent dimension includes two partially overlapping classes (see Figure 2.7(a)): *application-specific*, which refines generic metrics on the basis of the application domain, e.g., safety-critical, delay-tolerant, real-time; *infrastructure-specific*, which refines generic metrics according to heterogeneity and evolution capabilities of the different actors of the CONNECT architecture, e.g., timeouts adopted by communication protocols, number of operational phases.

2.5.3 The initial set of *Generic* metrics

According to the feedback obtained from the partners of the CONNECT consortium, the initial set of generic metrics that will be considered in the project is shown in Table 2.1. The table is organised in four parts, each of which is related to a non-functional class of attributes, i.e., dependability, performance, security, and trust.

Generic Metrics (initial set)		
Type	*Name*	*Synthetic Description*
Dependability related		
	Failure Probability	Probability that an operational / repaired system fails (at steady state or in a transient period)
	Catastrophic Failure Probability	Probability that a system failure is likely to lead to unacceptable consequences (at steady state or in a transient period)
	Safety	Probability that a system does not fail in a manner that causes catastrophic damages (at steady state or in a transient period)
	Coverage Probability	Probability for a system to deliver a service to a certain percentage of users
	Completeness Probability	Probability that a complete service is delivered to a certain percentage of users
	Access Probability	Probability that a service is accessible under certain specified operating conditions
	Retain Probability	Probability that a service, once obtained, will continue to be provided under given conditions for a given time frame
	Boundary Reliable Behaviour	Maximum/minimum expected correct operations performed by the system while providing a service
	Mean Time to Failure	Mean time to failure for a system/Networked System or, more in general, to the provision of a x% degraded service
	Unavailability	Mean accumulated down time by the system in providing a service in a specified time interval
	Survivability	Probability that a system fulfils its mission in the presence of attacks, failures or accidents

Type	*Name*	*Synthetic Description*
Performance related		
	Latency	Maximum/minimum/average expected delay incurred in communicating a message
	Processing Time	Maximum/minimum/average expected amount of time a system takes to process a request
	Response Time	Maximum/minimum/average expected time from when a system makes a request until it receives a response
	Jitter	Maximum/minimum/average expected time variation for a system to provide its intended service
	Transfer Rate	Maximum/minimum/average expected data transferred in a time unit
	Message Delivery Rate	Maximum/minimum/average expected messages successfully delivered in a specified time interval
	Processing Capacity	Maximum expected operations a system is able to complete in a specified time unit
	Schedulable Utilisation	Maximum/minimum/average percentage of time a system can be busy while still meeting its timing requirements
	Resource Utilisation	Maximum/minimum/average percentage of system resources used at the same time
	Reward Rate	Rate of accumulation of reward at a specified interval of time or in steady state
	Cumulative Reward	Amount of reward accumulated by the system during a specified interval of time
	Timely Coverage Probability	Probability for a system to deliver a service to a certain percentage of users in T seconds
	Timely Recovery	Probability that a failed system gets repaired in T seconds
Security related		
	Authenticity	Ability of a system to protect message content and origin from unauthorised modification
	Confidentiality	Absence of unauthorised disclosure of information
	Repudiability	Ability of a system/service of denying that an action occurred
	Mean time to a “security failed” state	Mean time for the system to reach a state where security properties do not hold
	Vulnerability coverage	Average number (or percentage y) of detected Level X vulnerabilities
	Mean time to vulnerability detection	Mean time to discover Level X vulnerability
Trust related		
	Confidence	Probability that a trusted system performs a particular action
	Recommendation	Probability expected by a trusted system to perform a particular action
	Reputation	Historic of the trusted system
	Malicious Detection	Probability to isolate malicious entities

Table 2.1: The initial set of *Generic* CONNECT metrics

2.6 Example of CONNECT Metrics

In this section, as an exercise of application of the conceptual metrics framework described before, we derive and classify CONNECT metrics for the "Stadium Warning System" scenario, which is one of the CONNECT scenarios described in Deliverable D6.1 [54].

We briefly recall from Deliverable D6.1 that the Stadium Warning System scenario deals with a service used in emergency situations. The goal of the warning system is to dispatch, in the case of danger, an alert message to all people in the stadium. People are assumed to have smart-phones, and the alert message signals a risk and provides instructions to guide people to proper exits. This scenario has been selected since it can be used to emphasise a number of dependability aspects that are of special interest in CONNECT.

The example is elaborated as follows: (i) we select a subset of generic metrics from Table 2.1; (ii) we derive CONNECT metrics that are relevant to the Stadium Warning System by refining the selected generic metrics along the different dimensions of the conceptual metrics framework defined in Section 2.5.

The selection of generic metrics used in the example, as well as of the refinements that we apply, are made just as a demonstration of the framework, and not as the definitive metrics or relevance for the Stadium Warning System.

Selected generic metrics

We extract from Table 2.1 the following four generic metrics:

- **Coverage Probability (Generic Reliability Metric)**: Probability for a system to deliver a service to a certain percentage of users.
- **Latency (Generic Performance Metric)**: Maximum/minimum/average expected delay incurred in communicating a message.
- **Timely Coverage Probability (Generic Performability Metric)**: Probability for a system to deliver a service to a certain percentage of users in T seconds.
- **Confidence (Generic Trust Metric)**: Probability that a trusted system performs a particular action.

CONNECT metrics derived from coverage probability

For each of the above generic metrics, we consider a class along each of the dimensions CONNECT-*dependent* or *context-dependent*, and within it we instantiate the generic metric into a more concrete metric. In the examples shown below, we construct a tag that specifies which class has been considered among those shown in Figure 2.7. For instance, the first set of metrics refined from coverage probability is collected below the tag: **CONNECT*edSystem-specific*, Application-specific**, which means that the system to which we refer is the CONNECTed System, and the service and users referred to in the definition of coverage probability are instantiated to the specific services and users of the application of interest, in this case the warning system. Note that in some case a tag includes only one class (i.e., we only refine the generic metric along one dimension), or three classes (this means that along the *context-dependent* dimension we put ourselves in the overlap of the two classes: *Application-specific* and *Infrastructure-specific*).

CONNECT*edSystem-specific, Application-specific*

- Probability that a certain percentage of smartphones display the same alert message when located in the same area
- Probability to deliver an alert message to all people in the stadium
- Probability to deliver an alert message to all people in the stadium when adversaries are present (assuming that adversaries are able to jam a certain number of messages)

Enabler-specific, Application-specific

- Probability of successful synthesis of a CONNECTor which allows to deliver an alert message to a certain percentage of people in the stadium
- Probability to improve the delivery ratio of a CONNECTor which allows to deliver an alert message to a certain percentage of people in the stadium

NetworkedSystem-specific, Application-specific, Infrastructure-specific

- Probability to successfully display an alert message on smartphones brand X
- Probability that a smartphone brand X receives duplicates of the same alert message

CONNECT*edSystem-specific, Application-specific, Infrastructure-specific*

- Probability to display alert message A on a set of heterogeneous smartphones located in the same area

CONNECT metrics derived from latency

CONNECT*or-specific*

- Time to deliver a message from n providers to m users

Enabler-specific

- Time to synthesise a CONNECTor between n providers to m users

CONNECT*edSystem-specific, Enabler-specific*

- Probability to synthesise a new CONNECTor which reduces message delivery time of d seconds

NetworkedSystem-specific, Infrastructure-specific

- Time to display a received message on smartphones brand X

CONNECT*edSystem-specific*

- Time to deliver a message to a given percentage of people which move at average speed X

CONNECT*edSystem-specific, Application-specific*

- Time to complete the registration process of a smartphone to the warning system
- Time to deliver an alert message to a given percentage of people located in the stadium

CONNECT*edSystem-specific, Application-specific, Infrastructure-specific*

- Time to deliver an alert message to a set of people with a certain percentage of heterogeneous smart-phones

CONNECT metrics derived from timely coverage probability

CONNECT*edSystem-specific, Application-specific*

- Probability that a certain percentage of smartphones receive an alert message in T seconds
- Probability to deliver an alert message in T seconds to all people in the stadium when adversaries are present (assuming that adversaries are able to jam a certain number of messages)

Enabler-specific, Application-specific

- Probability of successful synthesis of a CONNECTor which allows to deliver an alert message to a certain percentage of smartphones in T seconds

NetworkedSystem-specific, Infrastructure-specific

- Probability to successfully display a message in T seconds on smartphones brand X

CONNECT*edSystem-specific, Application-specific, Infrastructure-specific*

- Probability to display the same message in T seconds on a set of heterogeneous smartphones located in the same area

CONNECT metrics derived from confidence

CONNECT*edSystem-specific, Application-specific*

- Probability that a certain percentage of trusted neighbours re-broadcast an alert message

CONNECT*or-specific*

- Probability that a trusted CONNECTor allows communication with a certain percentage of trusted Networked Systems

Enabler-specific

- Probability that a trusted Enabler is able to synthesise a proper CONNECTor

NetworkedSystem-specific, Application-specific, Infrastructure-specific

- Probability that trusted smartphones brand X re-broadcast an alert message
- Probability that a certain percentage of mobile trusted neighbours re-broadcast an alert message

2.7 Discussion and Future Directions

The activity carried out in this task aimed at producing relevant metrics that can be used for V&V purposes and for ensuring long-lived usefulness of CONNECTed Systems. At the end of the investigations performed during this first year, we came to the conclusion that, rather than defining a new set of metrics, it is more useful to define a conceptual framework to obtain CONNECT metrics by refinement from classical dependability metrics.

The proposed framework allows us to derive and classify CONNECT metrics by using two refinement dimensions: (i) a CONNECT-*dependent* dimension, which considers structural aspects of the CONNECT architecture defined in Deliverable D1.1, where different actors have different roles; (ii) a *context-dependent* dimension, which takes into account the application scenario and heterogeneous / evolvable aspects of the actors of CONNECT. An example of derivation and classification of CONNECT metrics has been elaborated for one of the CONNECT scenarios, i.e., the “Stadium Warning System”.

We plan to investigate the appropriateness / completeness of the proposed metrics framework by validating its use in feeding verification and validation approaches, as well as security and trust activities. Consequently, extensions / modifications to the proposed conceptual metrics framework will be applied as needed in the course of the project.

3 Developing “Soft Metrics” for Dependability

This chapter introduces the idea of ‘soft metrics’ for dependability, The chapter reviews the literature and presents an argument for the involvement of a social scientific view of dependable socio-technical systems design based on the idea that, to be dependable, systems need to be appropriate both for the application domain and potential users. Before designers can solve a design problem they need to understand some basics - such as what they are designing, who should use it, how often and in what circumstances. Indeed, social analysis of settings where systems are deployed can expose subtle interactions and practices that are crucial to achieving this understanding, but might not be revealed by a technical analysis. This 'turn to the social' recognises a new kind of end-user, a ‘real time, real world’ human and suggests some of the ways in which social scientists can provide designers with insights and sensitivities, to inform dependable design.

The classical model of dependability overviewed in the previous chapter essentially views dependability as primarily a technical issue. Human factors [166] [168], whilst recognised as important, are regarded as somehow separate from the technical issues of dependability. Attempts to incorporate humans or ‘users’ into notions of dependability often seem to require that humans be treated much like machines. In the literature on dependability [165, 118, 15], there is an implicit assumption that the *Fault* → *Error* → *Failure* chain (see Section 2.3) applies equally to humans as it does to technical system components.

Of course, it is highly debateable whether such a techno-centric model can be applied to the humans that use these systems – and therefore whether it can apply to ‘systems in use’. The exclusively technical focus and orientation means that these dependability approaches do not properly consider the interactions between the user and the system and that, perhaps, additional, more social or human attributes, are required to understand the nature of dependability.

3.1 Dependability, Security, Privacy and Trust

“For most of us, most of the time, our natural attitude in the taken-for-granted world is one which enables us to maintain our sanity in our passage through life and the daily round. Routines, habits . . . and the consistencies with which our interactions with each other conform to expectations, together provide the infrastructure for a moral universe in which we, its citizens, can go about our daily business. Through learning to trust others we learn, one way or another, to trust things. And likewise, through learning to trust material things we learn to trust abstract things. Trust is therefore achieved and sustained through the ordinariness of everyday life and the consistencies of both language and experience [180].

In this section we deal briefly with some aspects of what everyday users might reasonably expect of a computing system – that it should be dependable, trusted and secure. Of course, these are complex notions and have long been the subject of debate. For example, there are a number of different theoretical approaches that have been taken to the study of trust, from Axelrod’s calculative model [17], through to Luhmann’s processual model [129]. Luhmann’s central point is that all approaches fail to pay attention to the *social process of trust production*, i.e., they leave unspecified “the social mechanisms which generate trust” [130]. Rather than emphasising what trust does, investigations of how trust is achieved, how it can be seen in action, is needed. Our concern as far as developing some understanding of how trust impacts on user notions of dependability requires that we take on board Luhmann’s recommendation to look at trust accomplishment as a social process.

If our interest lies in developing some metrics for dependability and trust then we need to understand how trust is accomplished as a mundane feature of everyday life, the ways in which trust enters into our everyday life, how we make judgements to trust or distrust people and things. Computer systems, databases, expert systems etc pervade the everyday world. Using technology is an everyday matter and part of this ‘everydayness’, part of its mundane, taken for granted character lies in trusting the technology – at least until given a good reason not to. Since trust is such a routine background expectancy in both everyday interaction and in our everyday use of technology, the interesting design issue becomes determining, investigating, exactly how, in the everyday world, people orient to the question of the trustworthiness of a system.

In the same fashion, when it comes to security, a whole series of explorations of technology and human factors highlight a range of significant problems concerning what might be achievable in terms of levels

of security. These problems are significant in the sense that unless they are resolved, and even when technologies are available and usable, they still may not be used. Accordingly, we feel that it is critical that we look not just at technologies but also at how those technologies are incorporated into users' security practice. Some research has been carried out. For example, Friedman et al. [78, 79] have investigated users' perceptions of security in Web-based interactions, and noting how problems arise in how accurately people are able to assess the security of settings they encounter.

We are interested in developing some understanding of and metrics for what might be regarded as 'usable security' – security that takes into accounts user concerns. But failure to correctly identify stakeholders and stakeholder values; to identify and meet users' needs are at the heart of many of the problems of "usable security". Dourish et al [67], for example, considered the attitudes that individuals had towards their encounters with security technologies, charting the expectation—and the disappointment – that stemmed from users conceptualizing security as a broad protective barrier. Most importantly, Dourish et al [67] suggest:

"Although the mathematical and technical foundations of security systems delimit the scope of ''security'' for the research community, end users see their encounters with security quite differently and set the scope of concerns more broadly".

Like Dourish et al ([67], [66]) then, we also wish to stress the ways in which security issues and perceptions are inevitably embedded in complex social and cultural contexts. We wish to focus our concern on some of the important socio-technical issues involved in determining security requirements in various settings. It is in this sense that this research requires some interdisciplinary sensitivities, not least in a concern with exactly how certain basic questions might be posed to users: what are users afraid of?, what do they value that's worth protecting?, how much do they want to be involved?, how automated do they want the process to be? etc.. and what weight do we need to attach to their answers?.

Friedman et al [77] use a prolonged semi-structured interview (and a drawing task) to understand users conceptions of security (in this case, Web security) and the types of evidence users call on in evaluating security. They demonstrate that many users mistakenly evaluated whether a connection was secure – and that this was not always a simple function of expertise. Even users who correctly recognized the security status of connections sometimes did so for completely incorrect reasons. The starting point for Edwards et al. study [70] is the way in which end-users are often perceived as the 'weak link' in information security and consequently there has been some emphasis on automated approaches. From a user perspective, generally using their system to accomplish some task, security can appear irrelevant or marginal to the task at hand (see [26]). Managing their security is generally not a goal for users, and their motivation to actively manage their security correspondingly low. Edwards et al suggest that although security automation may appear potentially beneficial, in practice it is not a security panacea. The automation approach is predicated on a set of (often incorrect) assumptions about technical, social, and environmental contexts in which security decision-making occurs. Therefore there are inherent limitations to how well automation can succeed in practice even if the technology behind it is faultless. They outline a number of technical and social factors that mitigate against the acceptance and efficacy of automated end-user security solutions and conclude by emphasizing increasing the acceptance and efficacy of security solutions for end-users:

"These limitations can lead to "failures" of automation that are not only technical (i.e., when the automation system simply stops working, for example), but are also failures of meeting the actual needs of the users. We further contend that such failures to meet users' needs are not simply annoying nuisances, but rather are at the heart of many of the problems described under the rubric of "usable security" [70]

Such a failure to meet users' needs and aspirations also has a political or power dimension with the users being usually and essentially powerless and having to live with decisions made by others;

"When security decisions are automated, the values behind these decisions are those of "empowered" sources rather than the users who will themselves be affected by, and must ultimately live with, these policies. . . . Problems arising from embedded values can range from subtle distinctions around perceptions of proper use, up to flagrant abuse of power.. . . an innate power differential exists between the person who sets or defines the security automation and the end-user who has to live with this decision; problems may arise when the values of these two stakeholder parties do not align". [70]

3.2 Dependability - A Human Perspective

In technical models of dependability, humans are considered to be system elements that can be treated in the same way as other software or hardware elements. Laprie, for example, recognises the importance of human operators but discusses them in terms of 'interaction faults' resulting from 'human errors'. Failures on the part of humans in the operational system lead to these interaction faults which result in unexpected computer system state and hence computer system failures.

Human 'errors' and the relationships between these errors and system failures have been extensively discussed by authors such as Reason [168] and Rasmussen [166]. Rasmussen discusses different types of human errors such as skill-based, rule-based and knowledge-based errors and Reason, in his 'Swiss Cheese Model' relates human error to system failure. He suggests that human errors lead to system failures when they bypass the checks and protection built into a system. Researchers argue that so-called 'human errors' arise because the systems designers did not consider the practicalities of system operation in their design.

If we consider broader socio-technical systems and apply the technical dependability model to the people in these systems, the *Fault* → *Error* → *Failure* chain breaks down because, for people, the notions of fault, error and failure are complex and may be inapplicable:

Failure recognition People are not machines and they use their intelligence to develop or devise many different ways of doing the same thing, even in supposedly rule-driven actions – because people are 'rule-users' rather than simple 'rule followers'. An action that might be interpreted as a failure for one person (such as an air traffic controller temporarily placing aircraft on a collision course) might be part of a dependable operational process for another (because the ATC may have a reliable method of ensuring that they will move one of the aircraft before any real danger ensues). If a 'near-miss' ensues was the failure placing the aircraft on a collision course or failing to subsequently separate the aircraft?

Error identification. How can we tell if an unwanted state has resulted in the failure? The notion of explicit state is one that is particular to computer systems and is difficult to apply outside these systems.

Fault recognition. What was the fault that resulted in the human error? Was it a training fault or something more fundamental. People are not deterministic and their emotional and physical state profoundly affects their behaviour. The notion that failures in the development process lead to faults in the 'system' clearly does not apply to people.

For some classes of highly automated system, where operational processes are tightly defined and operators are highly trained, then the benefits of adopting a consistent view of dependability that encompasses both people and computers may outweigh the disadvantages of treating the human operators in a simplistic way. However, there are many systems that are discretionary whose use is not constrained by organisational processes and where users do not face sanctions if these are not used. For those systems, the notion of what is mean by a human 'error' or 'failure' is more difficult. If a user does not read a system user guide and hence makes an input error is that a human failure? Or, is this a system failure because the designers have made invalid assumptions about the reality of system use?

3.3 Dependability as a 'Socio-technical 'and 'Interdisciplinary' Issue

"While pervasive systems will most likely not be mission and safety critical, their criticality will increase as they will become an even more integral part of the infrastructure upon which our society depends. Although the decreasing size and price of devices make it feasible to build and sell pervasive systems, the complexity of such systems might prevent us to install, operate, and maintain them. With the increasing density of computers we need to learn how to install, operate, and maintain such systems automatically. Since it is not economically feasible to have a large number of system administrators available, pervasive systems need to be automatically configured and repaired. Pervasive dependability addresses the problem of how to make pervasive systems dependable without making them unaffordable. Pervasive dependability differs from traditional dependability in the sense that it targets pervasive systems instead of mission or safety critical systems. Designing a pervasive dependable system imposes a new set of requirements that – as we believe – cannot be solved using traditional dependability approaches." [75]

When defined as “the ability to deliver service that can justifiably be trusted” – dependability [15] has a number of attributes, as presented in Chapter 2. But as we consider broader, socio-technical, notions of “system”, the ability to achieve a clear and documented understanding of the intended service of the system - and hence some view of dependability - becomes increasingly difficult. In these circumstances we may need to broaden our understanding of what dependability means to develop a more nuanced notion of 'dependable systems'. Instances of undependability in many settings are not normally catastrophic, but are rather mundane events. Dependability can then be seen as being the outcome of people’s *everyday, coordinated, practical actions*. Workers draw on more or less dependable artefacts and structures as resources for achieving overall dependable results in the work they are doing

The work of the ‘Dependability’ IRC in the UK – in which Lancaster University was a member – extended the technical notion of dependability to consider broader, socio-technical notions of ‘system’ and ‘service’ by incorporating cultural, organisational, and interactional considerations. In this view, even if technology is assessed as fit for purpose according to formal models of dependability; problems with the usability and acceptability of the technology bring into question the dependability of the system *as a whole*. Considering dependability as a socio-technical issue thus provides additional criteria to those of performance metrics for modelling the dependability of systems, to include matters such as user acceptability and the ability of a system to be able to adapt to different environments of use and different users. Consequently, dependability may not be simply technically *measured* but socially and organisationally *judged* and it is exactly some of those judgements we are interested in eliciting and understanding.

In the everyday world dependability means much more than is meant by the technical definition, dependability is very much an everyday, not just a technical matter. In the everyday world, systems that may be ‘formally’ defined not to be dependable, may, nevertheless, be perfectly usable for the practical purposes of accomplishing the work they support. In these terms, although they may not be formally dependable they may be ‘trusted’ by their users and within the organisation. Within the everyday world, they may be dependable enough for the practical purposes of getting the job done – whatever that might be - within the practical circumstances in which they are used.

Unlike machines, people are non-deterministic, self-aware, and thoughtful. Identifying faults, errors and failures in human activities is often complex and difficult, just as determining and assigning responsibility is difficult. In general explicit process descriptions are not part of the social life of most people and identifying ‘human error’ as a deviation from a process is not especially meaningful or helpful. Most importantly and relevantly, it is the human users that generally make decisions about how ‘dependable’, how ‘reliable’, how ‘trustworthy’ etc the systems they use are – and these are decisions that are rarely informed by mere technical factors but by a range of experiences and expectations of use. These experiences and expectations, these requirements, are often highly individual – they vary from one person to another, and from one context to another – and therefore are difficult to simply or rigidly quantify. These are what we refer to as ‘soft metrics’.

3.4 Developing ‘Soft Metrics’

The dependability of systems extends beyond the hardware and software into the social and lived experience of the user. It is essential that dependability does not just mean that a system behaves according to the expectations of its designers. Systems have to be designed so that they are *acceptable to* and *accepted by* users.

‘Soft metrics’ are concerned with developing measures that involve an understanding of users and their perceptions of the dependability of any particular system. This will include a range of judgements as to a system’s reliability, trustworthiness, timeliness, adaptability and more. These judgements may vary according to the context in which a system is used and who is using it. These judgements may well determine whether and how any system actually gets used – if it is used at all.

Key to these kind of ‘soft’ dependability measures is attempting to understand users and the contexts in which devices or systems get used – since these determine the notion of dependability adopted. The key problem any technology is how to ensure that the devices or installation meets the perceived needs of the user. To achieve dependability, it is necessary to take an approach that integrates the user and environment with the technology rather than simply focusing on technical issues and the operational processes involved in using a system.

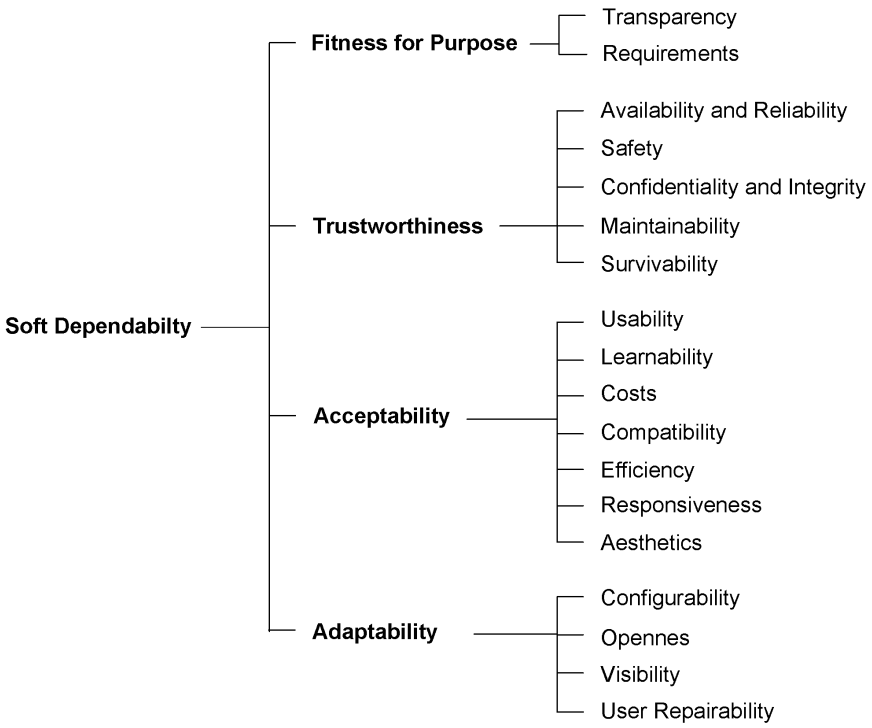


Figure 3.1: Soft-Dependability attributes

The framework of 'soft metrics' for dependability, or, briefly, *soft-dependability*, that is proposed draws heavily on the work of Dewsbury [62] and his metrics for dependable household systems with particular reference to disabled users. Although this model was created specifically to support the design of dependable systems in the home, it should be applicable to any other systems where the user has a choice of whether to accept or reject them, since where people have discretion over whether or not to use a computer system, issues such as fitness of purpose, acceptability and adaptability are likely to be important factors. The model, depicted in Figure 3.1, organises the characteristics of technology systems under four headings each of which represents a set of system attributes. These general headings are applicable to a range of technologies and reflect the functionality of the system, its technical quality of service, the suitability of the system for a particular user in context and the ability to the system to change in response to changing user needs.

The attributes reflect those identified by Batavia and Hammer [24] who identify seventeen factors - such as consumer and supplier repairability, personal acceptability and dependability - that affect the selection and evaluation of technology. Dewsbury's [62] model takes these further by considering 'fitness for purpose' as a critical dependability element and by decomposing dependability into the more specific attributes discussed by Laprie ([118, 15]) and Sandhu ([177]).

The top-level headings that encompass related system attributes are:

- ***Fitness for purpose*** The fitness for purpose of that system reflects the extent to which that system meets the real needs of its users. In short, is it the right system for the user's functional needs.

- ***Trustworthiness*** In order for a system to be dependable, the user must be confident that the system will provide the expected services when required and without undesirable side-effects. The trustworthiness of a system reflects the technical dependability attributes of availability, reliability, safety, etc.

- ***Acceptability*** The heading embraces those system attributes that govern whether or not the user of a system is willing to accept it for use. These relate to the ease of use of the system, its efficiency and compatibility with other systems and in context, rather than its technical dependability charac-

teristics. If a system is not accepted by a user then it clearly does not deliver the services that are required.

- ***Adaptability*** Within any social setting, both the context and the users of the systems change. If system dependability is to be maintained, then a system must be able to evolve over time, ideally without interventions from the system's designers.

The system attributes are covered by these broad categories. Of course, there are overlaps between these attributes but the broad categorisation provides a useful basis for describing the dependability model. Not all attributes are relevant for all users and all systems. There are also possible conflicts between attributes – making a system more usable may mean that it is less secure; making it configurable may make it both less usable and less transparent. Increasing reliability may reduce responsiveness because of the time required for additional checks carried out by software. There are no easy answers to resolving these conflicts – people rely on the judgement of the system designer to decide which attributes are most important.

3.4.1 Fitness for purpose

The fitness for purpose of a system reflects the extent to which that system meets the real needs of its users in all conceivable situations of use. A system's fitness for purpose depends on both the functional requirements for the system and what we term its 'transparency'. That is, the extent to which the functionality of the system is visible to and can be accessed by a user. Functionality that cannot be readily accessed is unlikely to be used.

3.4.2 Trustworthiness

The trustworthiness of a domestic system in Dewsbury's model corresponds closely to the technical notion of dependability recalled in Figure 2.2. That is, the trustworthiness reflects the availability, reliability, safety, confidentiality, integrity, maintainability and survivability of the system [16]. In most settings trade-offs between these attributes and between these attributes and other system attributes such as performance and cost inevitably have to be made.

The availability of a system reflects its ability to deliver services when required; The reliability of a system reflects its ability to deliver services as expected by the user. The safety of a system reflects its ability to operate without risk of injury or harm to users and the system's environment. Confidentiality and integrity are related security attributes – confidentiality reflects the system's ability to limit access to the system and its data to authorised agents and integrity reflects its ability to ensure that the system and its data are not corrupted. Maintainability is the ability of a system to undergo evolution with the corollary that the system should be designed so that evolution is not likely to introduce new faults into the system. This is in contrast to adaptability, which is the process of changing a system to configure it for its environment of use. Survivability is the ability of the system to continue to deliver essential services in hostile environments.

3.4.3 Acceptability

The notion of acceptability was initially conveyed through an advocate of Universal Design [177]. Universal Design is an inclusive approach to design where the designer tries to avoid design choices that exclude particular groups of users, such as users with disabilities. The model that Sandhu proposes considers the user and the technology together, reflecting the central significance of the user when considering dependability.

A user will only accept a system if they believe that the benefits that they receive justify the costs and effort of buying, installing, learning to use and using the system. If an unacceptable system is designed and deployed then it will either not be used or will not be used to its full potential. Hence, it can be argued, the required services are not being delivered to users and so the system is undependable.

In Sandhu's model (Figure 3.2), the acceptability characteristics for domestic systems are broken down into:

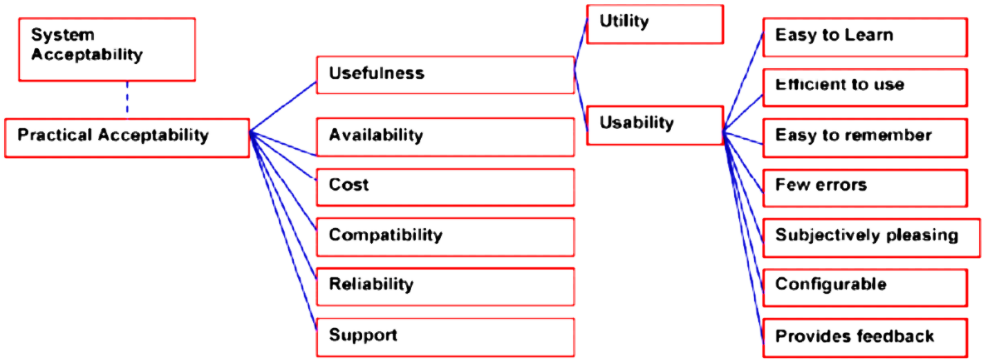


Figure 3.2: Sandhu's system acceptability model

1. ***Usability*** It must be possible to use the system on a regular basis without error and without having to re-learn how to benefit from the system.

2. ***Learnability*** It should be possible to learn to use the system without a steep learning curve before any benefits emerge.

3. ***Cost*** The cost of the system should be such that it is within the budget of the person or the organization buying the system. Over-engineering and 'gold plating' are undesirable.

4. ***Compatibility*** The system must be compatible both physically and electronically with other systems that are installed in the home.

5. ***Efficiency*** The effort and time saved by using the system must significantly exceed the effort involved in making use of it.

6. ***Responsiveness*** The system must respond in a timely fashion to user requests and provide feedback on its operation to the user.

7. ***Aesthetics*** If a system is to be actively used in the home, it should be aesthetically pleasing, blending in with the décor of the existing home and the users taste. If a system is aesthetically offensive (e.g. an industrial casing in a living room), experience has shown that some users will simply refuse to have it installed in their home.

Of course, these characteristics may conflict – and resolving such conflicts is a matter for the designer but the acceptability attributes are a means of highlighting potential conflicts and ensuring that these issues are not ignored.

3.4.4 Adaptability

CONNECT addresses a dynamic world, in which social spaces and the people living in them change over time. Spaces are reconfigured to cope with changing demands and tastes, new people come to live in the home, children grow up and the capabilities of older people typically decline as they grow older. Consequently, the requirements of users are constantly changing. If these systems cannot be adapted *in situ* to meet new requirements they are likely to become less and less used and, hence, less dependable.

Finally, it is well known that dependability problems in computer systems regularly arise because of errors made during system maintenance. These occur in spite of extensive quality control and testing mechanisms that are in place. This fact, along with the need to support system change leads to the following adaptability attributes that are relevant to dependability:

- ***Configurability*** This attribute reflects the ability of users or equipment installers to adapt the system to cope with a range of human capabilities such as variable hearing, eyesight, balance, etc.

- ***Openness*** This attribute is concerned with the system's ability to be extended with new equipment, perhaps from different manufacturers.
- ***Visibility*** This attribute reflects the extent to which the operation of the system can be made visible to users and installers of that system (e.g. does the system produce meaningful error messages). This is particularly important when problems arise as it increases the chances that these problems can be diagnosed without expert assistance.
- ***User repairability*** This attribute reflects the extent to which faults in the system can be repaired by users without specialist tools or knowledge. This is important as it means that the system can be brought back into operation quickly and the overall availability of the system is increased.

3.5 Methodological Issues in Developing 'Soft Metrics'

How can we best proceed to understand issues like dependability, trust or security ? And importantly, how can we understand it in a way that is useful and informative to design considerations? This section is interested in some of the methodological issues that emerge in developing some understanding of user notions about dependability and, thereby, producing 'soft metrics'. In order to develop soft metrics for dependability we need to consider ways in which we might go about gaining some insights into people's perceptions. Whilst the metrics themselves may not prove especially problematic – consisting, like many qualitative measures of a series of Likert like scales – exactly what we choose to measure, finding some agreement on acceptable measures - may well prove somewhat more difficult. We need to use a range of social science methods to provide some insights and suggest some possible measures for dependable systems. This section reflects on some of the approaches to understanding users that we have used at Lancaster University as a way of informing the methodological choices in CONNECT.

Whilst questionnaires and interviews are one obvious way of charting the user experience they often tend to be dependent on certain facets of that experience already being known, or known well enough to formulate some probing questions about experiences and expectations of dependability.

Another, and rather different, approach to developing relevant soft metrics is through ethnographic enquiry whose central characteristic is the researcher's detailed observation of how work or activity actually 'gets done'. Its focus is upon the circumstances, practices and activities that constitute the 'real world', situated character of any activity, whether that is formal work practice or the practices associated with hobbies, with everyday interaction and so on. This kind of ethnography being proposed here, ethnomethodologically-informed ethnography, has a strong background in this kind of setting. The defining feature of ethnographic study is the immersion of the researcher in the environment where a non-presumptive record is made of all aspects of the day-to-day work over an extended period of time. In this way a 'thick description' [82] is built up of the situated practices.

This kind of approach has had some success in recent years in CSCW (Computer Supported Cooperative Work) with such ethnographic methods being used to inform system design (see [98]). Ethnography has gained some distinction as a fieldwork method that could contribute both to a *general understandin*g of systems in use in a variety of contexts and to the *design* of distributed and shared systems. Efforts to incorporate ethnography into the system design process have had much to do with the (unfortunately belated) realisation, mainly among system designers, that the success of design has much to do, though in complex ways, with the social context of system use. If design, as a 'satisficing activity' is more of an art than a science, dealing with messy indeterminate situations and 'wicked problems'; then before designers can solve a design problem they need to understand some basics - such as what they are designing, what it should do and who should use it and in what circumstances. The 'turn to the social' recognised a new kind of end-user, a 'real time, real world' human being and consequently designers turned to the social sciences to provide them with some insights, some sensitivities, to inform design. Ethnography with its emphasis on the *in situ* observation of interactions within their natural settings seemed eminently suited to bringing a social perspective to bear on system design.

A more structured approach to understanding users and their concerns, about security, for example, involves the use of focus groups in identifying the most vulnerable 'assets' associated with an individual or group. One example of this is the Operationally Critical Threat, Asset and Vulnerability Evaluation (OCTAVE) method ([10]) used in business settings to determine an enterprise's critical assets and the

technical vulnerabilities associated with them. A process like OCTAVE brings to the fore security trade-offs, which will have an important role in forming a security strategy.

Another way at getting at user experience and understanding of aspects of dependability is through the use of 'probes' of various kinds. Probes have been deployed and described in various ways in HCI research [30] and have taken a number of different forms including, among others, Cultural Probes [81]; Informational Probes [59]; Technology Probes [99], [40]. We consider probes as a cluster of approaches and tools, some with considerable history in social science research (e.g. diary-keeping). Probes are designed to prompt and elicit information from people about their lives and 'local culture' [81]. They gather insights from within the site in question, as activities are performed and with the full cooperation and involvement of the participants concerned. Cultural probes have been used to understand people's lives – to probe their culture and access their rhythms and routines as well as to understand the kinds of technology designs that might work for them. Whilst obviously drawing from the idea of the space probe, broadly speaking probes consist of "an instrument that is deployed to find out about the unknown – to hopefully return with some useful or interesting data" [99]. *Technology* probes were first used as part of the interLiving project, [99] and subsequently have been used to uncover the use of situated displays by staff in a residential care setting [40]. Technology probes are:

"a particular type of probe that combine the social science goal of collecting information about the use and the users of technology in a real-world setting, the engineering goal of field-testing the technology, and the design goal of inspiring users and designers to think of new kinds of technologies to support their needs and desires". [99]

Central to the notion of Technology Probe is the possession of some form of embedded, relatively non-intrusive functionality enabling the monitoring of ongoing use of technology by users. [40] Technology Probes are deployed to produce digital data that supposedly helps with various forms of analytic work [59] ideally answering fundamental questions that are relevant for the ethnographer, engineer and designer: How is the technology being used? What needs to change? What should stay the same? What should we design next?.

Blogs can also act as a kind of Technology Probe collecting data on individual or community uses of technology and supporting the collection, sharing and dissemination of usage information. Studies of blogging show three primary types of blogs: individually authored personal journals, "filters" (because they select and provide commentary on information from other websites), and "knowledge logs" and that the majority of blogs are the personal journal "on-line diary" type [94]. How can such "on-line diaries" provide us insights into users' ideas about technology and dependability?

Nardi et al. [147] discuss how blogging is a social activity beyond diary-keeping that can be used for a number of purposes including: updating others, expressing opinions to influence others, seeking others' opinions and feedback, thinking through writing to an audience and the release of emotional tension. Of course blogs can be criticized as being self-indulgent, egotistical and little more than a monomaniacal digital journal to a disinterested and anonymous audience – the stuff of reality television and fashion magazine reader letters made public through the Internet. However, when set in the context of a community, whether digital, physical or hybrid, blogs can become poignant accounts of unfolding events. Blogs can have aspects of many forms of personal documents ([160]:14-33): they are part life history, part diary, part letter, part guerrilla journalism, part 'literature of fact' and they support media elements, such as photographs, videos and voice recordings. With this observation blogs, far from being overly individualistic, can reflect a community and its happenings from a humanized, contextualized and richly personal perspective. Indeed the blog, like a biography, has the opportunity to capture a subjective account of the ever-changing, constantly evolving relationship between community and technology.

3.6 'Soft Metrics' and the CONNECT Scenarios

In this section we are interested in linking some of the issues involved in developing 'soft metrics'; in understanding user ideas about dependability and finding ways to record and measure them; and the 'scenarios of use', the circumstances and applications envisaged by the CONNECT project where dependability of some kind might be important. In Deliverable 6.1 [54], several scenarios are provided in which different groups might have different ideas about the dependability of the system, ideas, moreover, that could change with time and circumstance. So for example, in the 'Stadium Scenario' the expectations of

security staff in terms, for example, of the reliability and trustability of picture quality might well change according to whether the pictures were being used to quickly direct security to troublesome areas of the stadium or whether they were to form part of the evidence in a later legal trial. Similarly sports fans might well have very different expectations of the quality of the images they might download and send on their mobile phones or similar devices depending on whether they were downloading them to broadcast and taunt opposing fans at the time or whether they wanted them to show friends or acquaintances later – ie, dependability in terms of the notion of 'timeliness' and quality might vary according to circumstance.

We were especially interested in examining the existing literature on the trialling of relevant devices and applications when applied to equivalent or similar technologies and scenarios of use, to uncover any relevance for the current CONNECT scenarios. Consider, for example, Dominic Lenton's scenario presented in 'The Small Screen'– *"...sitting on the train on the way to work, your smart new 3G phone rings to tell you that England have just taken the lead in Australia which kicked off while you were at home enjoying breakfast. Seconds later you're downloading a brief video clip"*. We were particularly concerned with what these small, experimental studies might tell us with regard to user interests and problems in advance of our own deployments and field trials.

One study of interest is Caj Sodergard's (ed) 'Mobile television - technology and user experiences' [181]. The project investigated interest in mobile TV by interviewing a number of people – public and 'experts' - and building and trialling a prototype system. Each user was asked to try the service at WLAN hot spots over a month. In terms of user experiences and reactions – and hence dependability attributes - the most liked feature was the ability to watch archived programs whenever the individual wanted. In terms of general use, typically users watched short programmes or pieces from longer programmes and typically the devices would be used when waiting for something - such as a train or bus; or when killing time in various ways.

Other studies that seemed relevant concerned the delivery of videos to mobile phones where there were a number of papers explicitly focusing on the user experience and user evaluations. Repo et al's [169] small 'field' study of 13 users - looked at situations where watching a video or clips from tv shows on a mobile phone was 'meaningful'. The situations they were interested in referred to a setting's physical and social contexts – what Dourish and Harrison call 'space' and 'place' ie – the concern was not merely with the physical layout or characteristics of the setting but some kind of 'meaning' attached to it that effectively turned it from a 'space' to a 'place'. This seems to suggest that dependability attributes may vary in importance according to where the user is when the application is being used. (This also seems interesting given Auge's work on 'non-places' – 'neutral' or essentially characterless places that people pass through, places of transit etc such as airport lounges, waiting rooms etc – since these may well be places or spaces where delivery of tv or video services to mobiles might be welcomed.) Repo's research question was *"in what kinds of situations is it meaningful to watch mobile videos". Their interest was in what they term* 'mundane reasoning' – which examines and analyses consumer choices and actions as routinised habits, practices and usage situations. As part of the research they gave a mobile phone with video capacity to users who watched videos in different situations which they then evaluated for 'meaningfulness' – users were 'asked to give genuine user experiences and critical feedback concerning the device and its services' and to add comments from family members and others. Participants were asked to watch video in specific pre-given situations – at the coffee table; on public transport; while teaching the use of the videophone; in connection with their hobbies and kept diaries of their, and others' responses. The choice of videos to watch was rather limited but Repo's study identified two different kinds of situations in which use of videos on mobiles seems 'natural':

- Avoiding boredom - to entertain themselves in boring situations such as a bus trip, or in queues;
- Having fun – sharing an experience (such as singing in a karaoke or watching a cartoon) – with others such as friends, or family.

The study identifies different kinds of sharing – sharing an activity, or sharing an experience and suggests that this is different to the avoidance of boredom in that it is 'social' – with social relations being enacted in front of the phone rather than through the phone. Although, like most of this kind of study, it is based on a very limited sample over a very short time-frame, a number of themes were identified:

- User experience and perception – the study notes an initial enthusiasm followed by an eventual (and quite rapid) tiring, noting the contrast between user expectations of the device and the application

and their subsequent disappointment which was associated with the small screen size and the boring content of the videos on offer – especially the TV clips

- Variations in use – the mobiles were used for both private and collective viewing; surprisingly perhaps most of the users favourable responses were to the use of the application for karaoke
- Good and bad viewing experiences;
  - good way of passing time, e.g., when waiting; or in the back of a taxi
  - positive experience watching at home with children
  - experiences sometimes linked to technical issues – technical features were mentioned as hindering use – especially inferior quality of sound and image and long downloading time for videos
  - negative experiences on public transport – felt sound had disturbed other passengers – "*I would'nt watch videos on a bus. What if the person sitting next to you is somebody who's totally exhausted after a day at work, has perhaps been harassed by his boss, and who will only be irritated by the crackling noise? No thanks.*"

Not surprisingly sports are often regarded as an activity that might furnish suitable multimedia content. Like the CONNECT Stadium scenario, Ojala et al [153]'in their study Mobile Karpat..' examine some of the issues involved in providing rich multimedia content to mobile phone users before and during an ice hockey game. While their sample size was small, as is usual with user studies of this the responses obtained were nevertheless interesting. An on-line survey at the hockey club's website obtained a number of interesting ideas concerning how a multimedia service might be used and point to some of the dependability issues that might arise:

- "Question 1: "What would you like to do with the device during the match" – *Look at the goals again of course. During the intermissions I could check the statistics.*"
- Question 2: "What would you like to save from the match to the device and take with you when you leave the arena?- *Goals, screw ups and fights. The smell and taste of hot sausages. ..*
- Question 3: What would you like to send to your friends device?" – *1. Comments from the match. 2. Meeting place at the next intermission. Fans of a competing team would get a summary of the goals of our team and the screwups of their team.*"

Okabe [154] studied the interrelationship between new social trends and new devices and how - although social identity and practice are embedded in and contingent on social situations –electronic media cross boundaries between situations previously held to be distinct. These are what Okabe means by 'technosocial situations', pointing to how mobile phones *'create new kinds of bounded places that merge the infrastructures of geography and technology, as well as technosocial practices that merge technical standards and social norms'.* This is similar in many ways to the argument advanced by Carroll et al [35] concerning technology appropriation (specifically mobile phones) by young people. Their argument is that *"technological artefacts provide a range of possibilities for users who shape, and are shaped by, the artefacts. .. use will be an outcome of various individual and group perceptions and experiences.. young people are adopting a lifestyle rather than a technology perspective: they want technology to add value to their lifestyles, satisfy their social and leisure needs and reinforce their group identity. They assess technology according to their needs rather than as a task oriented artefact .. ".*

Kindberg and coauthors [107] provide a taxonomy of reasons for captured images along two dimensions – affective vs. functional reasons and social vs. individual intentions. Affective reasons are about mutual experience – to enrich a shared experience either immediately or later. In this view taking photos with the camera phone is about the taking of photos as part of the social experience. Images taken were shared on the phone itself and were used to share aspects of experience or everyday life with absent friends or family – as a form of 'telepresence'. Like Okabe, Kindberg et al also note how the collection and review of images also formed part of a process of personal reflection and reminiscing – *"carried to keep some treasured person or object close".*

Finally, Kurvinen [110] examines one particular aspect and circumstance of the taking and sharing of images that may be more widely relevant. This concerns its use in 'teasing' or in joking relationships. The taking and sharing of images in this instance is not about maintaining any particular functional relationship or activity but is heavily implicated in everyday social relationships. Kurvinen liken his study and his argument to Taylor and Harper's work on SMS [188] and 'gift-giving and the findings of the Maypole project where pictures were used as messages to express affection and increase or maintain group cohesion.

This points to thinking about the possible use and impact of new applications or devices – the need to think widely, to think seriously about the content, the application, the device, the user, and the social circumstances of use and the various interactions between them. Thinking seriously about users and the 'spaces' and 'places' devices get used requires paying attention to a range of social interactions and issues of space, place – and 'non-place' – where, according to these studies use appears particularly prevalent. Thinking seriously about the mobile phone, for example, needs to recognise that it is not just a phone, not just a communicating device but, as these studies clearly show, a display device, a device for self-representation and presentation and, importantly, a computing device – incorporating scheduling, diary, lists of interests etc that can communicate with other devices.

3.7 Discussion and Future Directions

This chapter of the deliverable has been concerned with the idea of developing 'soft metrics' for evaluating the dependability of applications developed in CONNECT. There has been a particular emphasis on 'trust' because it is a simple version of what a 'dependable' application is; it clearly links to all the other attributes of a dependable system such as timeliness, reliability, responsibility etc. A dependable technology is trustable and while this idea certainly merits a little unpacking, trustable in what circumstances, to do what and so on, it also, importantly resonates with everyday understandings, with our users' understandings, of what a dependable technology should be. In simple terms the emphasis on 'soft metrics' comes from a concern with developing measures of dependability that are relevant to and understandable by CONNECT likely user groups.

'Soft metrics' are social metrics because it is a range of human factors, human notions of dependability, rather than (mere) technical features that determine whether or not (and how) any application gets used. In other words dependability is highly contextual and technical notions of dependability such as times to refresh, time to reboot, meantime to failure and so on can sometime count little with the people that actually use the application. For example, looking at the CONNECT stadium scenario where spectators and others can grab pictures of the match off some central facility as described in Deliverable D6.1 [54]- 'dependability' and use with regard to 'picture quality' will very much be influenced by contextual factors i.e. for the stadium security initially even vague images of which section of the crowd is proving troublesome would be useful in directing personnel to the scene of trouble, but later on if the images are relied on in court for prosecution much higher quality images would be required. Similarly for an average spectator even fuzzy images of his side scoring or an opponent being sent off would be enough to show his mates in the ground or in the pub later, but a journalist might require better quality images that show, for example, the number on a player's shirt, the facial expression as they score.

Accordingly, our argument is that once an application has been developed it needs to be tested in the real world, through tests that elicit and measure the kinds of human factors that we think, and research shows, are important in determining dependability. Such tests need to be done 'in the wild' and not just in the lab because it is in the wild that real uses and real problems are uncovered. Hence we need approaches and techniques that enable us to get at the user experience and provide the kind of soft metrics we have been talking about. Such techniques will include interviews, focus groups, blogs, experience diaries and cultural probes. from these we hope to derive a mass of design relevant information that will aide the redesign of the applications and contribute to their 'dependability'. Therefore, once the CONNECT application scenarios have been developed and deployed in real settings using the CONNECT software prototypes we plan to use the prior mentioned techniques to measure the soft metrics of CONNECT systems.

4 Off-line Dependability Analysis

Verification and validation (V&V) techniques are sought in CONNECT to ensure that Networked Systems as well as the generated bridging CONNECTors satisfy specified levels of accomplishment for dependability requirements, according to pertinent dependability metrics. Both off-line and on-line approaches to V&V and to fault forecasting are pursued, to cover a wider range of needs from the point of view of dependability assurance. As commonly intended in the literature, off-line analysis refers to activities devoted to analyze the system at hand before its deployment, or after deployment but in isolation with respect to the system in operation. On the contrary, on-line analysis refers to activities performed while the system is in operation, so accounting for the detailed system and environment aspects during that specific system execution. We adopt the off-line and on-line terminology with this meaning.

Activities on off-line dependability analysis during the first year are reported in this Deliverable and in Deliverable D2.1 [53]. On-line dependability analysis is addressed in Work Package 2 (with reference to verification and quantitative compositional reasoning of CONNECTors, with first results already presented in Deliverable D2.1) and in Work Package 5, where it will be investigated in future, also based on feedbacks from the monitoring Enabler (see Chapter 7).

4.1 Basic Concepts

Off-line approaches are very suited to early detect design errors and deficiencies, that could otherwise be very costly or even catastrophic when discovered at later stages. Methods for off-line fault-forecasting and dependability assessment considered in CONNECT belong to those for quantitative, or probabilistic, evaluation, that is aiming at evaluating in terms of probabilities the extent to which the attributes of interest are satisfied. As indicated in [15], the two main approaches to derive probabilistic estimates are modeling and (evaluation) testing. Since evaluation testing assumes to run a test suite on the system under test, which in the CONNECT vision is not a priori available, for off-line analysis we focus on modeling, as indicated in the Description of Work of the project.

Modeling is composed of two phases:

- The construction of a model of the system from the elementary stochastic processes that model the behavior of the components of the system and their interactions. These elementary stochastic processes relate to failures, to repair, service restoration and possibly to system duty cycle or phases of activity;
- Processing the model to obtain the expressions and the values of the dependability measures of the system.

Research in dependability analysis has developed a variety of models, each one focusing on particular levels of abstraction and/or system characteristics. As reported in [149], important classes of model representation include: *i)* Combinatorial Methods (such as Reliability Block Diagrams; *ii)* Model Checking; and *iii)* State-Based Stochastic Methods. We are not much interested in Combinatorial Methods, which are simpler approaches and do not easily capture certain features, such as stochastic dependence and imperfect fault coverage. Therefore, in the context of CONNECT we consider as off-line evaluation techniques:

- Stochastic Model Checking, which is a formal verification technique for the analysis of stochastic systems. It is based on the construction of a probabilistic model from a precise, high-level description of a system's behaviour.
- State-Based Stochastic Methods, which use state-space mathematic models expressed with probabilistic assumptions about time durations and transition behaviors. They allow explicit modeling of complex relationships (e.g., concerning failure and repair processes), and their transition structure encodes important sequencing information.

These two approaches will be exploited in CONNECT to enrich the variety of dependability analyses. In fact, the different formalisms and tools implied by the two methods allow: *i)* on the one side, to complement

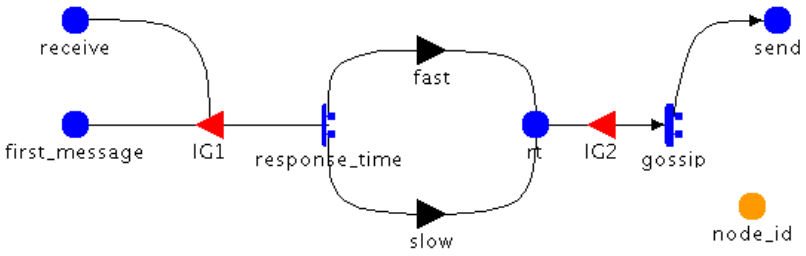


Figure 4.1: SAN model of the logic layer of the gossip protocol

the analysis from the point of view of a number of aspects, such as level of abstraction/scalability/accuracy, for which the two approaches may show different abilities to cope with; and *ii)* on the other hand, through the inner diversity, provide cross-validation to enhance confidence in the correctness of the analysis itself.

In the following we briefly review the two techniques, discuss their usefulness in the CONNECT environment and present some initial work in progress in CONNECT.

4.2 State-Based Stochastic Methods

State-based stochastic models can be classified in Markovian and non-Markovian according to the underlying stochastic process [44, 93, 191]. A wide range of dependability modeling problems fall in the domain of Markovian models, for example when only exponentially distributed times occur. Markov chains (DTMC and CTMC) [97, 144, 93, 191], Stochastic Petri nets (SPN) [141, 43, 21] and Generalized Stochastic Petri nets (GSPN) [8] are among the major Markovian models. However, there is also a great number of real circumstances in which the Markov property is not valid, for example when deterministic times occur; non-Markovian models are used for this type of problems. In past years, several classes of non-Markovian approaches have been defined [29], such as Semi-Markov Stochastic Petri Net (SMSPN's) [44], Markov Regenerative Stochastic Petri Nets (MRSPN's) [42] and Deterministic and Stochastic Petri Nets (DSPN's) [9]. Some major methods for analytically solving the non-Markovian models are discussed in [28, 144, 84].

A short survey on State-Based Stochastic Methods and automated supporting tools for the assisted construction and solution of dependability models can be found in [31]. The Stochastic Activity Networks (SAN) formalism is one of the most powerful (in term of modelling capabilities) stochastic extensions to Petri nets and is supported by the Möbius tool. SAN formalism and the Möbius tool are very commonly used in dependability analysis and therefore they have been initially chosen for dependability analysis in CONNECT.

4.2.1 Overview

In the following, we provide some background on SAN and Möbius to get the reader more familiar with (part of the) dependability models we are going to define in the project.

Stochastic Activity Networks (SANs)

Stochastic Activity Networks (SANs) were first introduced in [143] and then formally defined in [176]. The formalism is a generalization of Stochastic Petri Nets, and has some similarities to the GSPN formalism. The building blocks composing a SAN are: *places*, *activities*, *arcs*, *input gates* and *output gates*.

To graphically illustrate the major SANs symbols, Figure 4.1 represents the SAN model of the logic layer of the gossip protocol we analyzed as case study (see Subsection 4.2.3), when considering heterogeneous aspects of nodes.

Places in SANs have the same interpretation as in Petri Nets: they hold tokens, the number of tokens in a place is called the marking of that place, and the marking of the SAN is the vector containing the marking of all the places. There are two types of activities, timed and instantaneous (thick and thin blue bars in Figure 4.1, respectively). Timed activities are used to represent delays in the system that affect

the measure of interest, while instantaneous activities are used to abstract delays deemed insignificant with respect to the measures of interest. Uncertainty about the length of the delay represented by a timed activity is described by a continuous probability distribution function, called the *activity time distribution function*, that can be a generally distributed random variables, and each distribution can depend on the marking of the network. Activities can have cases (indicated with small circles on activities; in the Figure, both activities have two cases). Cases are used to represent uncertainty about the action taken upon completion of an activity.

Gates connect activities and places. Input gates (red/grey triangles in Figure 4.1) are connected to one or more places and one single activity. They have a predicate, a boolean function of the markings of the connected places, and an output function. When the predicate is true, the gate holds. Output gates (black triangles in Figure 4.1) are connected to one or more places, and the output side of an activity. If the activity has more than one case, output gates are connected to a single case. Output gates have only an output function. Gate functions (both for input and output gates) provide flexibility in defining how the markings of connected places change when the delay represented by an activity expires. Arcs in SANs are default gates, defined to duplicate the behaviour of arcs in Petri nets. Thus, arcs are directed. Each arc connects a single place and a single activity. The arc is an input arc if it is drawn from a place to an activity. An output arc is drawn from an activity to a place. An input arc holds if there is at least one token in the connected place. The function of an input arc removes a token from the connected place, while the function of an output arc adds a token to the connected place. An activity is thus enabled only when i) all of its input gates hold, and ii) all of its input arcs hold.

In the following we give some further details on two particularly important aspects: the execution of a timed activity, and the measure specifications.

- *Completion rules*. When an activity becomes enabled, it is activated, and the time between activation and the scheduled completion of an activity, called the activity time, is sampled from the activity time distribution. Upon completion of an activity, the following events take place: i) if the activity has cases, a case is (probabilistically) chosen; ii) the functions of all the connected input gates are executed; iii) tokens are removed from places connected by input arcs; iv) the functions of all the output gates connected to the chosen case are executed; v) tokens are added to places that are connected by output arcs to the chosen case. An activity is aborted when the SAN moves into a new stable marking in which at least one input gate no longer holds.

- *Measures definition*. Upon completing the model of the system, a modeller has to specify the measures in terms of the model. In the SAN modelling framework, the measures are specified in terms of reward variables [175]. Let $R(m)$ be the rate at which the reward accumulated in state m, and let $C(a)$ be the reward earned upon completion of transition a. If $\{X_t, t > 0\}$ is the modelled stochastic process and M the set of all possible states, a reward variable collected at an instant of time conventionally denoted by V_t is informally defined as

$$V_t = \Sigma_m R(m) P(X_t = m) + \Sigma_a C(a) I_t^a$$

Where I_t^a is the indicator of the event that a was the activity that completed to bring the SAN into the marking observed at time t. The steady-state reward can be obtained as $V_{t\to\infty}$. In the case in which the reward variable is evaluated considering an interval of time $[t, t+l]$, then the accumulated reward is related both to the number of times each activity completes and to the time spent in a particular marking during the interval. More precisely,

$$Y_{[t,t+l]} = \Sigma_m R(m) J^m_{[t,t+l]} + \Sigma_a C(a) N^a_{[t,t+l]}$$

Where $J^m_{[t,t+l]}$ is a random variable representing the total time that the SAN is in the marking m

during $[t, t+l]$ and $N^a_{[t,t+l]}$ is a random variable representing the number of completions of activity a during $[t, t+l]$.

Möbius

Möbius [46] provides an infrastructure to support multiple interacting modeling formalisms and solvers. Formalisms supported include: SAN's, Buckets and Balls (a generalization of Markov chains), and PEPA (a process algebra). The main features of the tool include:

- *Multiple modeling languages*, based on either graphical or textual representations. Supported model types include stochastic extensions to Petri nets, Markov chains and extensions, and stochastic process algebras. Models are constructed with the right level of detail, and customized to the specific behavior of the system of interest.
- *Hierarchical modeling paradigm*. Models are built from the ground up. First specify the behavior of individual components, and then combine the components to create a model of the complete system. It is easy to combine components in multiple ways to examine alternative system designs.
- *Customized measures of system properties*, with ability to construct detailed expressions that measure the exact information desired about the system (e.g., reliability, availability, performance, and security). Measurements can be conducted at specific time points, over periods of time, or when the system reaches steady state.
- *Study the behavior of the system under a variety of operating conditions*. The functionality of the system can be defined as model input parameters, and then the behavior of the system can be automatically studied across wide ranges of input parameter values to determine safe operating ranges, to determine important system constraints, and to study system behaviors that could be difficult to measure experimentally with prototypes.
- *Distributed discrete-event simulation*. the tool evaluates the custom measures using efficient simulation algorithms to repeatedly execute the system, either on the local machine or in a distributed fashion across a cluster of machines, and gather statistical results of the measures.
- *Numerical solution techniques*. Exact solutions can be calculated for many classes of models, and advances in state-space computation and generation techniques make it possible to solve models with tens of millions of states. Previously, such models could be solved only by simulation.

Möbius allows to combine (atomic) models to form the *Composed model*. To this purpose, it supports the two operators Rep and $Join$ to compose sub-networks. It supports the transient and steady-state analysis of Markovian models, the steady-state analysis of non-Markovian DSPN-like models [179], and transient and steady-state simulation. More information can be found in the web site:
http://www.crhc.uiuc.edu/PERFORM.

4.2.2 Utility of state-based stochastic modeling in CONNECT and related work

With reference to the four actors (see Figure 1.1) composing the CONNECT architecture defined in Deliverable D1.1 [52], some high level indications on benefits from dependability analysis are given in the following.

- CONNECT *Enablers*. The analysis could help in guiding the process towards the on-line generation of a CONNECTor with the desired dependability accomplishment level.
- CONNECT*ors* synthesised by CONNECT Enablers. The analysis would allow to assess whether the emergent CONNECTor satisfies the dependability requirements. Looking at possible reusability of already generated CONNECTors in subsequent interactions between the same or similar interacting Networked Systems, the provided assessment could be used as a further criterion for the optimal selection of the CONNECTor to deploy to satisfy specific interaction needs.

- *Networked systems* that use CONNECT services. In this case, information on dependability properties of the Networked System could be directly exposed by the system itself or be assessed through the dependability analysis performed inside the CONNECT framework.
- CONNECT*ed Systems* that are the outcome of successful creation and deployment of CONNECTors. To quantify metrics for end-to-end dependability, in order to verify whether desired levels for them are satisfied, is one of the major goals of CONNECT.

Off-line state-based stochastic models could be profitably employed in CONNECT to perform dependability analysis of all the four actors above.

A huge amount of studies on dependability analyses using state-based stochastic modeling is documented in the literature, targeting disparate application fields. Sessions on modeling and evaluations appear regularly in the technical programs of prime conferences on dependability and resilience, such as IEEE DSN, IEEE SRDS, IEEE ISSRE, IEEE HASE, EDCC. Archival journals, such as IEEE TR, IEEE TDSC, Reliability Engineering & System Safety (Elsevier), Performance Evaluation publish numerous articles on these topics. The group at CNR-ISTI has long dated experience in applying state-based stochastic modeling for dependability analysis , both as a method for early evaluation of under-development designs (e.g., [31, 122, 123]) and for assessment of already developed systems/protocols/infrastructures (e.g., [41, 172, 162, 72])

In order to be applied for the analysis of an actor, this approach requires a specification of the behavior and structure of the actor under analysis, as well as relevant information on failure assumptions and metrics of interest. Of course, the more accurate and complete is the specification, the more accurate is the dependability model and related evaluation. During this first year of the project, the definition of both CONNECT Enablers and synthesized CONNECTors has undergone deep investigation. Therefore, during this first year the activity on dependability analysis could not address CONNECT Enablers and synthesized CONNECTors, which were still under definition. Therefore, to allow for parallel progress of research on both CONNECT architecture and dependability evaluation, the pragmatic choice we took was to address to some extent the analysis of Networked Systems. The focus has been on diffusion protocols in wireless environment, specifically on *gossip*. Such kind of protocols is expected to be of interest in CONNECT as they may be considered as basic building block for many distributed services. In fact, gossip-based protocols have been applied in a wide range of situations such as data dissemination, overlay maintenance, and, more recently, also in peer-to-peer streaming applications. In a gossip, each node periodically exchanges messages with randomly selected nodes in order to disseminate data, exchange membership information, or build distributed state (http://gossiplib.gforge.inria.fr/).

To understand dependability aspects of dissemination protocols at middleware and application level, failure modes and performance indicators coming from lower layers need to be known. In the literature, gossip protocols have been typically studied in presence of homogeneous characteristics of the wireless network on which they are deployed. Instead, the many dimensions of heterogeneity now arising in networked environments and further called to increase dramatically in the future are fundamental drivers of the CONNECT research. Therefore, to improve on the existing analysis and make it more appropriate to be utilized in CONNECT, we investigated effects of heterogeneity at level of node communication and computation. The results are useful to properly set up failure characteristics of gossip-based middleware and application services.

Moreover, the gossip protocol we considered has been already analysed in the literature through the stochastic model checker PRISM. This brings the advantage that we can compare the assessment performed through the two approaches to model-based analyses undertaken in CONNECT. By comparing the results obtained through the SAN model (and Möbius as solution tool) with those obtained with PRISM, a cross-validation of the models built through the two approaches is performed, reinforcing confidence in the correctness of both of them.

4.2.3 Activities in progress: modeling a gossip protocol in wireless networks

We report here the analysis we performed on the impact of heterogeneous communication and computation capabilities of nodes on aspects related to dependability [15] of the gossip protocol. Regarding computation heterogeneity, we considered scenarios in which nodes respond to the protocol with different promptness level. This may happen, for instance, in the case of hardware heterogeneity (e.g., nodes may

have different processing speed), software heterogeneity (e.g., nodes may execute applications that keep communication and/or computation resources busy with different patterns), and protocol heterogeneity (e.g., nodes may use adaptation layers to be able to connect to new services, as envisioned in the CONNECT project). Communication heterogeneity, on the other hand, considers scenarios where nodes have different communication ranges. In real-world urban scenarios this is already a frequent situation, since the environment contains obstacles that may reflect and attenuate the radio signal of some nodes. Furthermore, this may happen also when nodes use power saving strategies that setup different transmission power depending on the depletion level of their batteries [32]. Major contributions of our work are *i)* the development of a modelling framework able to capture the behaviour of diffusion protocols under such aspects of heterogeneity, and *ii)* the quantitative assessment of dependability and performance indicators helpful in guiding the setup of wireless networks. Actually, the goal of the analysis is many-fold: *i)* to validate the developed model for the diffusion protocol by performing the analysis in the same conditions as in other previous studies and comparing the obtained results; *ii)* to explore the impact of newly included heterogeneous aspects on the selected dependability and performance indicators; *iii)* to assess the precision of the results obtained via simulations with respect to analytical solutions; *iv)* to explore the scalability of the proposed model by analysing larger networks.

In the following, we briefly sketch the SAN models developed for the gossip protocol and the supporting communication layers and present selected figures representing the results of the performed analyses. The details of this study are fully documented in [137].

The gossip protocol for wireless networks is specified as follows: whenever a node receives a message to be delivered to all nodes for the first time, the message gets forwarded with probability p_{tx} [89]. Despite its simple specification, the gossip protocol shows complex behaviour when executed in the network. Indeed, an interesting aspect is what happens for different values of p_{tx}. If p_{tx} is lowered down, the utilisation of the wireless channel is reduced since the overall number of transmissions is decreased. It has been shown that a critical value exists, such that if p_{tx} is above the critical value then the percentage of nodes reached with the protocol is barely reduced. If p_{tx} is below the critical value, the protocol dies out quickly, and the message is delivered only to a small percentage of nodes in the network. This phenomenon is known as *phase transition*. The critical value which triggers the phase transition represents the optimal configuration of the protocol, since it gives the best trade-off between performance and reliability. This value is not easy to obtain, since it depends on many characteristics of the wireless network, such as its density and diameter. Relevant aspects of the wireless communication that have been accounted for in the developed model include the communication range, that is the maximum distance that allows direct communication between nodes, which may differ from a node to another and the capture effect, that is the phenomenon tight to concurrent transmissions by neighboring nodes which may cause collisions between messages.

The model of the wireless network has been developed with the SAN [174] formalism, and is split into a number of sub-models that interact through shared variables. Each sub-model identifies a different logic layer of the communication stack: *network layer* (diffusion protocol), and the underlying *MAC*, *Link*, and *Physical* layers. The composition of the layers describes the behaviour of a single node, while the behaviour of the entire wireless network is obtained by replicating N times the model of a node, where N is the number of nodes in the network. The overall model of the system is shown in Figure 4.2: sub-models (atomic models) are represented with labelled dark boxes, the composition between sub-models is obtained through the *Join* operator, and the *Rep* operator creates N replicas of the composed model. The *Join* operator allows interaction between sub-models through shared places that have the same name. Similarly, the *Rep* operator allows interaction between replicas through shared places; in this case, the name of the shared place is given as parameter to the operator.

- *Source Logic* and *Gossip Logic*. These two atomic models represent the node's behavior in the gossip protocol when the node acts as source node, generating the messages to be diffused in the network, and as a relay node, which actually applies the diffusion protocol, respectively.
- *MAC Layer*. This atomic model represents the actions performed by the node to access the channel for sending a message (check on channel availability, management of collisions and re-transmissions).
- *Link Layer (transmitter)* and *Link Layer (receiver)*. Two atomic models have been defined to specify the behavior of the two end-points of the wireless link.

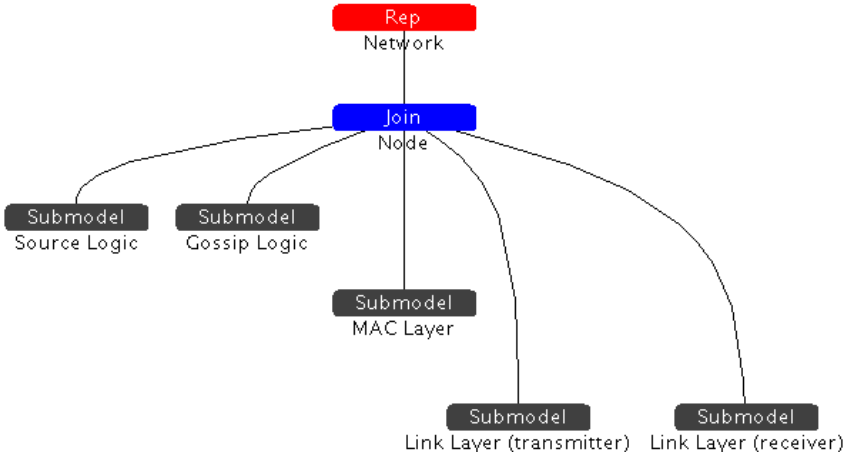


Figure 4.2: Model of the Wireless Network

This modeling effort has been developed to assess two key dependability-related metrics of diffusion protocols: *network coverage* and *end-to-end delay*. Network coverage is a reliability metric of the protocol defined as the percentage of nodes reached by the message. End-to-end delay is a performance metric that gives a bound to the time needed to reach a certain percentage of nodes. This metric is specially relevant for real-time applications that require hard / soft deadlines.

4.2.4 Analysis: some results

As already said, the performed analysis has had several objectives: *i)* to validate the developed model for the diffusion protocol by comparing results with those from a previous study; *ii)* to explore the impact of newly included heterogeneous aspects on the selected dependability and performance indicators; *iii)* to assess the precision of the results obtained via simulations wrt analytical solutions; *iv)* to explore the scalability of the proposed model by analysing larger networks.

The analysis for model validation purposes has considered the previous study in [73] where the analysis of the gossip protocol was performed with the probabilistic model-checker PRISM [95]. The specification of the protocol in [73] is detailed enough to derive all the assumptions introduced and to reproduce them in our model. Of course there are differences between the two models, essentially because the previous study had different objectives than ours and didn't introduce elements of heterogeneity and some details in the communication network; however, we managed to make them not relevant for the sake of the comparison (by assigning proper values to the pertinent model parameters). Although different formalisms are used and different solution techniques are adopted (simulation in our case, model checking in [73]), we found that our results are almost coincident with those shown in [73]. This cross-validation exercise reinforced our confidence on the correct definition of the model.

Concerning the impact of node heterogeneity on the metrics of interest, we evaluated separately the impact of communication heterogeneity (accounted for by considering that nodes in the network have different forwarding probabilities p_{tx}, each of which depends on the number of neighbors) and computation heterogeneity (accounted for by considering that nodes have different response times depending on their speed). A testbed, composed of 48 wireless sensor network, deployed at the UVA Computer Science Department of the University of Virginia and already used to study optimisations of MAC protocols, has been chosen as wireless network topology. Table 4.1 reports the main parameters considered in the model and their values for the different studies. Some parameters are made varying in a range of values during our studies, since we are interested in assessing the sensitivity of the measure of interest with respect to such parameters.

A first analysis concentrated on understanding whether network coverage can be improved by exploiting communication heterogeneity. To this purpose, we assessed the impact of different communication ranges by setting-up different probabilities p_{tx} for each node i according to the number n_i of neighbours

Parameter	Value	Description
N	12, 48, 200, 500	network size
p_{tx}	from 0.1 to 1.0	forwarding probability of gossip
p_{rxS}	0.92	probability of receiving a message on stable links
p_{rxI}	0.5	probability of receiving a message on intermittent links
$p_{capture}$	0.1	probability of receiving a message in the case of collision
p_{fast}*	from 0.1 to 1	probability of having a fast node in the network
tx_{rate}	1	transmission rate
std_{rate}	1	processing speed of a standard node
$fast_{rate}$	1000	processing speed of a fast node
$retry\_limit$	5	maximum number of re-transmissions performed by the MAC layer

*this parameter is set to 0 when not relevant in the study.

Table 4.1: Parameter setup

of the node. Figure 4.3 shows the results of coverage (in percentage) when nodes have different p_{tx}. In the figure, the curve relative to $k = 0$ corresponds to the coverage obtained when all nodes have the same p_{tx}, while curves for positive k correspond to the coverage in case of a direct proportionality ($k = 1$), either linear or quadratic relation ($k = 2$), between p_{tx_i} and n_i. Similarly, a negative k corresponds to an inverse proportionality, either linear or quadratic, between p_{tx_i} and n_i. For the chosen parameter setting, it can be appreciated the impact of heterogeneous p_{tx} on coverage, and it can be noticed that there is an improvement in coverage for direct relations. Considerations on this result are in [137].

Another study focused on computation heterogeneity by assessing the impact of different computation speeds by setting-up the promptness of nodes in taking a decision on gossiping (fast nodes and standard nodes are considered). The percentage of fast nodes that populate the network (p_{fast}) is a parameter of the study. Figure 4.4 shows the coverage for different time steps t (the actual value of t can be expressed in seconds once bitrate and message size are given). In the figure, we report the percentage of covered nodes for two cases: 10% of fast nodes ($p_{fast} = 0.1$) and 90% of fast nodes ($p_{fast} = 0.9$). The value on the x axis represents the end-to-end delay to reach a corresponding percentage of coverage (on the y axis). Not surprisingly, the configuration with a higher percentage of fast nodes performs better that the other until a certain value of t, after which the curves overlap. This kind of analysis is useful to give a quantitative assessment of different network configurations, and is specifically useful to properly setup the network in presence of deadlines imposed by real-time applications.

An effort has been also devoted to assess the accuracy of the results obtained through simulation, by comparing simulative and analytically obtained results, and to show the scalability of the proposed model. For the former, we considered a network of 12 nodes (smaller than in the previous studies, but still significant enough to appreciate the relevance of differences between the results obtained analytically and by simulations) and we found that the relative differences between analytical and simulative results for the study of the coverage at varying p_{tx} are very low. This points out that simulative results have high accuracy for these kind of studies. For the latter, larger networks (of 200 and 500 nodes) have been considered and successfully analyzed via simulation.

4.2.5 Final considerations

This work has addressed dependability-related analysis of diffusion protocols under heterogeneous aspects characterising the supporting wireless network. The emphasis was essentially on elements of heterogeneity and on the capability to deal with large-scale Networked Systems, to better fit within the CONNECT interest. The obtained results, although limited to the considered parameters setting, allow to understand some relevant dynamics, useful to *i)* properly setup the system, such as how to choose nodes with respect to communication levels in order to better trade-off coverage with traffic channels, and *ii)* understand dependability behavior of the underlying wireless network, such as reliability indicators, to be imported as basic failure parameters in the higher (middleware) level dependability model of gossip-based communication protocols, thus facilitating their modeling and evaluation by exploiting better

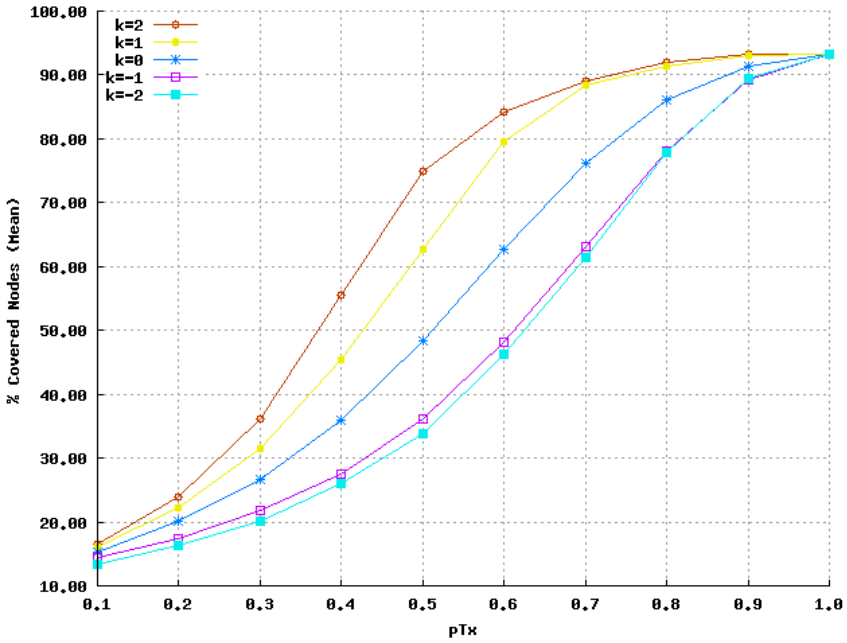


Figure 4.3: Percentage of covered nodes with different p_{tx} ($N = 48$)

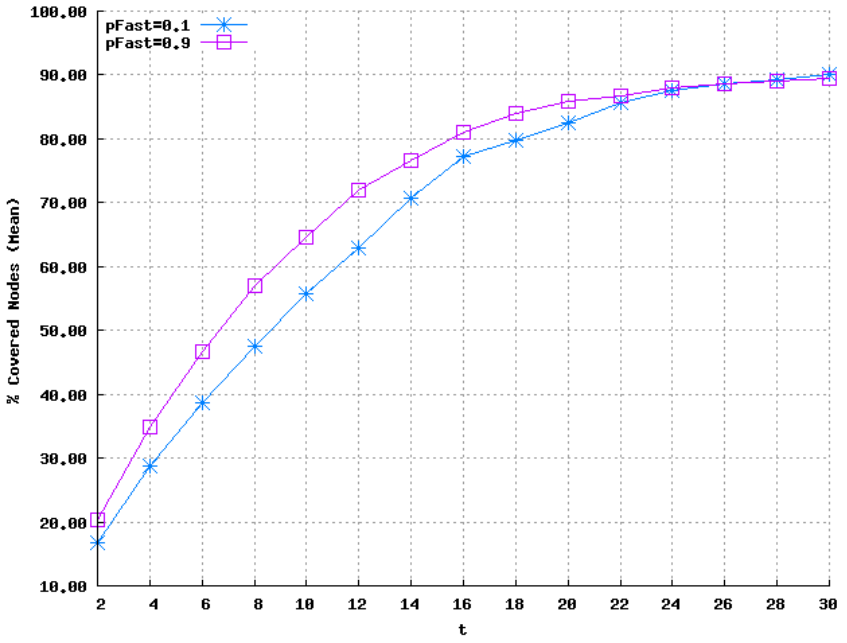


Figure 4.4: End-to-end delay vs. covered nodes for different percentage of fast nodes ($N = 48$)

abstraction facilities.

The analyses carried out have paved the way to the understanding of the phenomena related to heterogeneity in wireless networks, and several extensions to this study can be envisaged. For example, understanding the interplay between the different communication and computation capabilities of the nodes on selected measures of interest would improve finding an optimal network setup. Extending the evaluation campaign by enriching the set of measures of interest for the analyses would been another follow-on. Also, introducing other patterns of Networked Systems failures, other than the channels failures considered so far, would be highly interesting.

4.3 Stochastic Model Checking

Stochastic model checking (also known as probabilistic model checking) is a formal verification technique for the analysis of stochastic systems. It is based on the construction of a probabilistic model from a precise, high-level description of a system's behaviour. A quantitative analysis of this model is then performed, by applying a combination of exhaustive search techniques and numerical solution methods. Similarly to state-based stochastic methods, stochastic model checking can be applied to all four types of actors mentioned in Section 4.2.2, i.e., CONNECT Enablers, CONNECTors, Networked systems and CONNECTed Systems, depending on the level of abstraction.

4.3.1 Overview

In general, stochastic model checking deals with Markovian models because their memoryless property greatly reduces the computation complexity of model checking and therefore, allows stochastic model checking to be applied to real-world systems. As mentioned in Section 4.2, DTMCs and CTMCs are widely used Markovian models. In addition, MDPs (Markov Decision Processes) are another commonly adopted model, which is an extension of DTMCs with non-determinism. The semantics and model checking procedures for MDP/DTMC/CTMC can be found in Deliverable 2.1 [53]. In the literature, a number of probabilistic model checkers have been built. A quick overview is given as follows.

Model checkers for MDPs. LiQuor [45] is an explicit-state model checker for MDPs, expressed in Probmela (a probabilistic extension of SPIN's Promela language), against LTL. RAPTURE [102] implemented the MDP-based abstraction-refinement technique described in [60]. ProbDiVinE [23] supports parallel and/or distributed LTL model checking of MDPs.

Model checkers for DTMCs/CTMCs. MRMC [105] (formerly ETMCC) performs explicit-state (and approximate) model checking for DTMCs, CTMCs and CTMDPs (with rewards) against PC(R)TL and CS(R)L. It also supports bisimulation. The PEPA Eclipse Plug-in project [190] supports CSL model checking (plus steady-state/ODE analysis and abstraction techniques) for the stochastic process algebra PEPA. CASPA [171] is a symbolic (MTBDD-based) model checker for (extended) stochastic labelled transition systems against Stochastic Propositional Dynamic Logic (SPDL).

Approximate probabilistic model checkers. APMC [88] is a distributed approximate model checker for DTMCs and CTMCs. Ymer [196] (formerly ProVer) does approximate probabilistic model checking of generalized semi-Markov processes (GSMPs) and CTMCs against transient properties expressed in CSL.

PRISM [95, 114] is a symbolic model checker for probabilistic verification of MDP/DTMC/CTMC models. According to the experiments in [101], PRISM is among the fastest probabilistic model checkers for large scale systems. It also supports verification for rewards, which is the key feature for verifying non-functional requirements. PRISM has an easy to use GUI and supports rich functionality, such as simulation, graphical plot output and experiments, in addition to temporal logic model checking. We shall employ PRISM in CONNECT for off-line verification of dependability and performance. We summarise its features in the following section.

User interfaces. All of the functionality of PRISM is available from either a command-line version of the tool or through its graphical user interface. The former is useful for running lengthy or process-intensive jobs, executing large batches of model checking runs or for combining PRISM with other tools through scripting. The graphical user interface (GUI) provides a more intuitive entry-point for new users to the tools, as well as invaluable features for more experienced users. The GUI provides:

- a model editor for the PRISM modelling language, with syntax highlighting and error reporting;
- an editor for PRISM properties;
- a simulator tool for exploring and debugging PRISM models;
- graph-plotting tools.

Experiments. When PRISM performs the analysis of quantitative properties, it is often instructive to generate and plot a range of such values, as a parameter of either the property or model is varied. This can be invaluable for studying trends in quantitative results or identifying anomalous behaviour of the system. Examples are “the instantaneous availability of the system at time t" for a range of time values t or “the expected throughput of the system" for a range of different Networked System failure rates. PRISM is designed to facilitate such queries and includes a simple mechanism, known as experiments, for this purpose. The tool includes an integrated graph plotting tool for visualising and displaying results.

Discrete-event simulation. PRISM also incorporates a discrete-event simulation engine, which has two purposes. Firstly, it forms the basis of a tool for debugging models. This can be used for either manual exploration or creation of random traces. Secondly, it provides generation of approximate solutions to the numerical computations that underlie the model checking process, by applying Monte Carlo methods and sampling. These techniques complement the main model model checking functionality of the tool, offering increased scalability, but at the expense of numerical accuracy.

Model checking algorithms. The basic algorithms required for model checking are those proposed for PCTL [91] and for CSL [20]. Algorithms for the reward operators, which are based on standard techniques for Markov reward models, can be found in [112]. The detailed explanation of these algorithms can also be found in Deliverable 2.1 [53]. The key ingredients of these algorithms can be classified into two groups: *graph-theoretical* algorithms and *numerical computation*. The former class, which operate on the underlying graph structure of a Markov chain, are used, for example, to determine the set of reachable states in a model or to model check qualitative properties. Numerical computation is required for the calculation of probabilities and reward values.

For numerical computation, model checking typically requires either solution of linear equation systems or calculation of the transient probabilities of a Markov chain. Because of the large size of the models that often arise, the tool uses iterative methods rather than direct methods. For solution of linear equation systems, PRISM supports a range of well-known techniques including the Jacobi, Gauss-Seidel and SOR (successive over-relaxation) methods; for transient probability computation, it uses uniformisation. See for example [185] for good coverage of both these topics.

Symbolic implementation. An important aspect of the implementation of PRISM is its use of state-of-the-art *symbolic* techniques, applying data structures based on binary decision diagrams (BDDs). Some aspects of the probabilistic model checking process, such as reachability and graph- theoretical algorithms, can be implemented with well known BDD-based approaches. In general, however, representing and manipulating probabilistic models requires extensions of these techniques. PRISM uses a generalisation of BDDs called *multi-terminal* BDDs (MTBDDs) [48, 18]. These provide compact representation and efficient manipulation of large, structured models by exploiting regularities exhibited in the high-level modelling language descriptions. In fact, for the numerical solution aspect of the probabilistic model checking process, which is typically the most resource-intensive, PRISM provides a choice of three distinct computation *engines*. The first is purely symbolic, using BDDs and MTBDDs only. For models with a large degree of regularity, this option can be extremely efficient, allowing PRISM to scale to the analysis of probabilistic models with as many as 10^{10} states. Often though, especially for analysis of CTMCs, MTBDDs perform poorly in this phase due to irregularity in the numerical values computed. By contrast, the second engine is implemented using explicit data structures: sparse matrices and arrays. This provides more predictable performance and, where usable, is usually the fastest engine. It is, though, more demanding in terms of memory usage, limiting its applicability to large models. The final PRISM engine is called the *hybrid* engine, which uses extensions of MTBDDs [111, 156] to combine advantages of the other two engines. It is generally faster (and more often feasible) than the symbolic engine but uses less memory than sparse matrices and is the default engine in PRISM. This choice of engines provides a flexible implementation which can be adjusted to improve performance depending on the type of models and properties being analysed.

4.3.2 Model checking dependability and performance in PRISM

In order to analyse a PRISM model, it is necessary to specify one or more properties. PRISM's property specification language is based on temporal logic, which offers a flexible and unambiguous means of describing a wide range of properties. In particular, the language incorporates operators from PCTL [91], CSL [20] and some of its extensions [19]. These logics have already been shown able to express a wide range of performance, dependability and performability properties. Other logics, such as CTL and LTL, are currently also being incorporated into the property specification language.

PRISM puts particular emphasis on *quantitative* properties. For example, PCTL (and CSL) allow expression of logical statements such as "the probability of eventual system failure is less than p", denoted $P_{<p}$ [F $fail$]. In PRISM, it is more typical to simply ask "what is the probability of eventual system failure?", expressed as $P_{=?}$ [F $fail$]. The property specification language also allows numerical values such as these to be combined in arithmetic expressions, allowing more complex measures to be expressed.

The key constructs in the PRISM property specification language are the P, S and R operators. The P operator refers to the probability of an event occurring (more precisely, the probability that the observed execution of the model satisfies a given specification). The S operator is used to reason about the steady-state probabilities of the model. Finally, the R operator is used to express properties that relate to rewards (more precisely, the expected value of a random variable, associated with particular reward structure).

The the formal semantics of the PRISM property specification language has been summarised in Deliverable 2.1. In the remainder of this section, we give an overview of the different types of property that are available and examples of the kinds of performability measures that they can be used to express.

Transient and steady-state probabilities. The probability that the system is in a particular state of interest, either at a specific time instant (transient) or in the long-run (steady-state) can be expressed using the P and S operators, respectively. Consider, for example, a service-providing system whose state can be classified as either "operational" or "failed" and assume that oper is a Boolean variable whose value is true if system is operational. Alternatively, $oper$ could be a more complex expression over state variables expressing this fact. The following properties describe the availability of the system:

- $P_{=?}$ [$F^{[t,t]}$ $oper$] - "the instantaneous availability of the system, i.e. the probability that it is operational at time instant t";
- $S_{=?}$ [$oper$] - "the long-run availability of the system, i.e. the steady-state probability that it is operational".

Timing, occurrence and ordering of events. The P operator in the example above is used with the $F^{[t,t]}$ operator to refer to a single instant in time. This is a special case of the time-bounded operator F^{I} where I is a time interval $[t1, t2] \in R$. Also permitted is the *unbounded variant* F, equivalent to $F^{[0,1)}$. These allow us to reason about the probability of an event occurring either within in a particular period of time, or at any point in the lifetime of the system. Consider a system comprising two components, A and B, each of which can fail independently. We can express:

- $P_{=?}$ [$F^{[0,600]}$ $fail_A$] - "the probability that component A fails within 10 minutes";
- $P_{=?}$ [F $fail_A \mid fail_B$] - "the probability that either component A or B fails at some point".

There are also several other operators that can be used with the P operator. One example is G, which can be seen as the dual of F: it expresses the fact that a condition remains, rather than becomes, true. Like F, it has both timed and untimed variants. The former gives, for example:

- $P_{=?}$ [$G^{[0,3600]}$ $!(fail_A \mid fail_B)$] - "the probability of no failures occurring in the first hour".

Another related operator is U which, for two given conditions, states that the first remains true until the second becomes true. Examples of its use include:

- $P_{=?}$ [$!fail_B$ $U^{[3600,7200]}$ $fail_B$] - "the probability of component B failing for the first time during the sec- ond hour of operation";
- $P_{=?}$ [$!fail_A$ U $fail_B$] - "the probability that component B fails before component A".

To be more precise, the latter property gives the probability that component B eventually fails and no failure of component A occurs before this point. An alternative is the weak form of the operator, denoted W, for which the first part of this is not required:

- $P_{=?}$ [$!fail_A$ W $fail_B$] - “the probability that a failure of component B (if it occurs) happens before any failure of component A, i.e. the probability that either B fails before A or neither ever fail".

Reward-based properties. The PRISM property specification language also includes an R operator which is used to refer to the expected value of a random variable defined in terms of a reward structure. As we show below, R, like P, can be combined with a variety of operators. Since a model will often be decorated with multiple reward structures, we augment the R operator with a label: properties of the form $R_{\{"rew"\}=?}$ [...] refer to the *expected value* of reward structure rew.

Consider, as in our first set of examples above, a system which provides a service when it is operational. We assume a reward structure oper that assigns a state reward of 1 to all states of the model in which the system is operational (and 0 to all others). By reasoning about the amount of reward accumulated over a period of time, we can represent:

- $R_{\{"oper"\}=?}$ [C$\leq t$] - “the expected cumulative operational time of the system in the time interval $[0, t]$".

Another possibility is to consider the reward accumulated until a given event occurs. Assume that we have a simple reward structure $time$, which assigns a state reward of 1 to every state, and a Boolean state variable $fail$, which is true when a system failure has occurred. We have:

- $R_{\{"time"\}=?}$ [F $fail$] - “the mean-time-to-failure of the system, i.e. the expected amount of time that elapses before the first failure occurs".

A third operator to reason about cumulative rewards is the S operator, which gives the long-run expected rate of reward accumulation. For example, consider a model of a queue storing jobs to be processed by a server. Assume a reward structure $proc$ that assigns a transition reward of 1 to every transition corresponding to the a job being processed by the server. Then we can use:

- $R_{\{"proc"\}=?}$ [S] - “the throughput of the system, i.e. the expected steady-state rate of job completion".

Finally we consider a fourth operator for R which deals, not with accumulation, but the *instantaneous* value of rewards. In this context, we deal only with state rewards, but these do not need to represent a rate at which rewards accumulate: they can describe any measure of interest that can be defined in terms of the state variables of the model. Consider, as above, a model of a job queue and a reward structure size which assigns a state reward to each state equal to the number of jobs in the queue at that point. We can can then compute:

- $R_{\{"size"\}=?}$ [$I^{=t}$] - “the expected number of jobs waiting in the queue at time t";

Best- and worst-case scenarios. Because model checking is exhaustive and computes exact answers, values are usually generated for all states of a model. For example, when model checking $P_{=?}$ [F $fail$], PRISM computes the probability of reaching a state in which fail is true, starting from any state of the model. The PRISM property specification language includes a feature called filters, which computes the minimum or maximum value of a property over a range of states. Examples of its use are as follows:

- $R_{\{"time"\}=?}$ [F $oper$ $"fail"max$] - “the worst-case mean time required for system repair, from any possible failure state of the system";

- $P_{=?}$ [$F^{[t,t]}$ $oper$ $"init"$max] - “the best-case instantaneous availability of the system at time t, starting from any initial configuration".

Arithmetic expressions. Since the properties we have presented here, using operators of the form $P_{=?}$, $R_{=?}$ and $S_{=?}$, return numerical values, it is natural to also allow these to be built into more complex expressions. This extends further the range of properties expressible in PRISM. Examples include:

- 1 - $P_{=?}$ [$F^{[3600,7200]}$ $oper$] - “the probability that the system is not operational at any point during the second hour of operation".
- $R_{\{"oper"\}=?}$ [$C^{\leq t}$]$/t$ - “the expected interval availability of the system in the time interval $[0, t]$, i.e. the fraction of that time which it is available".
- $P_{=?}$ [F $fail_A$]\ $P_{=?}$ [F any_fail] - “the (conditional) probability that component A eventually fails, given that at least one component fails".

Another use of arithmetic expressions is to compute statistical properties such as variance or standard deviation. Consider the reward structure $size$ used earlier to represent the size of a job queue. If we define a second reward structure $size\_sq$, which assigns the square of this value to each state, we can define:

- $R_{\{"size\_sq"\}=?}$ [$I^{=t}$] - $pow(R_{\{"size"\}=?}$ [$I^{=t}$],2) - “the variance of the jobs queue size at time t".

4.3.3 Case study

We have shown the study of a gossip protocol using state-based stochastic method in Section 4.2.3. The protocol was modelled as an SAN there. Often a gossip protocol exhibits both non-deterministic and probabilistic behaviour. Nondeterminism arises because we consider a distributed network in which the activities of individual nodes occur asynchronously. Other actions, such as the random selection of a node with whom to exchange information, are inherently probabilistic. To demonstrate the application stochastic model checking on dependability analysis, we build an MDP model for a gossip protocol and check some reward properties [113].

We assume a network of N nodes, each with an address that is required for sending a message to it. Each node maintains a partial view of the network: a list of up to c ($< N$) node descriptors, each of which comprises a node's network address and an *age* that represents the freshness of the descriptor. Periodically each node will randomly choose a small number of nodes to exchange the information contained in their views. This allows information about the topology of the network (and changes to it) to be propagated between nodes. We use *hop-counts* as a coarse (bounded) measure of the age of each node descriptor. A receiving node increments the the hop-count of all the incoming descriptors, merges these with the descriptors in its own view (keeping the entry with the youngest count in cases of duplication) and then keeps the c newest entries from the combined set.

In this study, we investigate the expected number of rounds of gossiping required for the nodes to form a connected network and how the expected path length between nodes evolves over the execution of the protocol. The presence of nondeterminism means that these measures can take a range of values. Hence, we compute minimum and maximum values, representing the best- and worst-case performance of the protocol under any scheduling of nodes. We investigate the relationship of these results with average values, as would be obtained through simulation. We also use PRISM to identify the precise situations under which the best- and worst-case behaviour arises.

Unlike the analysis performed on the low level network layer in Section 4.2.3, here we concentrate on how the topology of the network induced by the local views of the nodes varies over time, investigating first the maximum path length (longest route between nodes) and then the time for the network to become connected.

Best, worst and average case behaviour. We first study how the longest path length between nodes in the network varies over the execution of the protocol. This is done using PRISM properties of the form:

$$R_{\{\text{“}path\_len''\}min=?}[I^{=T}]$$

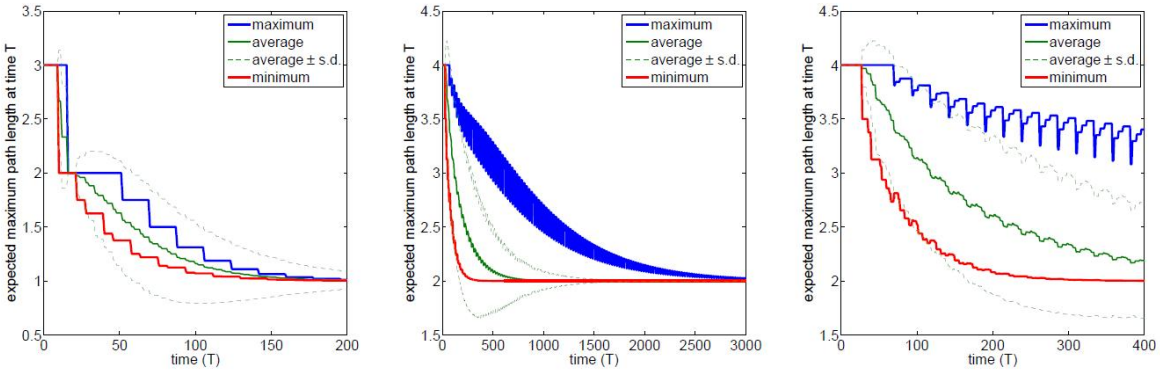
$$R_{\{\text{“}path\_len''\}max=?}[I^{=T}]$$

which represent the minimum/maximum expected value of “$path\_len$” at time instant T. This assumes that we have added a reward structure called “$path\_len$” to the PRISM model, which associates with each state of the MDP a value representing the longest path length between any two nodes at that point (for the case where the graph is not connected, we let “$path\_len$” be N). These properties give the *minimum* and *maximum* values over all possible resolutions of nondeterminism in the MDP which, in this model,

means quantifying over all possible schedulings of the nodes. This allows us to determine the best- and worst-case behaviour of the system in any eventuality. Since the gossip protocol makes random choices, its execution under a particular scheduling is probabilistic. Hence our use of minimum/maximum *expected* values.

It is also possible, with a simple modification of the PRISM model, to compute the *average* value of the longest path length over time. This is done by replacing nondeterminism in the scheduler component of the PRISM model with uniform probabilistic choices, yielding a DTMC instead of an MDP. Although this is no longer an accurate model of the scheduling, it is interesting because the results computed from the DTMC model can be seen as the values that would be obtained through simulation by averaging the results obtained over a large number of simulation runs.

Furthermore, we can also calculate the standard deviation of the random variable corresponding to the longest path length; again this is information that could be obtained (approximately) through simulation. Computing this value (exactly) with PRISM is done by adding a second reward structure, which associates each state of the model with the square of the "$path\_len$" value, and then using the equivalence $\sigma(X)^2 = \mathbf{E}(X^2) - \mathbf{E}(X)^2$.



(a) N=3 (b) N=4 (c) N=4 (zoom)

Figure 4.5: Expected path length: Minimum, maximum and average ($\pm$ standard deviation).

Figures 4.5(a) and 4.5(b) show the full set of these results for $N = 3$ and $N = 4$ nodes, respectively. The thicker solid lines show the minimum and maximum expected longest path length after T time-steps, for a range of values of T. In between these, the thinner solid line shows the average (i.e. expected) value for the same time points. The dashed lines indicate the standard deviation.

The results demonstrate that there is a significant difference between the minimum and maximum values, i.e. between the best- and worst- case behaviour of the protocol. Both values eventually stabilise at 1, for $N = 3$, and 2, for $N = 4$ (in each case, this is the shortest possible longest path length since the local views are of size two).

Note that, despite the discontinuities seen in the graphs, these results are exact and have been computed for every time step. In fact, plots of this kind are typical for systems which operate in rounds, each one requiring multiple discrete time-steps. In the case of the maximum values for $N = 4$ (Figure 4.5(b)) we see that, although the longest path length is converging towards two, there are many small jumps where it increases and then decreases again. This phenomenon can be observed more clearly in Figure 4.5(c), which shows the same plots for a smaller range of time values. This behaviour can be attributed to the fact that, within each round of the protocol, the adversary can schedule nodes in a malicious fashion such that the longest path length temporarily increases. Because of the design of the protocol, though, the expected longest path length decreases as the rounds progress. Figure 4.5(c) also demonstrates that, although they are not as pronounced, the same fluctuations occur for the other plots (average and minimum values).

It is also interesting to observe the relationship between the minimum and maximum values (obtained from the MDP) and the average and standard deviation values (obtained from the DTMC). For the case where $N = 3$, we see that the average values and standard deviation give a slightly pessimistic view:

in fact, the best and worst possible behaviour is within the bounds given by the standard deviation. For $N = 4$, on the other hand, the worst-case (maximum values) are significantly higher. Also, for $N = 3$, the average case falls roughly half-way between the minimum and maximum values, where as, for $N = 4$, it is much closer to the best-case (minimum) behaviour.

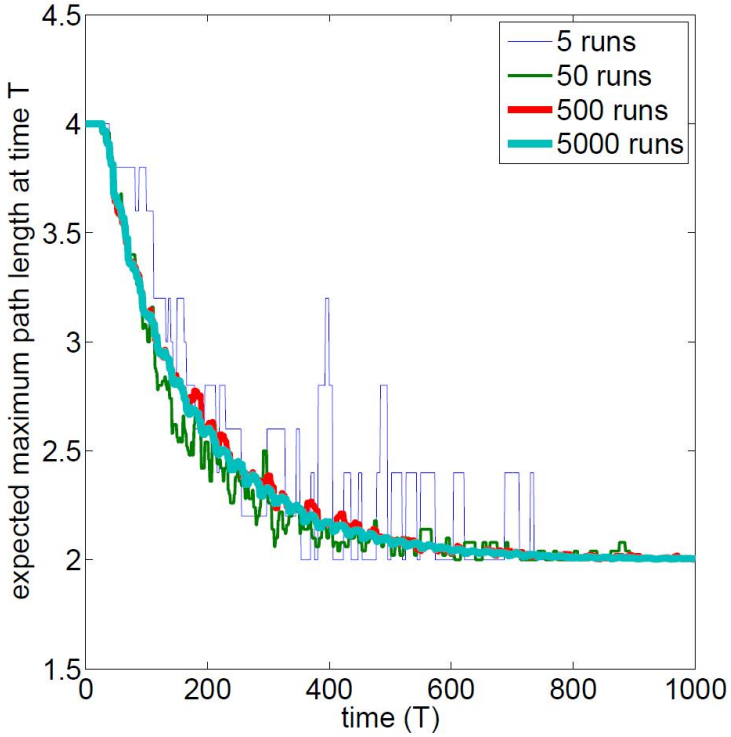


Figure 4.6: Simulation results ($N = 4$).

Lastly, to illustrate the relationship between the above results and those obtained from discrete-event simulation, in Figure 4.6 we have included the average results over 5, 50, 500 and 5,000 simulation runs for the network of 4 nodes. The plots demonstrate that, as we increase the number of simulation runs, the average values converge to the (average) results for the DTMC model given in Figure 4.5(b).

Best and worst case scheduling. One weakness with the property analysed previously is the notion of time used. Each time-step (as measured on the X-axis in the plots) corresponds to a single transition in the model. Because the model comprises a set of processes running in parallel this does not give an accurate measure of elapsed time. Since the gossip protocol proceeds in rounds of fixed time interval, however, we can improve this by considering the number of rounds.

More precisely, we compute the (minimum and maximum) expected number of complete gossiping rounds required before the combined views of the nodes generate a connected network. This is a desirable configuration for the network to reach since, when the local views do not form a connected topology, the nodes have insufficient information to ensure that a message gets propagated to all other nodes in the network. The PRISM properties are:

$$\mathsf{R}_{\{\text{“}num\_rounds\text{”}\}min=?}[\mathsf{F}\text{“}connected\text{”}]$$

$$\mathsf{R}_{\{\text{“}num\_rounds\text{”}\}max=?}[\mathsf{F}\text{“}connected\text{”}]$$

where “num_rounds” is a reward structure that assigns a reward of 1 to transitions marking the end of a round and “$connected$” labels states in which a path exists between any pair of nodes in the network.

As well as computing these measures, we use PRISM to generate actual adversaries that result in the minimum and maximum values. Since the only nondeterminism in the model is due to scheduling of the nodes (i.e. the order in which they forward their views to their neighbours), we can extract from an adversary the corresponding scheduling.

We consider first the network of three nodes. The minimum and maximum number of complete rounds required for the local views to generate a connected network is 0 and 1 respectively. For comparison, in the DTMC model, where nondeterminism has been replaced by uniform random choice, the expected number of rounds is 0.667.

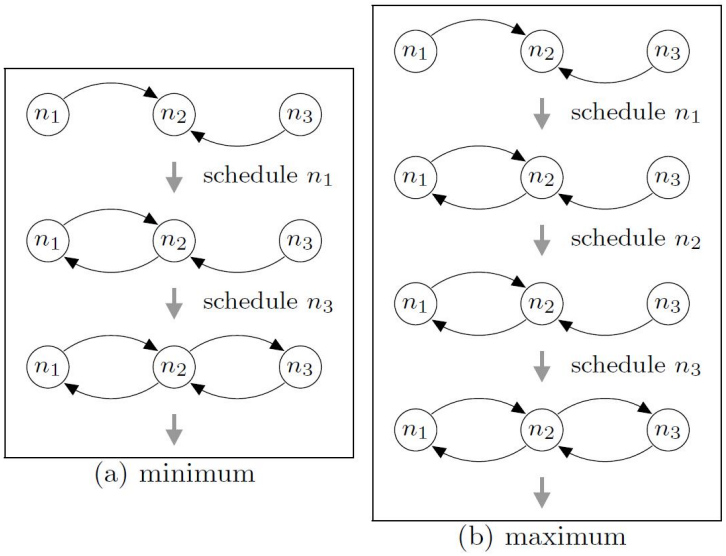


Figure 4.7: Scheduling for the 3 nodes network.

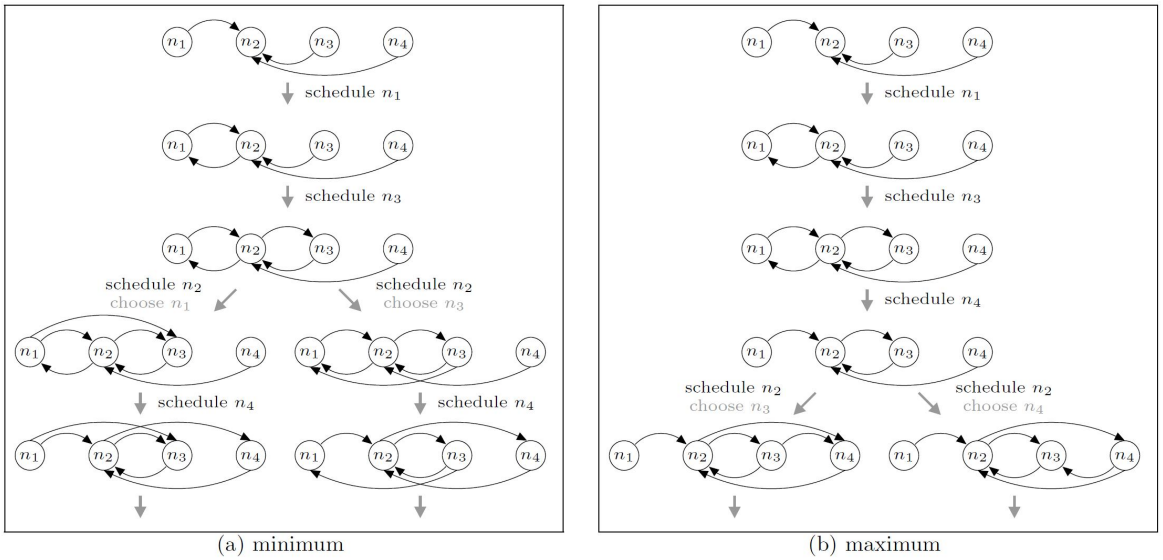
Figure 4.7 illustrates the sequences of node schedulings that result in these minimum and maximum values. Initially, all nodes can see node n_2, but no others. The best-case behaviour (minimum expected number of complete rounds) can be obtained by first scheduling node n1 and then node n_3, after which n_2 has added both n_1 and n_3 to its view and the network is connected. Since n_2 has yet to be scheduled, the minimum expected number of complete rounds is 0. For worst-case behaviour (maximum value), we can schedule n_2 before either n_1 or n_3 is scheduled. This adds no new information to the local views and means that a complete gossiping round is required before the network becomes connected.

For the network consisting of four nodes, we find that the minimum and maximum expected number of complete rounds before connectivity are 1.5 and 4.5 respectively. For comparison, the expected number of rounds for the DTMC model is 2.788 which, unlike in the case of the three node network, is closer to the minimum than the maximum.

This case is more complex than the three node network. Since more than one round may be required before the network becomes connected and the number of possible neighbours exceeds the size of the view, descriptors can be dropped from the views as more recent information becomes available. Furthermore, we must consider a node's choice of who to send data to. Figure 4.8 shows part of the scheduling (the first gossiping round) that can result in the minimum and maximum expected number of complete rounds. In both cases notice that, when node n_4 is scheduled the view of n_2 is updated, causing the removal of n_1 (the oldest descriptor). Not also that, when node n_2 is scheduled it makes a (random) selection between communicating with n_1 or n_3 since both are in its view at the time.

For the minimum case (Figure 4.8(a)) we see that, if n_2 chooses to gossip with n_3 (i.e. the right-hand branch), then the network is complete by the end of the first round. If it chooses n_1 (i.e. the left branch) this is not the case (there are no paths to n_1) and further rounds of the protocol are required. This leads to a (minimum) expected number of rounds of 1.5. For the maximum case (Figure 4.8(b)), the network is not connected under either choice and several further rounds are required.

The properties analysed in both this and the previous section demonstrate a considerable discrepancy between the minimum and maximum values. To give a simple intuitive explanation for this, consider a chain of nodes of length k (illustrated in Figure 4.9) in which n_1 is trying to pass a message to node n_k. Suppose that all nodes are scheduled in each round and that nodes send only messages to their righthand neighbour (n_i only sends messages to n_{i+1}). Then, the scheduling $n_1, n_2, \ldots, n_k$ would propagate the



(a) minimum (b) maximum

Figure 4.8: Scheduling for the 4 nodes network (first gossiping round).

message to node n_k in a single round but the scheduling $n_k, n_{k-1}, \ldots, n_1$ would require $k-1$ rounds to achieve this.

As the number of nodes in the network increases, so does the amount of nondeterminism present in the model. Hence, the potential influence of the scheduling (the difference between the minimum and maximum values) is also likely to increase. As the comparison of the cases of three and four nodes suggests, however, it may also be the case that, for larger numbers of nodes, the average behaviour is closer to the best-case behaviour than the worst-case.

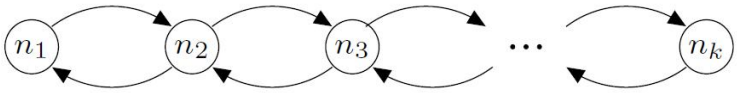


Figure 4.9: Chain of k nodes.

Other properties. The PRISM model we have constructed could also be used to analyse a variety of other properties. For example:

- the maximum probability that a connected network eventually becomes disconnected;
- the minimum probability node ni can communicate with node n_j after k gossiping rounds or t time steps;
- the probability that node ni can communicate with node n_j before it can communicate with node n_k;
- the maximum expected number of updates to the partial views before the network is connected.

Furthermore, the model could easily be adapted to study the effect on performance of a variety of other factors such as failures of network links and the dynamic addition/removal of additional nodes. It could also be modified to study possible ways of preventing the potential inefficiencies highlighted by our analysis. We could, for example, investigate the performance of a modified version of the gossip protocol in which the delays between each node's execution of the protocol is also randomised.

4.3.4 Activities in progress

Although many case studies have been constructed for large systems (up to 10^{10} states), integrating more reduction techniques with stochastic model checking is still necessary because we can always get sys-

tems whose state space cannot be handled by the most powerful model checkers. An important reduction technique *Assume-Guarantee* method has been extended to stochastic model checking in Work Package 2 for non-reward properties. Further extensions to incorporate rewards are under development (see details in D2.1 [53]). Indeed, we are currently working closely with WP2 to enhance assume-guarantee method for quantitative verification of rewards.

4.4 Discussion and Future Directions

This chapter focused on the activities on off-line dependability analyses in CONNECT carried on during the first year of the project. To this purpose, the two approaches adopted in CONNECT are state-based stochastic methods and stochastic model checking.

For reasons already explained, the state-based stochastic method has been applied to assess to some extent dependability metrics of Networked Systems. Specifically, the activity focused on gossip protocols in presence of heterogeneity at level of node communication and computation, which is typically neglected in already existing analyses. Such kind of protocols are expected to be of interest in CONNECT as they may be considered a basic building block for many distributed services. Hence, its analysis, by addressing such heterogeneous aspects, helps in properly set up failure characteristics of gossip-based communication protocols used by middleware and application services. In fact, the knowledge of dependability behavior of the underlying wireless network, such as reliability indicators, can be easily imported as basic failure parameters in the higher (middleware) level dependability model of gossip-based communication protocols, thus facilitating their modeling and evaluation by exploiting better abstraction facilities. In addition to the extension of the diffusion protocols analysis already mentioned at the end of 4.2.3, follow-up of the quantitative assessment through model-based stochastic methods will primarily address other actors of the CONNECT architecture (CONNECTors, Enablers, Networked Systems, CONNECTed Systems). On-going work is addressing the definition of the SAN model and its resolution to assess dependability indicators for the *Distributed Market* scenario identified in D6.1 [54], starting from the LTS specification of the involved Networked Systems and the synthesised CONNECTor under development in Work Package 3.

From a methodological point of view, an effort will be devoted to investigate directions for extending model-based approaches to deal with the dynamic aspects involved in the generation of CONNECTors, so as to allow to some extent an on-line assessment of quantitative dependability properties. Big challenges in this context are represented by the identification of efficient methods for model generation and, especially, model solution so as to provide feedbacks from the analysis in time to be profitably used. Methods based on incremental steps of model definition and refinement, so as to allow for partially pre-determined analysis to be refined/completed at runtime seem to be promising directions to explore. However, forms of partially-dynamic/partially-static methods will be analyzed as well, such as Case Based Reasoning methodologies, where, e.g., a Knowledge Base (KB) repository (or simply a Look-up table LT) could be set up off-line, storing the info on the most appropriate fault-tolerance solution for the connector to be synthesized for specific dependability requirements, on the basis of the results of a (off-line) model-based evaluation activity. At run-time, the KB (or LT) is used to search for the best pre-determined solution mapping the requested non-functional properties (or the closest one, in case a proper match is not found). This way, fast decision-making is achieved. The KB is dynamically extended with new "cases" to be added, to account for interoperability requests of evolving Networked Systems.

As for state-based stochastic models, stochastic model checking is another appropriate off-line approach to verify dependability and performance metrics for CONNECT, which is clearly demonstrated by the introduction in Section 4.3.2 and the case study in Section 4.3.3. The major limitation on the application of this approach in CONNECT is the so-called *state space explosion* problem, which is also the fundamental problem in all other model checking approaches. One way to tackle this problem is the assume-guarantee methods, which are under exploitation in Work Package 2 currently. One of our future research directions on this approach will be to develop of an efficient assume-guarantee method for stochastic model checking rewards in collaboration with WP2, and to apply it to verify dependability and performance metrics we proposed in Chapter 2. Application of an on-line monitoring framework based on stochastic model checking will certainly be an interesting future direction in CONNECT. Again, this will be joint work with WP2.

The two studies on the gossip protocol have shown the complementarities of the two approaches in dealing with dependability assessment: while the formal verification technique is very accurate in determining best and worst case behaviour but for small numbers of involved nodes, the state-based stochastic method is able to provide average values for large-scale networks. Therefore, through these two approaches, a variety of scenarios and user/application needs in terms of dependability analysis can be satisfied.

5 Security Models

The main goal of CONNECT is to allow eternal CONNECTion between different devices. As introduced in Chapter 1, to achieve effective communication several non-functional properties have to be considered and provided, such as reliability, security and privacy, trust and so on. In this chapter, we focus on security concerns.In particular, we aim at an end-to-end security framework based on common security mechanisms for allowing secure communication among different Networked Systems and between Networked Systems and Enablers.

5.1 Motivation and Scope of Research

With reference to D1.1, and as we have previously recalled, the CONNECT architecture is made up of four actors (see Fig. 1.1). In order to establish a communication, these actors interact with each other in the following way: let us suppose a basic case in which two Networked Systems want to establish a communication. To do this, the two Networked Systems send their requirements to an Enabler that is in charge to collect the information and provide a CONNECTor that allows the communication between Networked Systems according to their requirements. Such CONNECTor could be chosen from a pre-existing list of connectors or synthesized on-the-fly.

Our goal is to guarantee that the communication between Networked Systems is always secure. We firstly provide a description of the threat models for the CONNECT scenario by considering that each actor could be a malicious one, *i.e.*, it does not behave as it declares.

We distinguish between local and global security as follows:

- **Local security of each Networked System**. Each Networked System sets a local policy that has to be locally satisfied inside the Networked System itself. Such security policy is the combination of a *private* policy and a *public* one. A private policy is a security policy that a Networked System does not declare to the external world, *e.g.*, a policy that speaks about sensitive data. On the contrary, a public policy is a policy that the Networked System considers freely deliverable, *e.g.*, a policy that does not involve information that the Networked System considers private. Clearly two limit cases exist: i) a Networked System has only a private policy or ii) it sets only a public policy.
- **Global security of the CONNECTed System**. The CONNECTed System has to satisfy a security policy that describes the correct behaviour of the CONNECTed System as a whole.

Going more in detail of the analysis of possible threats, the following cases have to be considered:

- The Enabler could be a malicious agent.
- Each Networked System involved could be a malicious agent.
- No one is malicious.
- Everyone could be malicious, *i.e.*, each Networked System considers malicious both the other Networked System and the Enabler.

As part of CONNECT security, we aim to investigate how the Security-by-Contract paradigm (S×C) [68] can be exploited for guaranteeing that the communication among Networked Systems satisfies all requirements of security in the outlined threat models. Indeed S×C seems to be appealing for CONNECT due to the idea of *contract* of an application. As a matter of fact, contracts can be suitably applied to describe the behaviour of freshly synthesized CONNECTors.

The Security-by-Contract framework (S×C) was originally developed for providing security to mobile applications. Its application scenario consists of a system with a setted security policy, and an application that is going to be run on the system itself. S×C works by checking the contract of the application against the policy. If the check fails then the application could be deleted or the security policy could be enforced on it by exploiting the *enforcement* mechanism of the S×C. On the other hand, the system can proceed to verify whether the contract (correctly representing the application) satisfies the security policy. Once again, if this step fails the solution consists in enforcing the security policy on the execution. Finally,

if the previous checks were positively passed, the application can be executed with no active runtime enforcement.

In particular, here we present a first step along our lines of research by showing a possible application of S×C for dealing with local security of the system in which the Enabler could be a malicious agent.

5.2 Background and State of the Art

In this section we present some background and state of the art about the Security-by-Contract (Section 5.2.1) and verification of security property (Section 5.2.2). These notions are used hereafter in this chapter.

5.2.1 Security-by-Contract paradigm

The Security-by-contract (S×C) is a paradigm for providing security assurances for mobile applications [68]. According to [68, 69], the basic idea of Security-by-Contract is the notion of *contract*. Loosely speaking, a contract contains a description of the relevant features of the application and the relevant interactions with its host platform. Among the relevant features, one can list fine-grained resource control (*e.g.*, silently initiate a phone call or send a SMS), memory usage, secure and insecure web connections, user privacy protection, confidentiality of application data, constraints on access from other applications already on the platform. The mobile code released by a provider is annotated with a contract. By signing the code the developer certifies that the code complies with the stated claims on its security-relevant behaviour.

Definition 5.1 *A* contract *is a formal complete and correct specification of the behaviour of the application for what concerns relevant security actions (* e.g.*, Virtual Machine API call, Operating System Calls). It defines a subset of the traces of all possible security actions.*

On the other side, the other cornerstone of the S×C approach is the concept of *policy*. A mobile platform could specify platform contractual requirements, here referred to as a policy. Such policy could be also set by a user. In both cases, the policy should be matched by the contract of the application before the execution in order to guarantee security requirements on the mobile platform.

Definition 5.2 *A* policy *is a formal complete specification of the acceptable behaviour of applications to be executed on the platform for what concerns relevant security actions (*e.g.*, Virtual Machine API call, Operating System Calls). It defines a subset of the traces of all possible security actions.*

The Security-by-Contract paradigm works as follows: when a client receives an application. it verifies if the code and the contract actually match by an *evidence checking* procedure. If the check fails then the user can decide to delete the application or to enforce a security policy on it by exploiting the *monitoring/enforcement* infrastructure. Otherwise, the system can proceed to verify whether the contract (correctly representing the application) satisfies the user's policy. Once again, if this step fails the solution consists in enforcing the active policy on the execution. Finally, if the previous checks were positively passed, the application can be executed with no active runtime monitor (see also Fig. 5.1) . The Contract-Policy matching function ensures that any security relevant behaviour allowed by the contract is also allowed by the policy. This matching could be done with respect to different behavioural relations, *e.g.*, language inclusion [61] or simulation relation [87]. This matching function allows the user that is going to execute the application to understand if the behaviour of the application itself is compliant with the set of policies without running it.

The enforcing approach has been shown to be feasible on mobile devices. In particular two techniques have been detailed in the literature and exploited for experiments and tools: JVM customization [36] and bytecode in-lining [57]. Briefly, the first replaces the standard JVM with a modified one dispatching signals to the monitoring/enforcement agent whenever a program makes a call to (a subset of) the system APIs. The second instruments the sequence of bytecode instructions with invocations to the security policy monitor making the program send security signals at run-time. Both approaches use an external system, namely a *Policy Decision Point* (PDP), holding the set of rules that compose the security policy. Moreover the PDP reads the current device state (battery consumption, link strength, available credit) through dedicated internal components. When the PDP receives a request for an action violating the

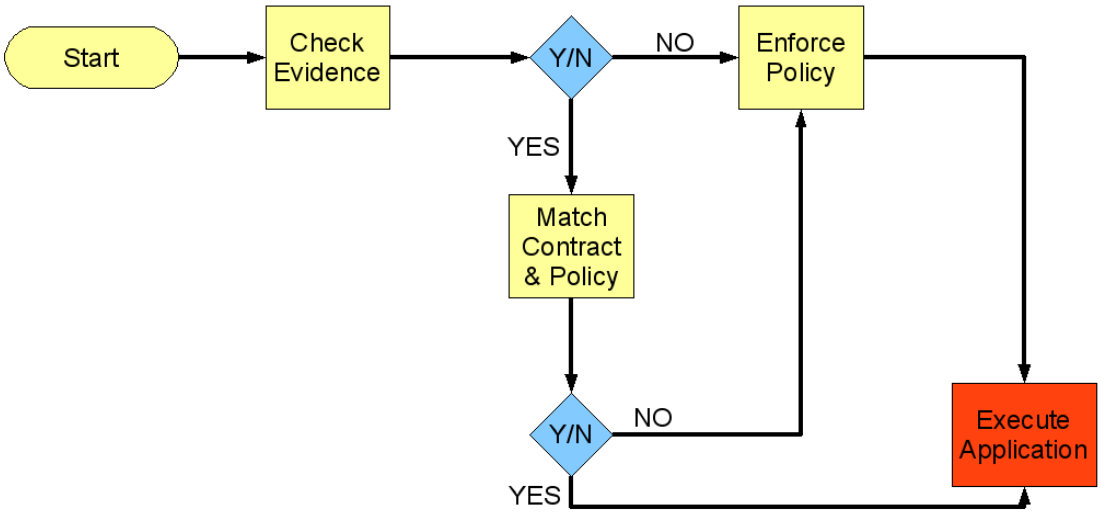


Figure 5.1: The Security-by-Contract Application lifecycle [69].

security policy, it answers denying the necessary permission. Then, the system reacts by throwing an exception.

Extensions of the Security-by-Contract paradigm for dealing with fine-grained access control, usage control and trust management have been proposed. In particular some work has been done for integrating trust management and fine grained access control in Grid Architecture. An attempt can be found in [109] where an access control system is proposed that enhances the Globus toolkit with a number of features. This copes with the fact that access control policies and access rights management become one of the main bottlenecks using Globus for sharing resources in a Grid architecture. Along this line of research, an integrated architecture, extending the previous one, with an inference engine managing reputation and trust credentials is presented in [51]. This framework is extended again in [108], where a mechanism for trust negotiating credential has been introduced to overcome scalability problem. In this way the framework provided preserves privacy credentials and security policy of both users and providers.

A reputation mechanism to facilitate the trustworthiness evaluation of entities in ubiquitous computing environments is proposed in [120]. It is based on probability theory and supports reputation evolution and propagation. The proposed reputation mechanism is also implemented as part of a dependability-aware Web service discovery middleware and evaluated regarding its overhead on service discovery latency.

5.2.2 Verification of security properties

Off-line verification often adopts temporal logic, such as CTL (Computation Tree Logic) [47] and LTL (Linear Temporal Logic) [161], as the specification language. The temporal logic formalise the temporal behaviour of systems being checked, such as "*variable x is always greater than or equal to 0 in all runs*" and "*eventually x is less than 0 in some run*". While temporal logic in its various forms has proven essential to reason about reactive systems, agent-based scenarios are typically specified by considering high-level agents attitudes. In particular, specification languages based on epistemic logic [71], or logics for knowledge, have proven useful in a variety of areas including robotics, security protocols, web-services, etc. For example, security specifications involving anonymity [38] are known to be naturally expressible in epistemic formalisms as they explicitly state the lack of different kinds of knowledge of the principals. Epistemic logic can specify a single agent's knowledge, such as "*agent A knows x is equal 1*", as well as knowledge shared by a group of agents, e.g., "*agents A and B both knows x is equal to 1*" and "$x = 1$ *is common knowledge among agents A, B and C*". In recent years, temporal epistemic logic [164], a

combination of temporal logic and epistemic logic, has been introduced for the verification of electronic contracts [5, 126, 128] and security protocols [49, 50, 124]. It can describe system behaviour from both temporal and knowledge aspects, such as "*whenever a violation occurs, any contract party that does not violate the contract knows the contract will be recovered*" or "*At any time, an intruder does not know it can decode the content of a packet it receives.*" Using temporal epistemic logic, we are not limited to verify safety and liveness properties, and therefore, more behaviour can be checked in order to eliminate system vulnerabilities.

5.3 The Security-by-Contract Paradigm in the CONNECTed World

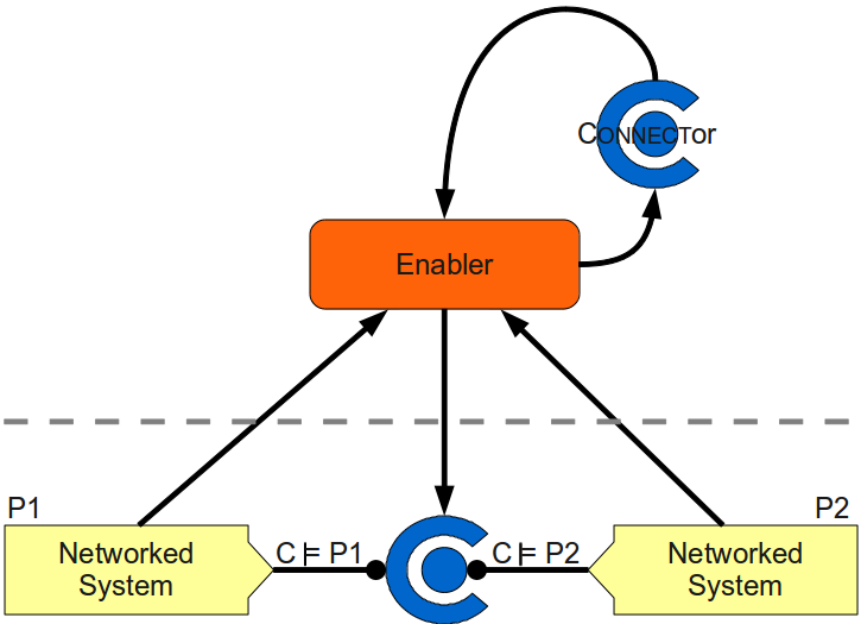


Figure 5.2: Secure CONNECTion

Here we are going to show how the framework described in [56], based on *Security-by-Contract* (S×C), can be adopted in the CONNECT world for guaranteeing security aspects in the CONNECTion phase. Indeed, thinking to the CONNECT world, the Security-by-Contract paradigm can be applied for guaranteeing security properties during the CONNECTion phase [58]. A graphical representation of how S×C works in CONNECT is given in Figure 5.2.

According to Figure 5.2, let us consider two Networked Systems that want to communicate. Hereafter we tread only local security aspects and in the scenario we investigate the only possible malicious agent is the Enabler. Each Networked Systems has a security policy set on it. This security policy is a combination of a private and a public policy. Several cases arise as follows:

1. The public policy is a generalization of the private policy. This means that the public policy allows more executions than the private one. A particular instance of this case occurs when only the private policy has been defined.

2. The private policy is a generalization of the public one. Referring to the notion of compliance used in the S×C paradigm, this case means that whenever the public policy is satisfied also the private one will be. An instance of this case occurs when no private policy has been set on the Networked System.

3. Private and public policy partially overlap. There exists at least one execution that satisfies both the private and the public policy.

Some examples can clarify the above classification.

Example 5.3 *Imagine a Networked System requiring a* CONNECT*or and declaring the following, informally defined, policies:*

- P_{pub}*:* Always connect to the same host
- P_{priv}*:* Encrypt every message with key κ

As it refers to some sensitive data, i.e.*,* κ*, the Networked System can be interested in keeping* P_{priv} *secret. Clearly, these two policies do not imply each other (unencrypted messages can be delivered to a single host or correctly encrypted messages can be sent to multiple hosts). However, there are executions satisfying both of them,* e.g.*, a single, encrypted message is sent. This example belongs to the third case introduced above.*

Example 5.4 *Imagine now a Networked System similar to the previous one, but defining a different private policy.*

- P_{pub}*:* Always connect to the same host
- P_{priv}*:* CONNECT only with host h

Here, the set of behaviours accepted by P_{priv} *also satisfies* P_{pub}*. Hence we are in the second case.*

In addition to the relation existing between private and public security policy, when we apply the Security-by-Contract framework, we must also investigate the relation among security policies and applications contract. All possible relations are summarized in Figure 5.3.

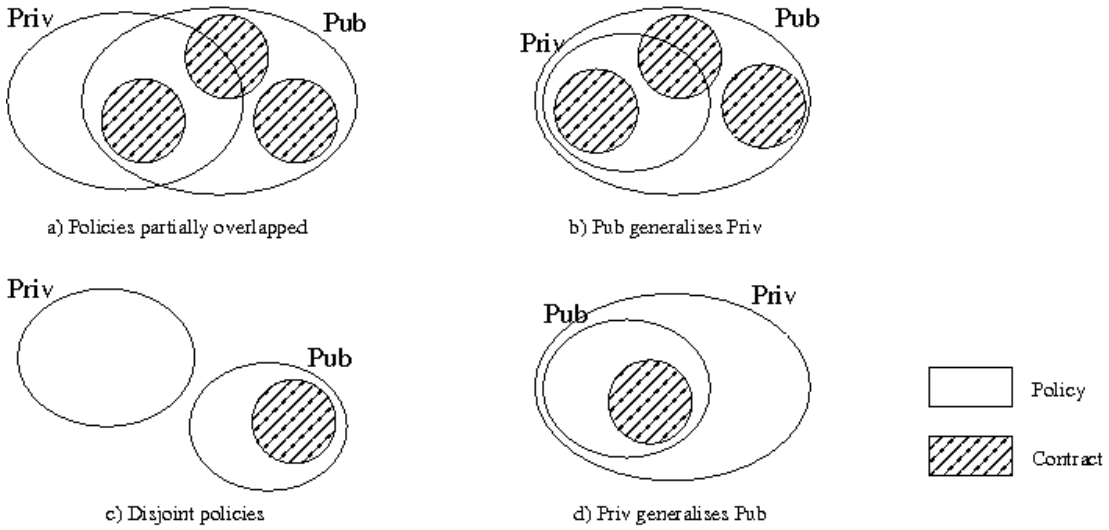


Figure 5.3: Relations among Private policy, Public policy and Contract

In this contribution, we assume that the public policy generalizes the private policy (case (b) in Fig. 5.3). The relationship between the policy and the contract drives the Security-by-Contract run-time strategy.

Let P_1 and P_2 be the public policies of the two Networked Systems. In order to communicate, both Networked Systems send a communication request and their local public policies to an Enabler that has to provide a CONNECTOr that allows the communication between them. Such a CONNECTOr will be an application, that the Enabler may chose among a set of already existing connectors piece or it synthesizes new ones on-the-fly. In both cases, the Enabler provides to each Networked System a CONNECTOr and a contract C that describes the CONNECTOr behaviour. Moreover C satisfies both P_1 and P_2. We are considering the case in which both Networked Systems have also a private policy, P_{1priv} and P_{2priv} respectively. Note that, being not aware about private security policies of the Networked Systems, the

Enabler cannot guarantee the CONNECTor will comply with them. Each Networked System involved in the communication has to locally check if the CONNECTor contract is compliant with both its local public and private security policies. Figure 5.4 shows schematically the system behaviour. If the contract complies with the private policy (M in figure), the system monitors the application run-time compliance with respect to its contract. Instead, if the contract does not comply with the private policy (E in figure), the private policy is enforced on the executing application. Note that we are intentionally ignoring the case in which private policy and contract have no intersection. Indeed, whenever this happens, the enforcement process forces the execution to generate only empty traces.

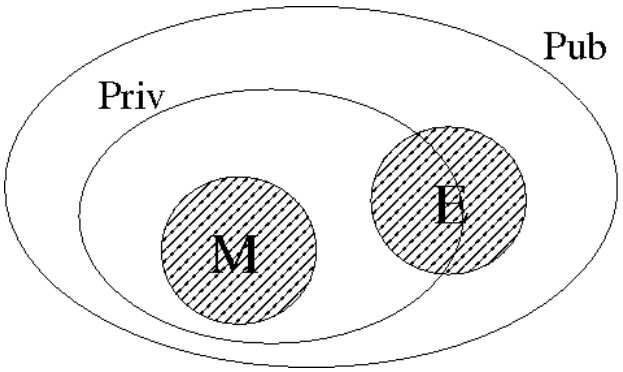


Figure 5.4: A graphical representation of scenario (b)

Note that, whenever the private security policy is a generalization of the public one, as it is in this case, once the Enabler has guaranteed that the contract satisfies the public security policy we have, for granted, that the contract satisfies also the private one. Consequently, checking the compliance of the CONNECTor with respect to its contract is useless. Satisfying the public security policy is enough for our purpose.

5.4 Management of the Enablers' Level of Trust

The security model described in the previous section considers to have one single possible malicious Enabler. Let us consider to have more than one Enabler. In this scenario, the S$\times$C paradigm can be extended for dealing with trust in quantitative way. In particular, it could be extended by considering to be able to measure the level of trust of the Enabler (the model of CONNECT trust management is described in Chapter 6) by adding a contract monitoring (see Chapter 7) that allows us to check if the execution of the CONNECTor adheres to the contract of the CONNECTor itself and, according to the answer, we update the level of trust of its Enabler.

Let us consider to have a set of Enablers to which Networked Systems send a communication request and their local public security policies. Each Enabler provides a CONNECTor with its contract C. Now the two Networked Systems have to decide which CONNECTor is better to use. The decision could be taken by considering several factors. One of those could be considering the level of trust of the Enabler that provides the CONNECTor.

Our strategy takes place in two phases: at deploy-time by setting the monitoring state and at run-time by applying the contract monitoring procedure for adjusting the Enabler trust level.

We deploy this model with a framework with quantitative trust in such a way that it is possible to update the level of trust dynamically according to the adherence between the real execution of the CONNECTor and its contract. As a matter of fact we extend the existing architecture by adding a *contract monitoring*, whose functionality is described more in detail in Section 7.3.4. The contract monitoring checks the compliance between the CONNECTor and its contract. Hence the extended Application/Service Life-Cycle results as in Fig. 5.6. In fact, we replace the Networked System Check Evidence by a Trusted Provider that manage the level of trust of the Enabler.

Whenever the contract satisfies the local public policy, two possible relations with the private policy arise:

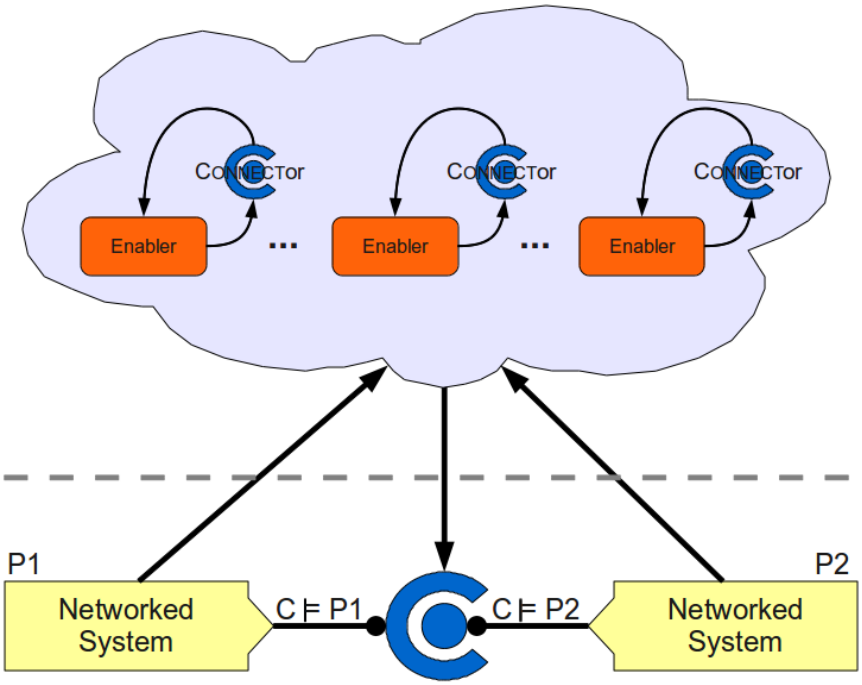


Figure 5.5: Secure and trusted CONNECTion

1. The contract satisfies the local private security policy. In this case our monitoring/enforcement infrastructure is required to monitor only the application contract. Indeed, under these conditions, contract adherence also implies private policy compliance. If no violation is detected during the execution then the application worked as expected. Otherwise, we discovered that a trusted party provided us with a fake contract. Our framework reacts to this event by reducing the level of trust of the indicted provider and switching to the policy enforcement modality.

2. The contract does not satisfy the local private security policy. Since the contract declares some potentially undesired behaviour, policy enforcement is turned on. Similarly to a pure enforcement framework, our system guarantees that executions are policy-compliant. However, monitoring contract during these executions can provide a useful feedback for better tuning the trust vector. Hence, our framework also allows for a mixed monitoring and enforcement configuration. This configuration is statistically activated with a certain probability. Several strategies are possible for deciding the probability to activate the monitoring procedure, *e.g.*, using a fixed value or a function of the current trust level.

Let us notice that, in the second scenario, the verification Networked System plays a central role. Indeed, a negative result denotes that a trusted provider released a fake contract. Clearly, this event means that the previous trust value must be updated. Moreover, this Networked System could be missing or unavailable (*e.g.*, due to the limited device resources). In this case we reduce ourselves to the mixed (*i.e.*, monitoring and enforcing) scenario.

5.5 Off-line Verification of Security in the CONNECTed World

In the CONNECT world, a large number of heterogeneous Networked Systems may exist and try to communicate with each other. It could be hard to design an appropriate contract policy for each kind of Networked Systems to allow the communication. For example, an important issue is how to guarantee there are no conflicting clauses in two Networked Systems' policy prohibiting the mutual communication. It might be very expensive to correct conflicting clauses after the CONNECTed System is deployed. We also need to consider the robustness of security contracts. Such a contract should be carefully checked

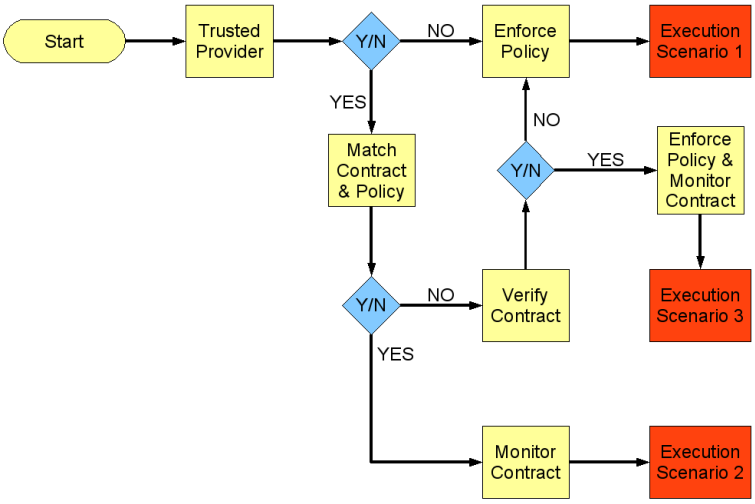


Figure 5.6: The extended Security-by-Contract application lifecycle

against malicious attack. Further, the overall system should not be made vulnerable by various contracts. In this setting, off-line verification of security aspects has been proved to be very useful.

In addition to quantitative verification of security using stochastic model checking described in Section 4.3 and in D2.1 [53], e.g., [150], knowledge-based properties cannot be easily handled by stochastic model checking so far. MCMAS [125] is an agent-based symbolic model checker tailored for verification of knowledge. Although the research mentioned in Section 5.2.2 on the verification of contracts and security protocols was carried out separately, most of them is based on MCMAS, which provides us with a platform to verify security-by-contract models. MCMAS supports specifications based on CTL, epistemic logic (including operators of common and distributed knowledge), ATL (Alternating Time Logic) [11], and deontic modalities for correctness [127]. The model input language includes variables and basic types and implements the semantics of interpreted systems, thereby naturally supporting the modularity present in agent-based systems. MCMAS implements OBDD-based algorithms optimised for interpreted systems and supports fairness, counter-example generation, and interactive execution (both in explicit and symbolic mode).

In principle, verification of security properties of a CONNECTed System can be done in the following way. Figure 5.7 shows the high level of translating the system into an ISPL model, which is then fed to MCMAS for verification. Generally speaking, each CONNECTor, Enabler or Networked System is mapped to an agent. Precisely, the behaviour of a CONNECTor (Enabler or Networked System) is modelled by an agent. In addition, a special agent *environment* can be created to cover everything that is outside CONNECTors, Enablers and Networked Systems. The policies in the system are translated into temporal epistemic logic formulae. Moreover, user-defined properties characterising the system can be added to the ISPL model. MCMAS takes the ISPL model and returns the verification results for the formulae. These results can be used as reference to change the design of CONNECTors/Enablers/Networked Systems, as well as the policies. More detail will be investigated as a future work.

5.6 Discussion and Future Directions

In this chapter, we presented a contract-based framework for security to be adopted and integrated in CONNECT. In particular, we show a strategy for establishing a secure communication among Networked Systems by guaranteeing that they are CONNECTed through the usage of a secure CONNECTor, *i.e.*, a CONNECTor that works according to its contract, which is compliant with the security policies required by

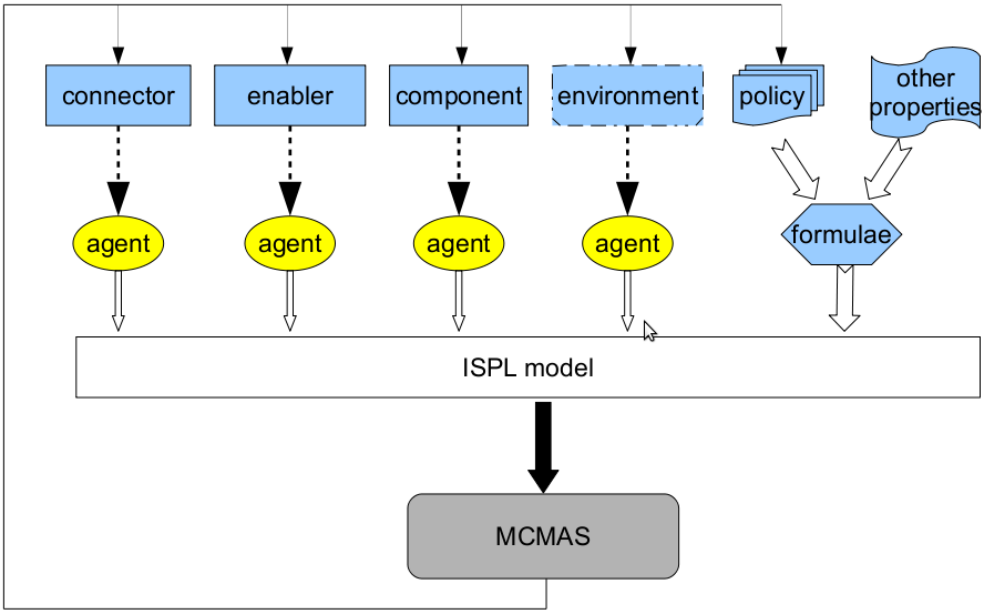


Figure 5.7: Verification flow

Networked Systems involved in the communication.

In Chapter 7 we also describe how we plan to extend the Security-by-Contract approach for managing trust measures by a contract monitoring strategy. As a matter of fact, at deploy-time, the monitoring/enforcement structure is decided depending on both the CONNECTor contract and the credentials (*i.e.*, trust measure) of the Enabler that providers the considered CONNECTor. The main novelty of this model consists in the contract monitoring scenario. At run-time, a trusted CONNECTor violating its contract leads to a correction of the trust relationship with the Enabler and activates the policy enforcement configuration of our system.

We also propose a general approach to apply off-line verification to check security aspects in the CONNECT world. Particularly, we are interested in model checking temporal epistemic properties for the overall Security-by-contract framework. After the framework has been constructed for the CONNECT world, we would like to project our result much deeper in this direction to enhance the framework's safety.

Many other future directions are viable.

Regarding the Security-by-Contract approach, we aim to improve the S×C and the S×C×T paradigms for dealing with all possible threat models we have pointed out in the introduction. Mainly, we will work for integrating a trust management strategy. Currently, trust weights can only decrease monotonically as a consequence of contract violations with the only exception of a direct intervention of the user, see *e.g.*, [146]. Moreover, quantitative representation of this adjustment we consider to present in future work.

In the presented approach, we have spoken of privacy w.r.t. privacy of policy. Indeed, we have considered that each Networked System has a private policy for protecting , *e.g.*, access to its resources. In the future, we aim at investigating also data privacy. In particular we would like to study how private data will be treated once they are sent to another Networked System via a CONNECTor.

We also aim to take into account the problem of the composition of policies within different Networked Systems. As a matter of fact, the CONNECTor's contract has to satisfy the composition of the policy provided by all Networked Systems involved in the communication phase. How much the contract adheres to the composition of policies could be considered as another criterion for choosing one connector over another.

6 Trust Model

In this chapter we define the CONNECT trust model, which serves as a basis of the development of an autonomous system managing the trustworthiness of three CONNECT actors, namely, Networked Systems, Enablers and CONNECTors (see Figure 6.1).

In accordance with the dependability-related metrics defined in Chapters 2 and 3, we define the CONNECT trust model to satisfy the following needs:

- Allowing Networked Systems to apply their trust policies;
- Allowing Enablers to safely cooperate in order to build and deploy CONNECTors;
- Allowing Enablers to assess CONNECTor trustworthiness and hence providing CONNECTed Systems with the most trusted CONNECTor(s);
- Handling monitoring feedbacks to fairly update the trustworthiness of CONNECT actors;
- Defining a flexible bootstrapping process to allow the trust model to continually evolves (trust survivability aspect).

This chapter is structured as follows: Section 6.1 presents the background we build upon. Then, Section 6.2 introduces the CONNECT trust model, which leads to define the trust assessment process in Section 6.3. Finally, we discuss our current work in Section 6.4.

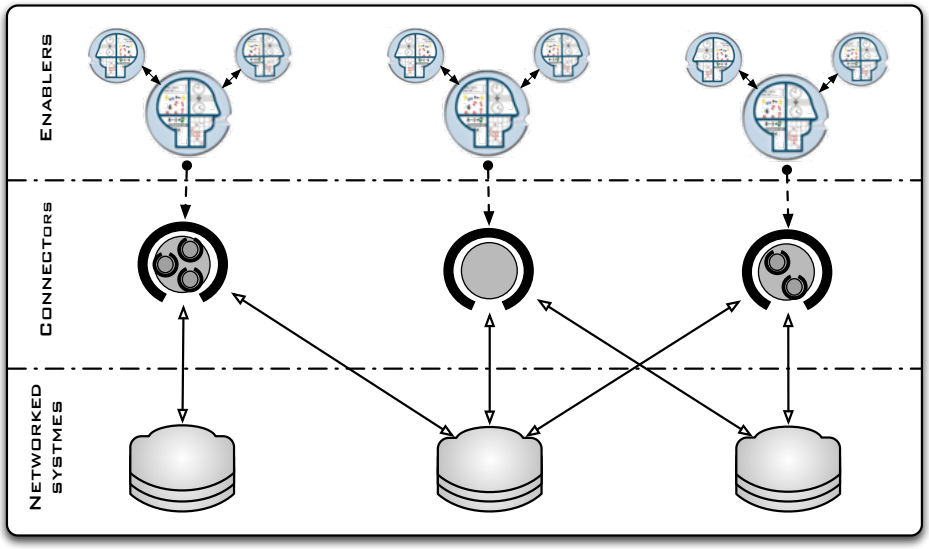


Figure 6.1: Basic scenario

6.1 Background

As widely discussed in Chapter 3, there has been substantial research on the concept of trust in the field of social sciences. Findings have been applied in various areas including economics, finance, management, government, and psychology. In recent years, trust has generated considerable interest in the computer science community as the basis of security solutions for various distributed systems, such as *ad-hoc* networks, pervasive environment, Grid, Web services, etc.

A trust model can be decomposed into: the definitions of trust relations (Section 6.1.1), trust assessment (Section 6.1.2), trust bootstrapping (Section 6.1.3), and risk management, which plays a key role (Section 6.1.4).

6.1.1 Trust relation

Manifestations of trust are easy to recognize because we are confronted to this paradigm everyday, but at the same time trust is more complex than it seems considering it manifests itself in many different forms (issues have been previously discussed in Section 3.5). As illustrated in Figure 6.2, there are three different classes of trust relations depending on their *arity* (i.e., the number of involved actors).

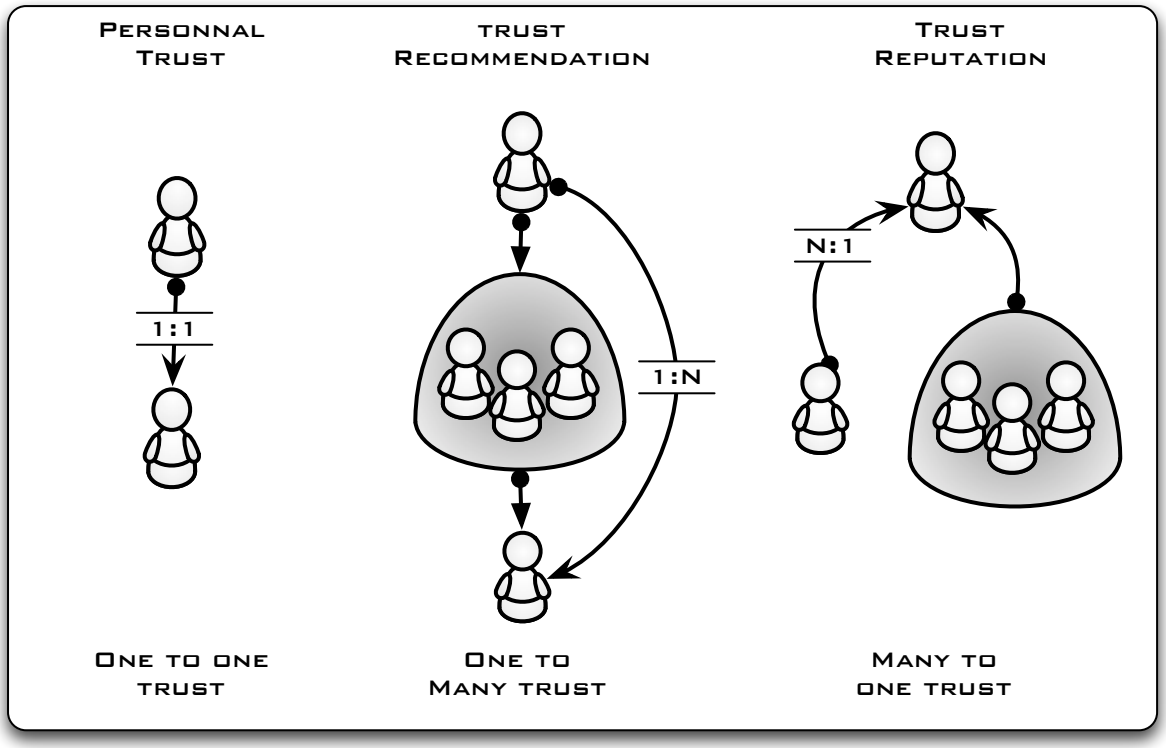


Figure 6.2: Trust relations

[One-to-One] Trust relation

The One-to-One (1:1) trust relation defines a direct relationship that links one trustor (**1**) to one trustee (**1**). This relation is maintained locally by each trustor and represents trustor's personal opinion regarding its trustees. The 1:1 relation may, for instance, represent: a belonging-driven relationship (e.g., an employee trusts his company), a social-driven relationship (e.g., trust among friends), a profit-driven relationship (e.g., a person trusts a trader for managing its portfolio), etc.

[One-to-Many] Trust relation

The One-to-Many (1:N) trust relation allows each trustor (**1**) to indirectly trust a trustee through a trustworthiness group recommendations (**N**). This relation results from the composition and/or aggregation of many 1:1 trust relations. The composition of 1:1 trust relations usually represents a trust transitivity through a trust path, where each entity can trust unknown entities based on the recommendation of its trustees. Thus, this relationship is built by composing personal trust relations tow at a time [6, 173]. Furthermore, in case of existing several trust paths that link the trustor to the recommended trustee, the aggregation can be used to aggregate all given trust recommendations [103].

More recently in the Web Service Oriented Architecture (WSOA), the 1:N trust relation represents the trustworthiness of Web Services (WS) composition [148, 155, 106]. In this case, the aggregation of sub-services recommendation allows defining the trust recommendation for the whole composition.

[Many-to-One] Trust relation

The Many-to-One (N:1) trust relation represents the aggregation of many personal trust relations that have the same trustee. Hence, the N:1 trust relation allows defining the reputation of each trustee. Such reputation can be used, by any entity as a reference, to define its trust relation with unknown ones (e.g., *"I trust you because of your good reputation"*,*"I distrust you because of your bad reputation"*, *"I trust you less than before due to your current bad reputation"* or *"I trust you more than before due to your current good reputation"*).

In the literature, existing reputation systems are divided into two categories: (i) *Centralized reputation systems*, where the reputation of each participant is collected and made publicly available on a centralized server (e.g., eBay, Amazon, Google, [151]) ; and (ii) *Distributed reputation systems*, where centralization is not possible because there is no trusted or scalable central location. In this case, the reputation is spread throughout the network where each networked entity is responsible to manage the reputation of the other entities ([103, 195, 197, 182], etc.).

6.1.2 Trust Assessment

The number of computational trust models has been increasing rapidly in recent years. Thus, according to the aforementioned classification we introduce common solutions from related work that address the assessment process for each kind of relation (i.e., 1:1 , 1:N and N:1 trust assessment).

[One-to-One] Trust assessment

One-to-One trust relations are assessed and measured in different ways. In [200], the 1:1 trust relation is assessed by qualitative labels (e.g., high trust, low trust etc.). Other solutions represent 1:1 trust relations by a numerical range. For instance, this range can be defined by the Interval [-1..1] (e.g., [135]), [0..n] (e.g., [85, 6, 173]) or [0..1](e.g., [103, 189]. The latter range usually represents a probability, which takes into account subjective variables such as uncertainty.

[One-to-Many] Trust assessment

All the solutions dealing with 1:N trust assessment mainly compose and aggregate trust recommendations by computing the average [173], the minimum [136] or the product [6] of all the intermediary trust values. In Web service composition, some approaches (such as [155]) evaluate each sub-service recommendation by assessing its provider, whereas other approaches (such as [148, 106]) assess the sub-service itself in terms of its previous invocations, performance, reliability, etc. Almost all of these approaches perform the trust composition and/or the aggregation according to the service composition flow (sequence, concurrent, conditional and loop). More recently, in [148], the authors tackle another problem that consists of how to fairly reward or penalize service providers from the behavior of the whole composition. The authors propose to penalize or reward service providers, proportionately to their degree of involvement in the whole composition.

[Many-to-One] Trust assessment

Bayesian systems are often used for N:1 trust relation (reputation) assessment. It represents the trust value by a beta Probability Density Function (PDF) [194], which takes binary ratings as input (i.e., positive or negative) from all trustors. Thus, the reputation score is updated from the previous reputation score and the new rating [103, 151]. The advantage of Bayesian systems is that they provide a theoretically sound basis for computing reputation scores and can also be used to predict future behavior. Other solutions [182, 13] use the fuzzy logic approach, which offers the ability to handle uncertainty and imprecision effectively, and is therefore ideally suited for interpreting trust. In contrast to Bayesian inference, the Fuzzy inference copes with fuzzy inputs, such as assessments of quality, and allows inference rules to be specified using imprecise linguistic terms, such as "very high quality" or "slightly late".

6.1.3 Trust bootstrapping

Most of the existing trust models focus on the assessment process and neglect the bootstrapping stage. The trust bootstrapping challenge consists of deciding how to initialize trust relations in order to efficiently start the system and also allows newcomers to join the running system [131].

Trust initialization (i.e., a priori trust) is usually not addressed. Almost all existing solutions initially evaluate trust relation with a fixed value (e.g., 0.5 [92], a uniform Beta probabilistic distribution [103], etc), which do not reflect reality. In general, existing solutions tackle this problem: by automatically initializing existing trust relations according to given peers recommendations[163], by defining various trust disposition level, i.e., pessimistic, optimistic, and undecided [158], by applying a sorting mechanism instead of affecting fixed values [173] or by performing a learning mechanism that computes the degrees of proximity and similarity between entities to automatically deduce and evaluate pre-existing relations [198].

For the CONNECT trust model, we investigate the definition of an assessment mechanism based on the risk factor to manage the bootstrapping process.

6.1.4 Risk management

One of the most important challenges in trust model is the risk perspective [86, 132]. In fact, riskless exchanges are practically impossible and usually risk increases with the number of involved actors. Thus, we can consider that the risk aspect is unavoidable for any trust model.

Risk and trust mutually interfere (risk vs. certainty, gains vs. losses, risk vs. ambiguity). To compensate risk taking, some proactive solutions, such as, law enforcement, insurance and traceability mechanisms are defined to prevent abusive behavior.

For our trust model, we define a mechanism to optimize the benefit/risk ratio. In fact, a well balanced benefit/risk ratio is essential for the CONNECT architecture especially for the durability of the CONNECT system by enabling the bootstrapping of newcomers. For instance, a given Enabler (i.e., newcomers) may adopt a risky behavior in order to be selected and hence increase its reputation. Then, after a while, this Enabler may take a risk-averse behavior in order to preserve and maintain its acquired reputation.

In the following we introduce the CONNECT trust model (see Section 6.2) in order to identify the trust relations and model trust interactions to efficiently assess the trustworthiness of the CONNECTed System (see Section 6.3).

6.2 CONNECT Trust Model

The objective of the CONNECT trust model is to make the CONNECTed System as much trustworthy as possible. Thus, in order to define all trust relations that link CONNECTor, Enablers and Networked Systems, we identify, for the CONNECT trust model, the following trust actors (see Figure 6.3):

- **Networked systems (N):** The entities that need to be CONNECTed.
- **Enablers related to Networked System (A):** All the Enablers that work with Networked Systems. These Enablers represent the Access point of the CONNECT architecture, and are mainly responsible to discover Networked Systems (Figure 6.3 step 1) and to learn (i.e., Learning Enabler) their behavior (Figure 6.3 step 2) in order to provide the Enablers that are related to CONNECTor with relevant inputs.
- **Enablers related to CONNECTors (E):** The Enablers that handle the CONNECTor (e.g., Synthesis Enabler). These entities are mostly responsible to synthesizing (Figure 6.3 step 3) or hosting (Figure 6.3 step 4) the CONNECTor.
- **CONNECTors(Co):** They represent the glue that are created by the Enabler(s) E.

Thanks to CONNECT Enablers, Networked Systems get CONNECTed via CONNECTor(s) that can be composed and reused. Thus, within the CONNECT architecture we can have several Networked Systems, several CONNECTors and several Enablers (related to Networked System and to CONNECTor).

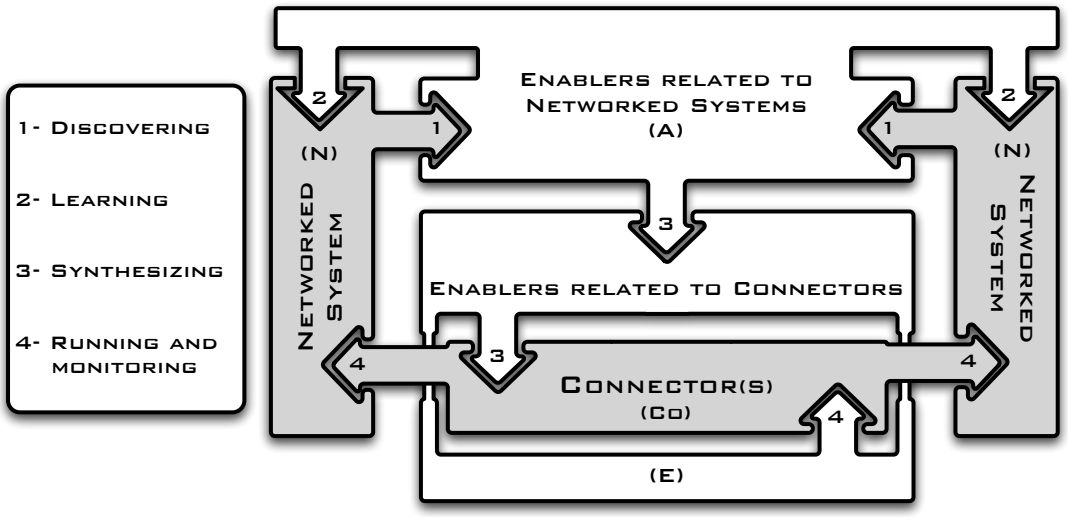


Figure 6.3: CONNECT actors

In this section, we introduce the CONNECT trust model with the defined actors, by firstly defining and formalizing CONNECT Trust relations (see Section 6.2.1) in order to modeling a trust graph of the CONNECT architecture (see Section 6.2.2).

6.2.1 CONNECT trust relations

As shown is Figure 6.4, the CONNECT trust model is composed of three categories of trust relations as follows:

	N	A	E	Co
N	1:1/N:1Trust	1:1/N:1Trust	NO	NO
A	NO	NO	N:1 Trust	NO
E	NO	NO	1:1/N:1 Trust	1:N/1:1 Trust
Co	NO	NO	NO	NO

Networked System relations	Enabler relations	CONNECTor relations

Table 6.1: The CONNECT trust relations

- **CONNECTor Relations:** These relations is the core of the CONNECT trust model. They represent all trust relations that allow the evaluation of the CONNECTors. As illustrated in the table 6.1, the trust connection between E and Co defines two trust relations. The first is direct trust relation (1:1 trust), which allows each E to assess its synthesized CONNECTor in term of previous deployment. The second Trust relation is a represent a 1:N Trust relation. It allows the Enabler E to assess the recommendation of its synthesized CONNECTors, which can be a simple or a complex CONNECTor (assessing complex CONNECTor means assessing more than one sub-CONNECTor and also more than one Enabler). The assessment process is made of 'E to E' trust relations (direct trust and reputation) and also all direct trust relations that link the involved Enablers E(s).

- **Enabler Relations:** These relations mainly define two kind of links, between the Enablers E and between the Enablers A and E. (i) The former trust relation represents direct trust relation (1:1 trust or N:1) that is maintained by each Enabler E with one or more E(s). This relation is used by the

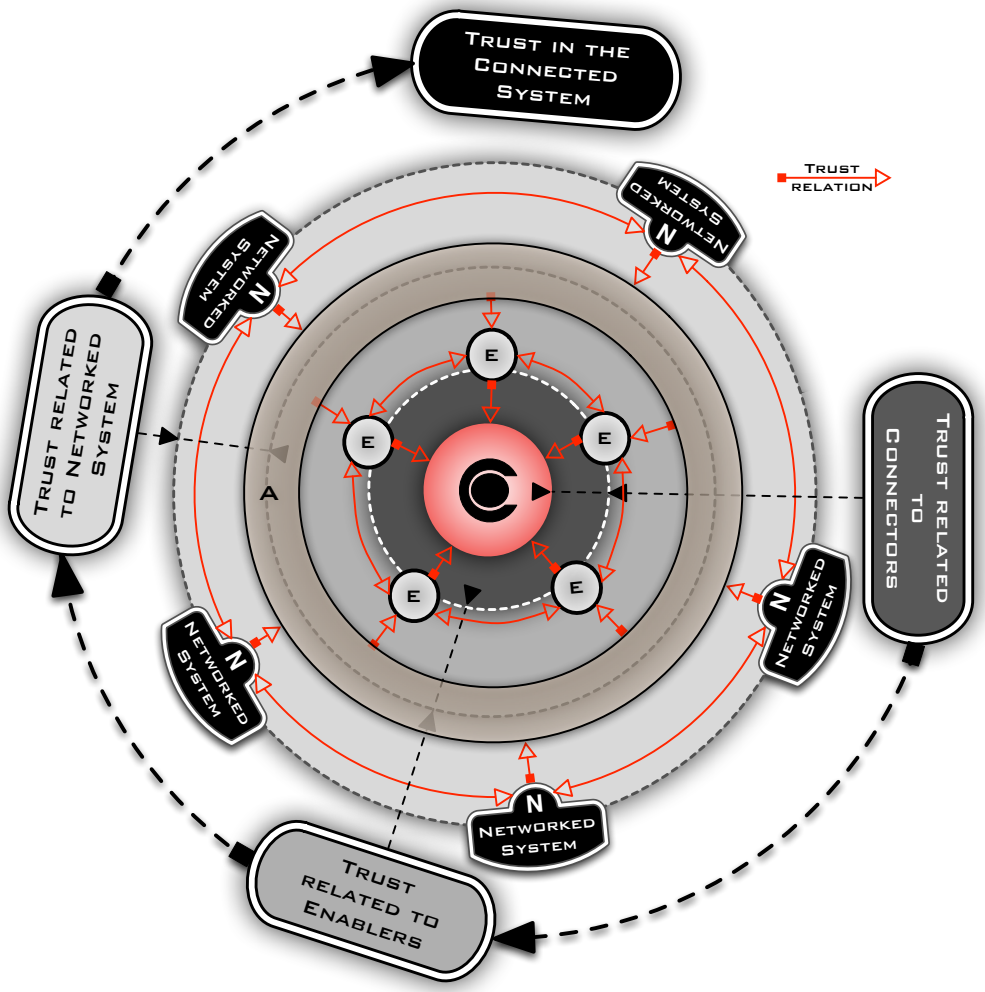


Figure 6.4: The CONNECT trust model

CONNECT trust model to compute and to update the reputation of each E. (ii) The second relation is defined by the Enabler A which trusts only Enablers E with high reputation (N:1 trust).

- **Networked system Relations:** These relations are the bridge that links Networked Systems with the CONNECT trust model. In fact, at this stage two kinds of trust relations are managed, relations among Networked Systems N and relations between N and the Enablers A.

Note that, in this deliverable, we focus on the CONNECTor and the Enabler relations, whereas the Networked System relations will be considered in future work. This in particular leads to postpone the definition of trust relations among the Enablers A. Thus, for the rest of the chapter we deal with only one Enabler A.

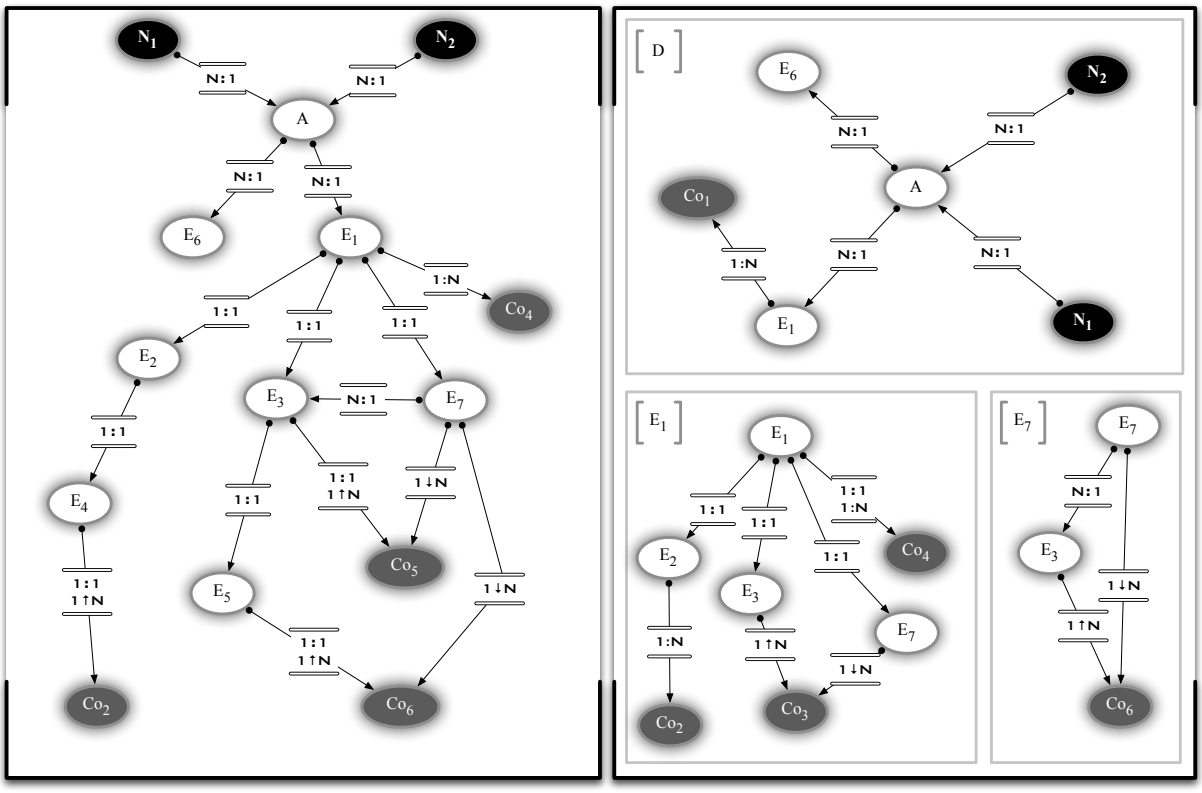
6.2.2 CONNECT trust graph $CoTG$

Let be $\mathbb{S}$ the set of CONNECT actors and $\mathbb{T}$ be the edges that links $\mathbb{S}$ members. We call the CONNECT Trust graph, and we note $CoTG(\mathbb{S}, \mathbb{T})$ the follows labeled and directed graph:

- The vertex represents one element of $\mathbb{S}$. The set $\mathbb{S}$ is the union of $\{\mathbb{N} \cup \mathbb{A} \cup \mathbb{E} \cup \mathbb{C}\}$, where:

- $\mathbb{N}$ is the set of Networked Systems.
- A is the Enabler A, which is considered as the access point of the CONNECT trust graph.
- $\mathbb{E}$ is the set of Enablers E.
- $\mathbb{C}$ is the set of CONNECTors.

- Each Trust relationship between two vertices is represented by a directed edge. Consequently, each edge is labeled by one of these labels $\{1{:}1,\ \text{N:1}, 1{\uparrow}N, 1{\downarrow}N, 1{:}\text{N}\}$ where:
  - $X \xrightarrow{1:1} Y$ represents a personal trust relation that links X to Y.
  - $X \xrightarrow{N:1} Y$ means that X trusts the reputation of Y.
  - $X \xrightarrow{1\uparrow N} Y$ represents the trust recommendation of X for synthesizing the connector Y. Thus, X represents the Enabler that synthesizes the CONNECTor Y and we note $X = \uparrow Y$.
  - $X \xrightarrow{1\downarrow N} Y$: represents the trust recommendation of X for running the CONNECTor Y. Thus, X represents the Enabler that hosts the CONNECTor and we note $X = \downarrow Y$.
  - $X \xrightarrow{1:N} Y$: represents the trust recommendation of X for synthesizing ($X \xrightarrow{1\uparrow N} Y$) and running ($X \xrightarrow{1\downarrow N} Y$) the CONNECTor Y.



(A) EXAMPLE OF A GLOBAL TRUST VIEW

(B) EXAMPLES OF A LOCAL TRUST VIEW

Figure 6.5: Example of a CoTG graph

Thus, the CONNECT trust Graph is able to represent all trust interactions among CONNECT actors. In the Figure 6.5(a) we illustrate the whole conception of the CONNECTor Co_1 that makes N_1 and N_2 CONNECTed (Global trust view). The CONNECTor Co_1 is composed from four CONNECTors which are managed (synthesized and hosted) by (E_1, E_2, E_3, E_4, E_5, E_7). Each actor can also define its trust

graph (Local trust view). For instance, A assumes that E_1 provides Co_1 as one CONNECTor, E_1 assumes that Co_1 is the result of the composition of three CONNECTors, while Co_1 is effectively composed by four CONNECTors.

In the next section we introduce the bootstrapping and the assessment process which is performed through the defined CONNECT trust graph.

6.3 Trust Bootstrapping and Assessment

Thanks to the CONNECT trust model, in this section we introduce the assessment process of the CONNECTors and the Enablers E. Indeed, CONNECTors are assessed by their synthesized Enablers E with a trust recommendation value, which results from (i) the trust on the CONNECTor based on previous deployment and also from (ii) the trustworthiness of all the Enablers E that synthesize and host this(these) provided CONNECTor(s). The assessment of the Enablers E is performed by a decentralized reputation mechanism. The trust reputation of each Enabler is managed by other Enablers and is computed with its trustors feedbacks.

Furthermore, the assessment process must fairly compute/update the trustworthiness of each CONNECT actor before/after each CONNECTor deployment. Thus, the CONNECT trust model deals with two parameters: (i) The degree of involvement (i.e., responsibility) of each Enabler in the process of synthesizing and running CONNECTors and (ii) the recommendation value that is given by each Enabler for its contribution to the CONNECTor. Indeed, we consider that each Enabler must be rewarded or penalized proportionately to both its involvement and the value of its given recommendation. Therefore, in course of time, the CONNECT trust model will be able to identify trustworthy Enablers and hence will provide more efficient and relevant CONNECTors.

However, after a while, the CONNECT system will mostly solicit Enablers with a high reputation (i.e., good history). This will preclude newcomers (without history) by making their participation to the running system very difficult or even impossible. Thus, in order to allow CONNECT architecture to evolve with new capable Enablers, we endow the CONNECT trust model with an incentive Risk-based property, in which, Enablers that have a high reputation are pushed to reduce their recommendation values in order to maintain their reputation (i.e., minimize the penalization). Thus, by adopting this behavior, everyone wins. On the one hand, the entities with high reputation will save their position, and on the other hand, this incentive behavior will boost the bootstrapping phase by giving the opportunity to new legitimate Enablers to be considered by the CONNECT architecture.

In this section, we introduce the CONNECT trust assessment process, which is implemented by: managing Enabler's Reputation (Section 6.3.1), managing CONNECT's actors trust relations (Section 6.3.2), and assessing Enabler's recommendations (Section 6.3.3). We also illustrate in Section 6.3.4 how CONNECT trust relations and Enablers' reputation are fairly updated after receiving CONNECTor monitoring feedbacks. However, in this chapter, we assess only the synthesizing part of the 1:N trust relation, namely, (1↑N), whereas the running part (1↓N) will be treated in future work.

6.3.1 Reputation management

The CONNECT trust model defines for each Enabler E a reputation value, which is represented in CoTG graph as vertex's label (see Figure 6.6). The CONNECT trust model implements a decentralized reputation mechanism. Thus, the reputation of each Enabler E_i is managed by other Enablers, which have been selected with a distributed hash table, such as CAN [167] or Chord [186]. We use three hash functions ($H_1, H_2\ and\ H_3$) to replicate the reputation of each Enabler. This will prevent against malicious Enablers and also keep the system more resistant to inherent dynamic network behavior, namely, Enablers that unexpectedly leave or disconnect. Thus, due to the decentralization aspect of the CONNECT reputation mechanism, each Enabler has to define its Reputation set and its Reputation vector as follows:

Definition 6.1 ($\mathbb{R}_i$: The Reputation Set) *The $\mathbb{R}_i$ set gathers all Enablers for which E_i manages their reputation. This is described as follows:*

$\mathbb{R}_i = \{E_j | E_i = H_1(E_j) \vee E_i = H_3(E_j) \vee E_i = H_3(E_j)\}$

Definition 6.2 (R: The Reputation Vector) *The Reputation Vector R of the Enabler E_i stores the reputation values (bounded between 0 and 1) of all Enablers that are included in E_i's Reputation set. For the Vector initialization, we distinguish primary Enablers from the other Enablers. In fact, there are some Enablers that are known to be trustworthy. The reputation of these Enablers start from a fixed high reputation threshold noted h^0 (e.g., If E_j is a primary Enabler, then $R[E_j] = h^0$). Whereas the other Enablers start their reputation from a low reputation threshold noted l^0 (e.g., If E_j is not a primary Enabler, then $R[E_j] = l^0$)*

The evolution of the reputation vector is performed with a monitoring feedback and hence will be detailed in Section 6.3.4.

6.3.2 Direct trust relation management

As defined in section 6.2.1, 1:1 trust relation is used between Enablers E and between Enabler E and Synthesized CONNECTor Co. Whereas N:1 trust relation is established between unknown Enablers E and between Enabler A or Enabler E. Thus, we define, in the following, trust structures (i.e., Sets and Vectors) which are maintained by each Enabler in order to store the assessment values of direct trust relations:

1. **E to E trust relation:**

   Definition 6.3 ($\mathbb{E}_i$: The CONNECTor Enabler's Trust Set) *Each Enabler E_i has to define its trust set. This trust set contains trusted Enablers, i.e., all CONNECTor Enablers in which E_i trusts personally. This set is described as follows:*

   $\mathbb{E}_i = \{E_j \in \mathbb{E} | E_i \xrightarrow{1:1} E_j\}$

   Definition 6.4 ($r\mathbb{E}_i$: The CONNECTor Enabler's Trust Reputation Set) *Each Enabler E_i has to define its trust reputation set. This trust set contains Enablers in which E_i trusts their reputation. This set is described as follows:*

   $r\mathbb{E}_i = \{E_j \in \mathbb{E} | E_i \xrightarrow{N:1} E_j\}$

   Definition 6.5 (eT_{E_i}: The CONNECTor Enabler trust Vector) *The CONNECTor Enabler trust Vector cT_{E_i} stores the assessment values of all $\mathbb{E}_i$'s and $r\mathbb{E}_i$'s members. The assessment of the relation $E_i \xrightarrow{1:1} E_j$ is bounded in the interval]0..1] and fixed by the Enabler E_i. This trust link represents E_i personal trust relation, which can, for example, resulted from a contract or motivated by the profit (e.g., huge storage capacity, smart video CONNECTor synthesizing, etc.), etc. On the other hand, the relation $E_i \xrightarrow{N:1} E_j$ is established, in case of E_i trusts the reputation of E_j. Thus, the assessment of this relation is defined by the trustee's reputation (i.e., $cT_{E_i}[E_j] = R[E_j]$). As for the reputation function, the evolution of this trust vector is updated from monitoring feedbacks and hence will be detailed in Section 6.3.4*

2. **E to Co trust relation:**

   Definition 6.6 ($\mathbb{C}_i$: The CONNECTor Composition Set) *Each CONNECTor can be composed of more than one CONNECTor. Thus, we define $\mathbb{C}_i$ as the set that indexes the composition of the CONNECTor Co_i*

   Definition 6.7 ($\mathbb{E}_i^j$: The Synthesis Composition Set) *This set gathers all trusted Enablers that help E_i to synthesize the CONNECTor Co_j, as follows:*

   $\mathbb{E}_i^j = \{e \in \mathbb{E}_i | \forall co \in \mathbb{C}_j, \exists e / e = \uparrow co\}$

   Definition 6.8 ($c\mathbb{E}_i$: The CONNECTor Enabler Synthesis Set) *The $\mathbb{E}_i$ set gathers all CONNECTor that have been synthesized by the Enabler E_i. This set is described as follows:*

   $c\mathbb{E}_i = \{co \in \mathbb{C} | E_i = \uparrow co\}$

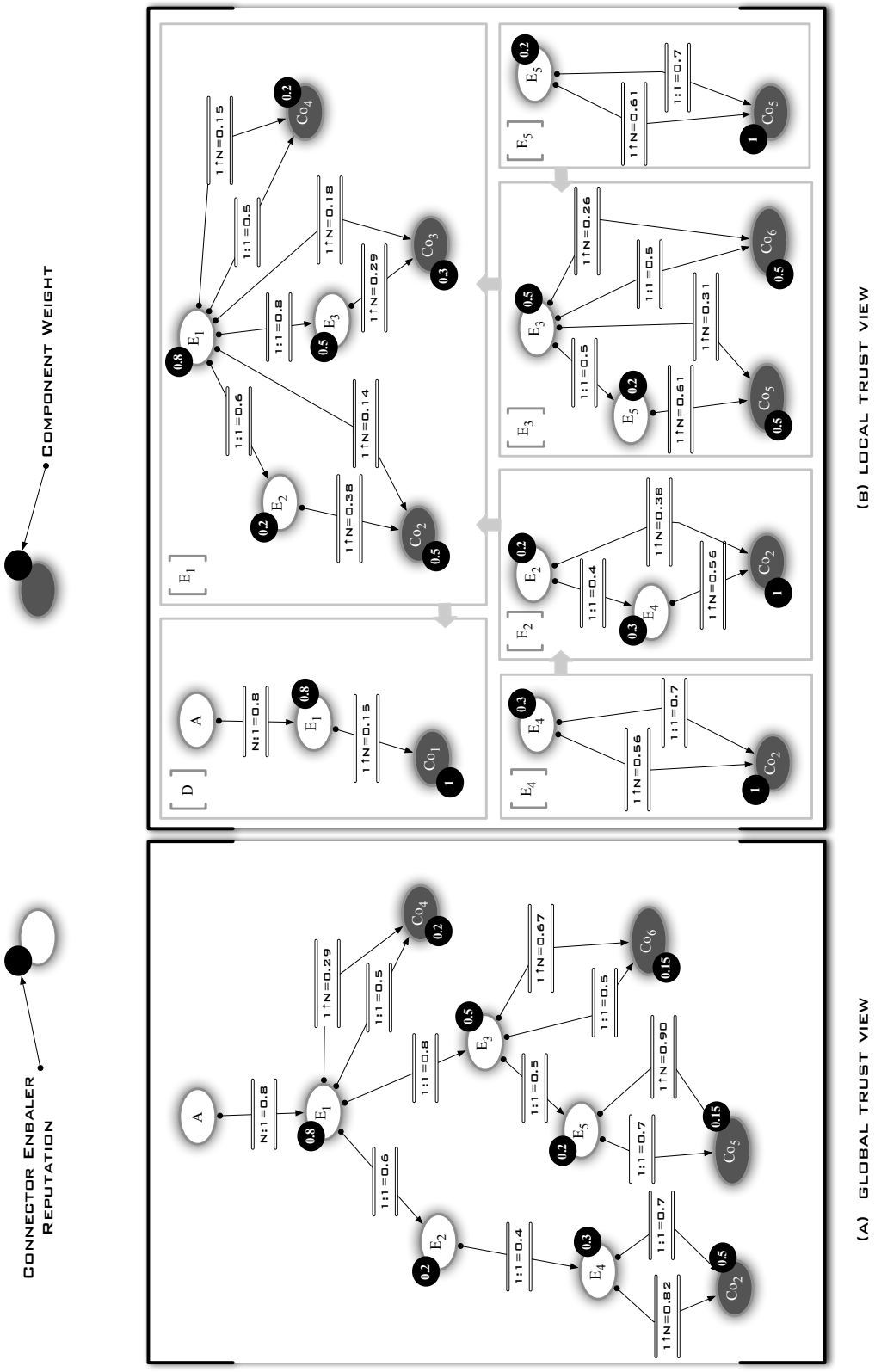


Figure 6.6: Example of trust recommendation assessment

Definition 6.9 (cT_{E_i}: The CONNECTor Trust Vector) *The* CONNECT*or Trust Vector cT_{E_i} stores the assessment values of E_i's synthesized* CONNECT*ors. Each evaluation represents the past* CONNECT*or behavior. We assume that the assessment of this trust relation is initialized, at the first* CONNECT*or deployment, with the average trust 0.5 (i.e., $cT_{E_i}(Co) = 0.5$). Then, after each successful/failure deployment this trust value is increased/decreased (see Section 6.3.4).*

3. **A to E trust relation:**

Definition 6.10 ($\mathbb{A}$: The Access Enabler's Trust set) *The Enabler A has to define its trust set, which contains the Enablers E that have a high reputation value (i.e., greater than the high reputation threshold h^0), as follows:*

$\mathbb{A} = \{E_i \in \mathbb{E} | R[E_i] \geq h^0\}$

In fact, we assume that A establish a trust relationship only with the Enablers E that have a high reputation. We define this restriction, because the Enabler A delegates to Enablers E the task to synthesize and to run CONNECT*ors. Thus, obviously the Enabler A asks only reliable Enabler (i.e., Enablers with high reputation).*

Example: From the Figure 6.5 we can find the following sets:

- $\mathbb{E} = \{E_1, E_2, E_3, E_4, E_5, E_6, E_7\}$, $\mathbb{D} = \{E_1, E_6\}$
- $\mathbb{E}_1 = \{E_1, E_2, E_3, E_7\}$
- $\mathbb{C}_1 = \{C_2, C_3, C_4\}$, $\mathbb{C}_3 = \{C_5, C_6\}$
- $\mathbb{E}_1^1 = \{E_2, E_2, E_1\}$,

6.3.3 Recommendation assessment

The main objective of the CONNECT trust model is to define an assessment process that allows Enabler (E) to assess CONNECTor and hence provides its recommendation. Thus, we define the trust Recommendation function (Rec) to assess $1 \uparrow N$ trust relations.

As illustrated in Figure 6.7, this assessment is performed by composing ($\overset{R}{\bigotimes}$) and aggregating ($\overset{w}{\bigoplus}$) trust relations as follows:

$$Rec(E_i, Co_j) = \begin{cases} \overset{w}{\underset{co \in \mathbb{C}_j}{\bigoplus}} Rec(E_i, co) & \text{if } |\mathbb{C}_j| > 1 \\ \overset{R[E_i]}{\bigotimes} cT_{E_i}[Co_j] & \text{if } (|\mathbb{C}_j| = 1 \wedge E_i = \uparrow Co_j) \\ T_{E_i}[\uparrow Co_j] \overset{R[E_i]}{\bigotimes} Rec(\uparrow Co_j, Co_j) & \text{if } (|\mathbb{C}_j| = 1 \wedge E_i \neq \uparrow Co_j) \end{cases} \quad (6.1)$$

To compute the trust composition, we define two operators:

- The aggregation operator: $\overset{w}{\bigoplus}$.
- The composition operator: $\overset{R^t}{\bigotimes}$.

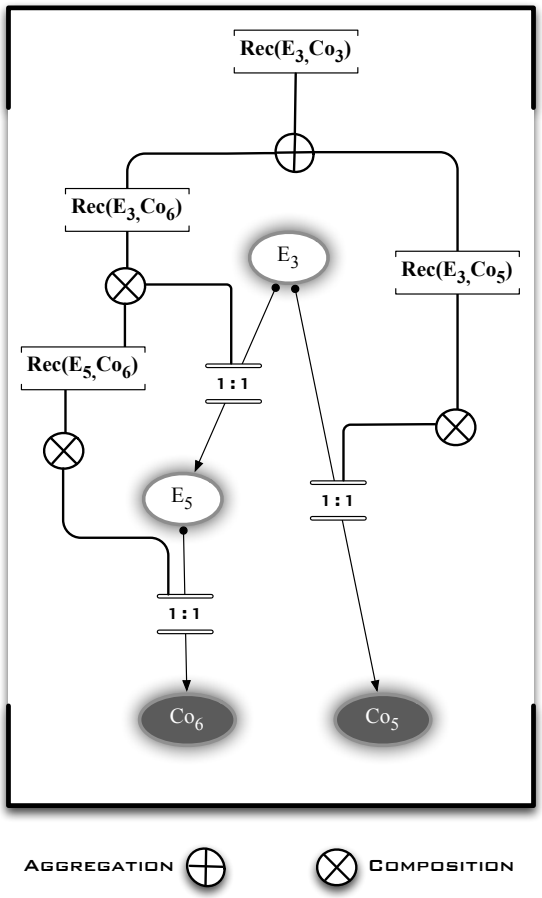


Figure 6.7: Trust aggregation and composition

Aggregation Operator:

The aggregation operator is defined to aggregate all sub-CONNECTors' trust values. For example: as illustrated in Figure 6.3, knowing that, E_3 trusts the CONNECTor Co_5 and the CONNECTor Co_6, the aggregation operator allows E_3 to compute: how much it can trust the CONNECTor Co_3, which is composed from Co_5 and Co_6.

We define, in the CONNECT trust model, the aggregation operator as the sum of all sub-CONNECTors' weighted trust value, as following:

$$\bigoplus_{co \in \mathbb{C}_j}^{w} Rec(E_i, co) = \sum_{co \in \mathbb{C}_j} W(Co_j, co) \times Rec(E_i, co). \tag{6.2}$$

Where W is the CONNECTor weight function

In order to perform a fair computation of the trust aggregation of a composition of CONNECTors, we must assess the involvement degree of each one in the whole composition.

Definition 6.11 (CONNECTor Weight Function)
We define the function $W(Co_i, Co_j)$ which return the weight (a float value bounded in the interval [0..1]) of the CONNECT*or Co_j in the composition of the* CONNECT*or Co_i, where:*

$$W(Co_i, Co_j) = 0 \Rightarrow Co_j \notin \mathbb{C}_i$$

$$\forall Co_i \in \mathbb{C}, W(Co_i, Co_i) = 1,$$

$$\forall Co_i \in \mathbb{C}, \sum_{co \in \mathbb{C}_i} W(Co_i, co) = 1$$

When an Enabler E receives a synthesized CONNECT*or request, the weight of the whole* CONNECT*or is equal to 1. Whereas if the* CONNECT*or is divided into several sub-*CONNECT*or, the Enabler E weights each sub-*CONNECT*or according to its involvement degree on the whole composition (assessing the involvement degree is not detailed in this chapter). The weight of each sub-*CONNECT*or is given between 0 and 1, and the sum of all sub-*CONNECT*ors is equal to the weight of the requested* CONNECT*or.*

This value is represented in the $CoTG$ graph by labeling CONNECTors. As illustrated In Figure 6.6, the CONNECTor Co_1 is decomposed into four CONNECTors $\{Co_2, Co_5, Co_6, Co_4\}$ which are weighted respectively by $\{0.5, 0.15, 0.15, 0.2\}$. The weight of each sub-CONNECTor is defined by the Enablers which decide to split Co_1, namely, E_1 and E_3. However, through the local view each Enabler weights only its requested CONNECTor(s). For instance: E_2 and E_4 are requested for Co_2, thus for these Enablers the CONNECTor is labeled with $W(Co_2, Co_2) = W(Co_2, Co_2) = 1$, whereas for E_1, the CONNECTor Co_2 is labeled with $W(Co_1, Co_2) = 0.5$ because E_1 has been requested for Co_1.

Composition Operator:

The composition operator is defined to evaluate the trust transitivity. This assessment is often performed by two different approaches: (i) a pessimistic approach in which the multiplication of all composed trust values, that are bounded between 0 and 1, is used or (ii) a more moderate one, which instead of the multiplication, implements the average of all composed trust values. For example: let us consider three Enablers E_1, E_2, E_4 where $T^{(t)}(E_1, E_2) = 0.6$ and $T^{(t)}(E_2, E_4) = 0.5$. Thus, if E_1 is pessimistic, it can compute this composition by the multiplication $T^{(t)}(E_1, E_4) = 0.6 \times 0.5 = 0.3$, while if E_1 adopts a moderate behavior, it can use the average and obtain a higher trust value as $T^{(t)}(E_1, E_4) = (0.6 + 0.5)/2 = 0.55$.

The CONNECT trust model respects an Incentive property, in which the behavior of each Enabler (pessimistic or moderate) is dictated by its reputation. Thus, Enablers with high reputation may adopt a pessimistic behavior in order to maintain their reputation (i.e., proportionately decreasing their Recommendation values). Whereas Enablers with low reputation has to adopt a more moderate behavior (i.e., proportionately increasing their Recommendation values) in order to be highly rewarded in case of successful deployment (see Section 6.3.4).

Thus, we define the composition operator as follows:

$$\begin{aligned} \overset{r}{\bigotimes} T_1 &= I_r(T_1) \\ T_1 \overset{r}{\bigotimes} T_2 &= I_r(AVG(T_1, T_2)) \end{aligned} \tag{6.3}$$

Where I is the incentive function,
r is a reputation values and
T_1 and T_2 are trust values which are returned by the vectors eT or cT
or computed by the Rec function.

Definition 6.12 (I_r The Incentive Function) *The incentive function is able to express, according to the given reputation value r, three kinds of behavior: pessimistic, moderate or optimistic. In fact, as defined in Figure 6.8, the function I takes a moderate trust evaluation (the average value) as an input and provides a pessimistic output (by reducing the input), if the reputation r is in the interval $]0.5, 1]$. On the contrary, the function I returns an optimistic output (by increasing the input), if the reputation r is in the interval $]0, 0.5]$.*

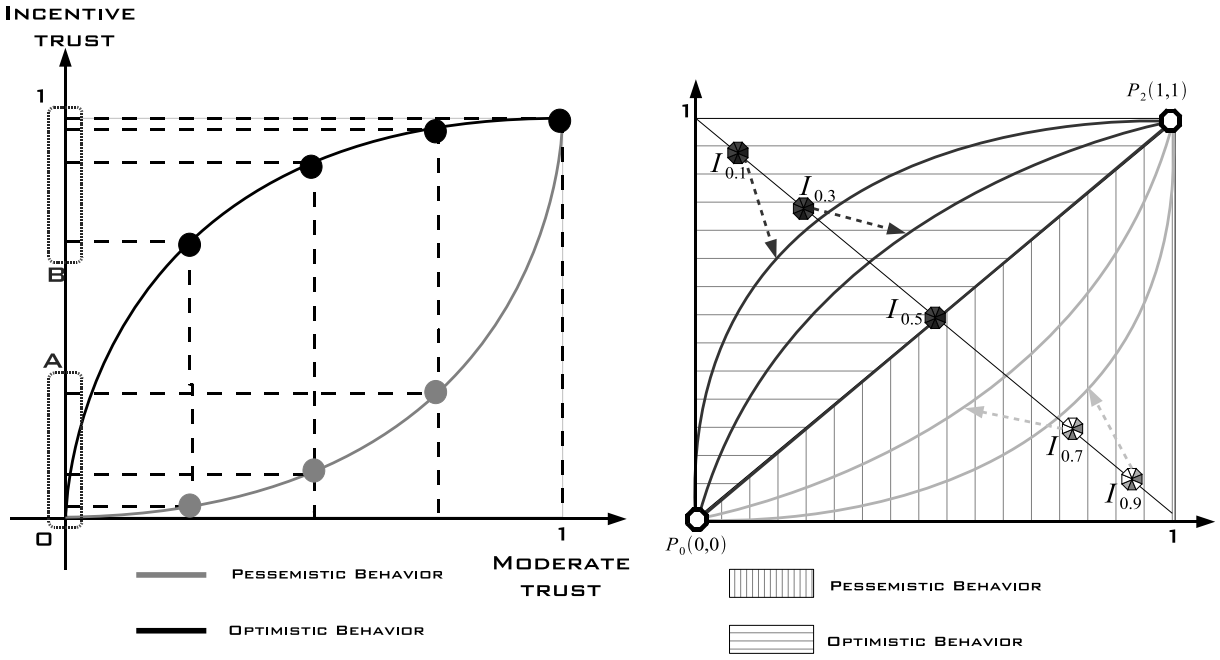


Figure 6.8: Incentive function

To implement the I function we use the following *Behavior function* BV that is defined in [173] to shape our functions:

Definition 6.13 (BV_{l,h_x,h_y}: The Behavior Function) *The BV function is the Cartesian form of a quadratic Bezier curve. Thus, the BV function is able to draw smooth curves which are achieved through three points P_0, P_1 and P_2, starting at P_0 going towards P_1 and terminating at P_2, where:*

- $P_0(0,0)$ *is the origin point.*
- $P_2(h_x, h_y)$ *is called the threshold point, where the h_x and h_y represent respectively the abscissa and the ordinate thresholds.*
- $P_1(b_x, b_y)$ *defines the behavior point. This point is guided by a given variable 'l' inside the rectangle that is defined by P_0 and P_2, through the second diagonal (e.g., if $l = 0$ then P_1 has the coordinates $(h_x, 0)$ and if $l = 1$, then P_1 has the coordinates $(0, h_y)$).*

Thus, by computing the Cartesian function from the Bezier parametric format and by fixing the position of the point P_1 according to the value of l, we obtain the following function:

$$\begin{array}{rcl} BV : [0, h_x] & \longrightarrow & [0, h_y] \\ X & \longrightarrow & Y \end{array}$$

$$BV_{l,h_x,h_y}(X) = \begin{cases} \frac{(h_y - 2b_y)}{4b_x^2} X^2 + \frac{b_y}{b_x} X & si\ (h_x - 2b_x = 0) \\ (h_y - 2b_y)(\propto (X))^2 + 2b_y \propto (X), & si\ (h_x - 2b_x \neq 0) \end{cases} \tag{6.4}$$

$$Where \begin{cases} \propto (X) = \frac{-b_x + \sqrt{{b_x}^2 - 2b_x * X + h_x * X}}{h_x - 2b_x} \\ (0 \leq b_x \leq h_x) \wedge (h_x > 0) \wedge (0 \leq l \leq 1) \\ b_x = l \times h_x \wedge b_y = h_y \times (1 - l) \end{cases}$$

Thus, the incentive function I_r of the CONNECT trust model is defined from the BV function as follows:

$$I_r(x) = BV_{f(r),1,1}(x) \tag{6.5}$$

where, $l = f(r)$ is a a monotonic increasing function.

For example if $l = r$ the incentive function shape pessimistic and optimistic behaviors, whereas, if $l = \frac{r}{2}$ the incentive function will only balance between moderated and pessimistic behavior.

Example: In Figure 6.9, the Enablers E_2 and E_3 synthesize the CONNECTor Co that has been requested by E_1. In this case, E_1 has to assess its recommendation regarding each synthesized CONNECTor. As illustrated in Figure 6.9, only by adopting an incentive behavior (where $l = \frac{r}{2}$) E_1 trusts E_2 to synthesize the CONNECTor Co, whereas moderated or pessimistic behavior selects E_3 mainly due to its high reputation. Thus, in this example we illustrate the advantage of the incentive policy in helping newcomers in their bootstrapping phase.

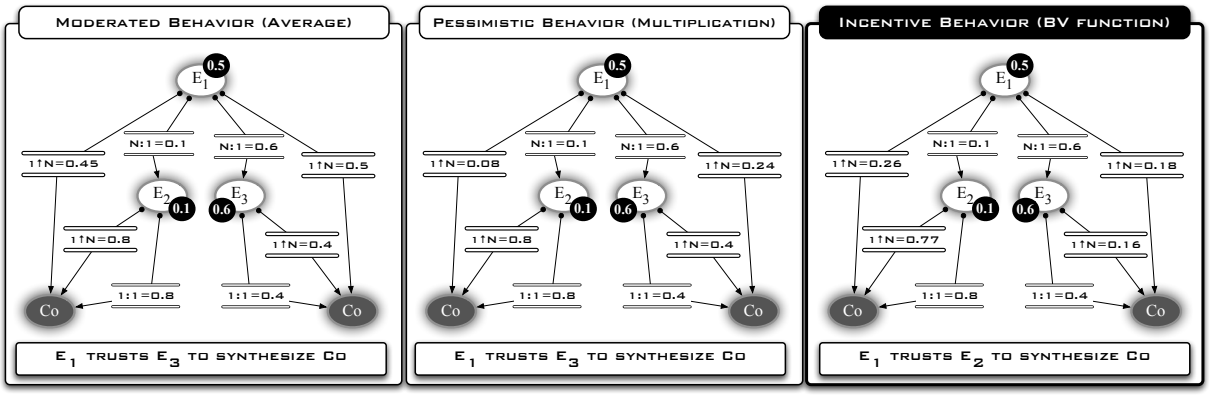


Figure 6.9: Incentive policy for bootstrapping newcomers

In Figure 6.6, we provide another example which shows the whole trust assessment process. In this example the CONNECTor Co_1 is recommended by Enabler E_1 to Enabler A with trust value 0.15. Thus, from this figure, we present in the following some trust Recommendation assessment steps:

$$\begin{aligned} Rec(E_1, Co_1) &= W(Co_1, Co_2) * Rec(E_1, Co_2) + \\ &\quad W(Co_1, Co_3) * Rec(E_1, Co_3) + \\ &\quad W(Co_1, Co_4) * Rec(E_1, Co_4) \\ &= 0.5 \times 0.14 + 0.3 \times 0.18 + 0.3 \times 0.15 = 0.15 \end{aligned}$$

$$Rec(E_4, Co_2) = \overset{R[E_4]}{\bigotimes} cT_{E_4}[Co_2] = 0.7 \overset{0.3}{\bigotimes} 0.7 = 0.56$$

$$Rec(E_1, Co_2) = T_{E_1}[E_2] \overset{R[E_1]}{\bigotimes} Rec(E_2, Co_2) = 0.38 \overset{0.8}{\bigotimes} 0.6 = 0.14$$

6.3.4 Trust feedbacks management

One of the CONNECT monitor Enabler (see Chapter 7) tasks is to monitor the CONNECTed System in order to know if deployed CONNECTors work as expected or not. Thus, according to the given monitoring feedbacks, each Enabler updates its trust relations which result into updating its trustees' reputation.

For the CONNECT trust model three feedbacks messages are defined:

1. $Co\_Feedback[Co_n, Co_m, State]$ which gives the deployment state (success or fail) of the CONNECTor Co_m that has been used to compose the CONNECTor Co_n.

2. $Rep\_Feedback[E_i, E_j, Diff]$ which gives the difference (Diff) between the new and old trust relation of E_i to E_j.

3. $High\_Rep\_Feedback[E_k, Rep[E_k]]$ which is used to inform the Enabler A about any modification affecting the reputation of its trusted Enablers.

Thus, in the following, we introduce the distributed algorithm that is performed by each Enabler E in order to update the evaluation of its direct trust relations (lines 01-43) as well as the reputation (lines 44-53) of its managed Enablers:

01 **if** E_i **receives** $Co\_Feedback[Co_n, Co_m, State]$ **then –>** // **Updating direct trust relation**//
02 BEGIN
03 -**For each (**$co \in \mathbb{C}_m$**) do**
04 -BEGIN
05 –$E_j = \uparrow co$;
06 –**If (**$E_i == E_j$**) then ——>** // **(**$E_i \xrightarrow{1:1} co$**) *trust relation*//**
07 –BEGIN
08 —If ($State == success$) then
09 —-$cT_{E_i}[co] = cT_{E_i}[co] + [\alpha \times (1 - cT_{E_i}[co])]$;
10 —Else
11 —-$cT_{E_i}[co] = cT_{E_i}[co] - [\alpha \times (cT_{E_i}[co])]$;
12 —END
13 –Else
14 —**If (**$E_j \in \mathbb{E}_i$**) then ——>** // **(**$E_i \xrightarrow{1:1} E_j$**) *trust relation*//**
15 —BEGIN
16 —-$Old = eT_{E_i}[E_j]$;
17 —-If ($State == success$) then
18 —-BEGIN
19 ——$eT_{E_i}[E_j] = eT_{E_i}[E_j] + [\alpha \times (1 - eT_{E_i}[E_j]) \times Rec(E_j, co) \times W(Co_n, co)]$;
20 ——Send ($Co\_Feedback[Co_n, co, State = success]$) to E_j;
21 —-Else
22 ——$eT_{E_i}[E_j] = eT_{E_i}[E_j] - [\alpha \times (eT_{E_i}[E_j]) \times Rec(E_j, co) \times W(Co_n, co)]$;
23 ——Send ($Co\_Feedback[Co_n, co, State = fail]$) to E_j;
24 —-END
25 —-$Diff = eT_{E_i}[E_j] - Old$;
26 —-Send [$Rep\_Feedback(E_i, E_j, Diff)$] to $H_1(E_j)$, $H_2(E_j)$ and $H_3(E_j)$;
27 —Else
28 —-**If (**$E_j \in r\mathbb{E}_i$**) then ——>** // **(**$E_i \xrightarrow{N:1} E_j$**) *trust relation*//**
29 —-BEGIN
30 ——If ($State == success$) then
31 ——BEGIN
32 ——$Diff = \alpha \times (1 - R[E_j]) \times Rec(E_j, co) \times W(Co_n, co)$
33 ——Send ($Co\_Feedback[Co_n, co, State = success]$) to E_j;
34 ——Else
35 ——$Diff = -\alpha \times R[E_j] \times Rec(E_j, co) \times W(Co_n, co)$
36 ——Send ($Co\_Feedback[Co_n, co, State = fail]$) to E_j;
37 ——END
38 ——Send [$Rep\_Feedback[E_i, E_j, Diff]$] to $H_1(E_j)$, $H_2(E_j)$ and $H_3(E_j)$;
39 —-END
40 —END
41 –END
42 -END
43 END

44 **If** E_i **receives** $Rep\_Feedback[E_j, E_k, Diff]$ **then —>** //**Updating trust reputation**//
45 BEGIN
46 -$Old = R_{E_i}[E_k]$;
47 -If $Diff > 0$ then

48–$R[E_k] = \alpha * R[E_k] + (1 - \alpha) \times Min(1, R[E_k] + Rep\_Feedback(E_j, E_k)) * R[E_j])$;
49 -Else
50–$R[E_k] = \alpha * R[E_k] + (1 - \alpha) \times Max(0, R[E_k] + Rep\_Feedback(E_j, E_k)) * R[E_j])$;
51 -If (($R[E_k] > h^0$ and $Old < h^0$) or ($R[E_k] < h^0$ and $Old > h^0$)) then
52 –Sends $High\_Rep\_Feedback[E_k, Rep[E_k]]$ to A;
53 END

We have introduced the α parameter ($\alpha \in]0..1]$) that defines the amount of the impact on the past acquired trust value, due to each new action. Thus, each Enabler E increases or decreases its past evaluation of its synthesized CONNECTors proportionately to α (Lines 09, 11). However, as aforementioned, in addition of α, the relation that links Enablers E (Lines 14-40) must be updated fairly by considering: the CONNECTor recommendation value that is returned by the corresponding trustee (E_j, line 05) and (ii) the weight of that CONNECTor in the whole composition. Thus, each Enabler updates its trust relations proportionately to all later values (Lines 19, 22, 32, 35), and then propagates CONNECTor deployment state to the involved trustee E_j (Lines 20, 23, 33, 36). Finally, in order to update the reputation of that trustee, the current Enabler sends a $Rep\_Feedback$ message to the three Enablers that are responsible to manage the reputation of that trustee by using the three defined hash functions (Lines 26, 38).

On the other hand, when the Enabler E receives a $Rep\_Feedback$ message, it updates (Line 48, 50) the reputation of the given trustee (E_k) proportionately (i) to α, (ii) to the amount of the trustor's (E_j) given update ($Diff$) and (iii) to the trustor's reputation ($R[E_j]$). Then, if the new trustee's reputation becomes bigger than the high reputation threshold (h^0) or decrease under h^0 threshold, the current Enabler informs (line 52) the Enabler A to respectively adds or removes the trustee from its trust set $\mathbb{A}$.

6.4 Discussion and Future Directions

In this chapter, we have focused on identifying the building blocks of the CONNECT trust model. We have defined a CONNECT trust graph to represent all the relevant trust relations among CONNECT actors. Then, through this trust graph, we provide the CONNECT trust assessment process. The assessment process is performed in a distributed manner where Enablers E adopt an incentive behavior by adjusting their risk factor according to their reputation. We have also managed monitoring feedbacks to update direct trust relation and Enablers' reputation fairly. In order to improve the incentive policy, we plan to perform some experimentations with considering the time fading [121] in the reputation assessment process.

Furthermore, the CONNECT trust model is closely related with the CONNECT security model. The latter may use the trust values in its Security-by-contract approach (see Chapter 5), while the former may update Enablers reputation according to security feedbacks.

As part of our ongoing work, we are studying the deployment phase, especially to assess the best Running platforms according to the reputation and the capabilities of available Enablers. In order to treat this perspective, we have to define and to assess all trust relations that are related to Networked Systems.

Moreover, we will investigate with partners that are involved in the monitoring task (see Chapter 7) to define the time dimension. Indeed, some questions need to be answered, for instance: how much time is need to claim that the CONNECTor has been successfully synthesized and deployed? and also how does the CONNECT trust model manage a connection problem after a long successful deployment?

7 Monitoring

The need for sophisticated means to monitor the behaviour of software systems arose as soon as they begun to grow in size and complexity, evolving from simple, monolithic, single-threaded, locally-executed programs to large, distributed, complex systems. The CONNECT project and the new vision it advocates push the very concept of system evolution and dynamism even further, envisioning a new generation of systems that can achieve long-lived interoperability by exploiting automated mechanisms for synthesizing software CONNECTors.

The result of assembling Networked Systems – whose behaviour is inferred by run-time observation (i.e., active testing) – with CONNECTors that are built on-the-fly, is a system whose characteristics (both functional and non-functional) are exceptionally difficult to predict and to guarantee. This new breed of systems exacerbates the problems related to monitoring.

Furthermore, the role of monitoring in the CONNECT project is not only to support dependability assurance, but also to act as a core functionality that is meant to complement the other CONNECT Enablers, especially those responsible for CONNECTor synthesis and behaviour learning.

This chapter is structured as follows. Section. 7.1 presents the basic concepts related to monitoring and defines the terminology used in the chapter. Section 7.2 breaks-down a generic monitoring system into its fundamental elements, mentioning theoretical constraints and implementation issues. Based on this discussion, Section 7.3 sketches a road-map towards an integrated CONNECT monitoring framework, identifying high-level requirements and providing a preliminary discussion of possible integration with other CONNECT Enablers. Finally, Section 7.4 concludes the chapter and briefly outlines the next steps that are planned for the second year of the project.

7.1 Basic Concepts and Terminology

Before discussing monitoring, we characterize the system being monitored, which is usually referred to as the *subject system*. Unless otherwise stated, in this chapter we refer to *concurrent distributed systems*, which are the implied target of the CONNECT project and that are so commonly found nowadays in the large majority of practical applications. Besides, by adopting a reference model that accommodates concurrency and distribution, we will as well be able to capture special cases with just one process and/or one computing node.

For our purposes, the subject system can be conveniently modeled as a set of components that represent units of *sequential* computation. The communication among components can be either synchronous or asynchronous and is realized by executing the following three basic types of actions: (a) send, (b) receive, (c) rendez-vous. Action of types (a) and (b) are used to model communications realized as an asynchronous message exchange; instead, *rendez-vous* actions (c) represent synchronous message exchanges [90].

Communication actions happen at the interfaces of components and are regulated by inter-component communication protocols that are independent of the components' internals. In CONNECT, Networked Systems play the role of components and the communication protocols may be realized by CONNECTors that are automatically synthesized.

Another type of actions, the *internal actions*, describes the computation that is performed internally to the components. These actions are not visible at the components' interfaces. However, in certain applications of monitoring, observing them is as important as observing inter-component communication and therefore we need to make them visible from the outside of the component (which can be done through different techniques, as explained later in this chapter).

As we have discussed so far, the operation of a subject system is modeled in terms of actions, which are a suitable abstraction to represent the computation that takes place in the subject system, either internally to its component or at the interfaces between components.

In principle, a primitive event can be associated to the execution of each action, as its completion can be assumed to happen instantaneously and at a precise point in time.

There is a distinction between the subject of the observations (i.e., the actions) and the way they are manifested for the purposes of the observation (i.e., events): based on this distinction we can say that the fundamental concern of a monitoring system are not directly the actions themselves, but rather the events

through which they are manifested. In this view, we have no means to observe actions but through the events that are associated to them.

While obviously, events *represent* actions, an important consequence follows by their definition: from the standpoint of monitoring, actions just *happen*, whereas events are *fired* according to the decisions taken as part of the configuration of the monitoring system. Therefore, event specification is the subject of a design and configuration activity that is central to the overall setup of a monitoring system: the decision of associating an event to the execution of an action (or sequence of actions) is inherently arbitrary, and is one of the basic choices that must be taken when setting up a monitoring system. By this association, actions (i.e., the computation) become visible and manifest themselves as events.

According to most dictionaries, an event can be defined as "something that happens"[1]. According to what we stated in the previous paragraphs, our notion of event is slightly different, as we consider events as a "*representation* of something that happens".

Other works on monitoring do not define the concept of an *event* explicitly. They characterize it implicitly with the property of being associated with a *timestamp*, leaving the rest to the reader's intuition. Elsewhere, an event is also referred to as an "occurrence of interest" [134]

In addition to basic (or "primitive") events – i.e., events that correspond to the completion of an action – it is useful to define the concept of structured events (also called "composite events" in [134] or "event definitions" in [184]), i.e., events that happen when a certain combination of basic events and/or other composite events happen.

The problems of event specification and event detection has been studied extensively in the *Active Database Management Systems (ActiveDB)* [157, 7, 159]. In ActiveDBs, databases are made to react to certain situation as specified in so-called ECA-Rules (Event-Condition-Action Rules). The approach followed in ActiveDBs is to encode both the definition of what is relevant and how to react when an observation matches that definition of relevance. While we prefer not to mix the concerns of observation and reaction[2], the studies on event definition, event algebras, event composition, and composite event detection coming from ActiveDB literature are applicable to a large extent to CONNECT. In particular, we assume that primitive events are produced by the observed object (e.g., by a CONNECTor), whereas *complex* events can be defined from simpler events by using operators of a suitable *event algebra* [199].

A very useful distiction among the different types of event that may happen in a distributed system is given in [184], where events are divided in: *local, non-local, global*. Local events are produced on a single node, which means that observing them does not require addressing the problems that are related to distribution and inter-node synchronization. Non-local events, on the other hand, are (composite) events whose observation requires considering and correlating events originated from more than a single node. Global events are a special case of non-local events, and require considering *all* the nodes of a system.

The term *correlation* is often used informally to refer to the activity of identifying causal relationships among events that were observed on set of nodes. Using the notion of complex events as recalled from the ActiveDB literature, we can define correlation as the activity of recognizing complex events. The difficulty of this task can be complicated when such complex events are non-local (or global).

To conclude with, we can say that the goal of a monitoring system is twofold: (1) to observe the operation of the subject system as abstracted by the associated *events*, and (2) to support the interpretation of such observation in a way that is useful for the clients of the monitoring system.

7.2 Fundamental Elements of a Monitoring System

In this section, we concentrate on the fundamental concerns that a typical monitoring system has to address, and we identify a set of corresponding essential functions that are typically put in place to deal with those concerns. Where appropriate, we reference problems or aspects that are peculiar to the CONNECT project.

Not all real-world monitoring systems need to provide all the functions, and at the same time the literature has plenty of approaches that address different concerns jointly. However, for the sake of clarity,

[1] See e.g., the Merriam-Webster Dictionary or the Longman Dictionary of Contemporary English

[2] In CONNECT we aim to define a monitoring framework that is general enough to serve different purposes and to support different functionalities of the CONNECT infrastructure. Reactions to interesting situations are therefore demanded to the clients of the monitoring system, while the latter is responsible for efficiently performing the observations and for notifying interested clients.

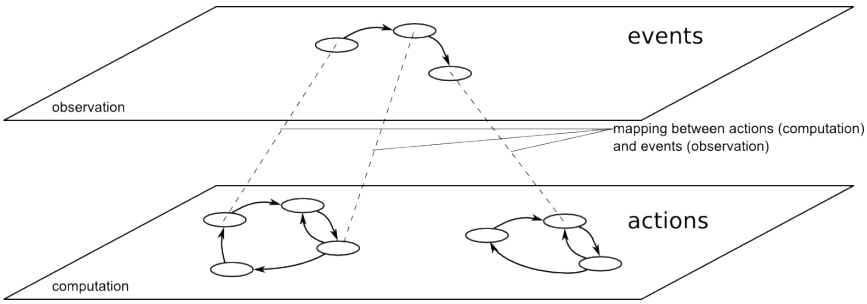


Figure 7.1: Events are an observable representation of computation steps

it is useful to describe these functions individually and to decompose the high-level problem of monitoring into smaller problems, in order to present the specific methodological and technological solutions that have been proposed to address them.

Such a break-down simplifies the classification and the comparison of existing approaches to monitoring, and partitions the design space providing guidelines for building monitoring systems that possess certain desired characteristics.

An attempt to elaborate a reference conceptual model to describe monitoring systems has been put forth by other authors before. An extensive discussion of this matter was done by Mansouri-Samani and Sloman [133], who, in turn, elaborated on the work previously done by Feldkuhn and Ericsson [74].

In the following, we build on this background, integrating it with views emerging from other works. We try to harmonize concepts and terminology used – sometimes with somewhat inconsistent meaning in the different works we examined. Also, we strive to give a fresher view on some of the problems that the construction a monitoring system entails, especially in the light of new technologies and trends that have emerged in the last decade.

7.2.1 Defining "monitoring"

Monitoring has been defined as:

> the process of dynamic collection, interpretation, and presentation of information concerning objects or software processes under scrutiny [104].

This definition suggests a few remarks.

Firstly, it is defines monitoring as a *dynamic* (as opposed to static) activity. However, the word "dynamic" has far too many meanings that are easily confused; therefore, we prefer to say that monitoring is a *run-time* activity, to stress that it is inherently concerned with the *execution phase* of a system, as opposed to activities that are carried out in the development/coding phase. In this respect, it is common to find works in the literature that use the expression "run-time monitoring" to indicate what we refer to as "on-line monitoring", i.e., monitoring that is done on production systems (i.e., *in-the-field*), rather than in a testing environment (i.e., in the lab). It is important to stress that in the CONNECT project monitoring is used both in-the-lab (e.g., for active testing approaches applied to behaviour learning) and in-the-field. However, the latter is the topic of our study, whereas the former is not in the scope of WP5, although it is addressed in WP4 as part of the behaviour learning activities. In both cases (execution in-the-field and execution in-the-lab), the subject system *is running*, so in fact there is no such thing as "non-run-time" monitoring.

Secondly, the definition of [104] immediately identifies the monitoring process as a composition of several different activities, namely collection, interpretation, and presentation. Each of these activities is meant to address a specific problem and may use dedicated techniques.

Finally, the definition stresses the notion of monitoring as a means for observation, as opposed to monitoring as a means for management/control of systems, which is not uncommon in the literature, but which may cause some confusion and makes it harder to map the field of research and practice of monitoring. For example, the terminology used in [133] is somewhat biased towards SNMP terminology (e.g., the entity that is the subject to observation is called "managed object"), and consequently the aspects

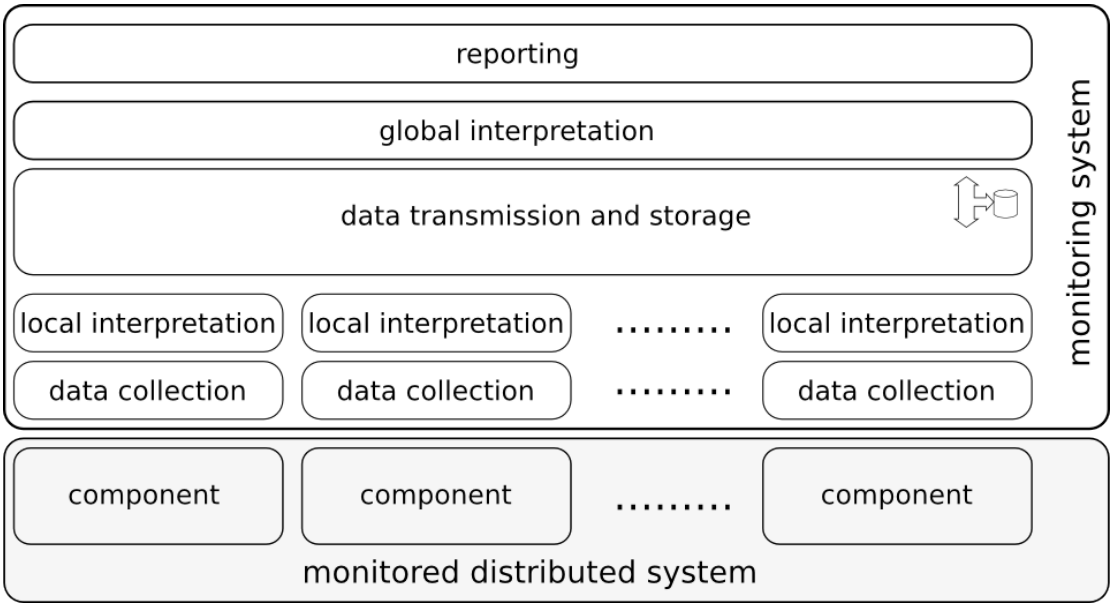


Figure 7.2: Architectural elements of a monitoring system

of observation and control may appear sometimes mixed. For this reason, we prefer to talk about an "observable object" as *a component whose behaviour can be observed by a monitoring system*. This is a rephrasing of the definition of "managed object" of [133]: it is worth remarking that although the wording is only slighly different, the meaning is substantially changed. Undoubtedly, in many practical cases, one may want observable objects to be controllable, so that observations can be used for management; however this concerns the applications or uses of monitoring, rather than monitoring *per se*.

7.2.2 Architectural elements of monitoring systems

This section concentrates on detailing each of the basic activities that a generic monitoring system carries out (see Figure 7.2).

Raw data collection The lowest layer of a monitoring framework is realized by a set of probes whose goal is to fire a primitive event whenever certain computation steps (actions) are performed. This is done locally on a per-node basis. This activity requires "attaching" a piece of software to the target observed entity in order to make the computation observable (i.e., to generate events from corresponding actions); from a technical viewpoint, it can be performed according to two styles [90]:

- *Inspect/instrument:* this style of data collection is implemented by adding code to the subject system. Monitoring actions are inserted in the program to be monitored in order to emit an event when the control flow reaches a certain spot in the execution. Technically speaking, this can be realized in several ways, spanning different forms of code instrumentation at different levels of abstraction. In general, code instrumentation techniques include statements in the original source code of the program to be monitored. In the simplest form, these statement produce an event meaning that the execution has reached a specific point in the program. The output can be guarded by a condition, whereby it is possible to refine the event definition and to emit an event only when the guard condition is satisfied (which is somehow mix of data collection and filtering). Different concrete implementation techniques are possible, most of which rely on adopting a certain underlying technological infrastructure (e.g., Java or .NET). In the literature, approaches can be found that are based on code-instrumentation [83], byte-code instrumentation [65, 33], aspect-oriented programming [152, 187], dynamic instrumentation of running binaries such as PIN[3]. Also relevant to data

[3] http://pin.webhop.org

collection is the support given by operating system facilities such as Dprobes [142], available for the Linux kernel and based on a similar idea implemented in Dtrace [139] for SUN's Solaris operating system. All the approaches based on inspection share to some extent the characteristic of being intrusive, so they can be applied only when the source code/bytecode is available for modification[4].

- *Intercept:* when adopting this style, data collection is achieved through a proxy-like probe that is put on the wire and snoops interactions, as in [22]. Although this approach may not be as flexible as the previous, it has the advantage of being non-intrusive, therefore it is well suited in other contexts where the control over the system is distributed/partioned across several organizations.

A special (very common) case is represented by those systems that provide built-in tracing/logging capabilities. The module of such systems that is dedicated to emitting events to a memory buffer or to a logfile fits well in our definition of probe. Despite the presence of native built-in logging facilities, additional instrumentation or proxy-based data collection can be added when necessary, i.e., when we are interested in events other than those natively emitted or when more flexibility is demanded, e.g., to accommodate application-specific or complex events.

Orthogonally to the instrument/intercept criterion, other criteria to classify data collection techniques can be adopted. For example, the techniques for collecting monitoring data can also be distinguished according to whether the collection is based on sampling or complete executions are observed. Most sampling approaches sample on a time basis; however certain (composite) events may go undetected if some of the constituent events are discarded by the sampling. A possible way to address this problem is by sampling in space, rather than in time, i.e., by alternating the processes (or components) that are chosen as the target of monitoring, in such a way that, locally to the target, the observation is complete and no event is discarded.

Another classification can be done based on the level of abstraction at which the data collection is performed: hardware-level, instruction-level, process-level, network-level, application-level.

Local interpretation The process of local interpretation is concerned with making sense (locally) of the information extracted by probes. This means giving an interpretation to the raw data collected by probes by applying a filter that extracts interesting sequences of events out of the raw string of events recorded by the probes. (according to a specified set of criteria)

In practical implementations, data collection and filtering can overlap to some extent: a sort of rough preliminary filtering is done if the events emitted in the data collection step are not just the result of reaching a given point in the execution but also of some other logic or processing embedded in the probe.

Concrete examples of local interpretation tools have been developed as open-source projects: SEC[5], logwatch[6], logsurfer[7], swatch[8].

Data transmission and storage The relevant events that are revealed locally need to be collected at one or more sites where they can be aggregated with analogous data coming from other nodes. Transmission may occur immediately, to reduce detection latency, or may be delayed, using buffering, e.g., to cope with network congestion (at the expense of memory occupation or CPU cycles if compression is used to mitigate mem occupation). If adaptive mechanisms of this or other kinds are to be enforced, it is necessary to devise a framework to analyze and reason about the impact of trade-offs and configuration settings at the data transmission level. In CONNECT, this type of reasoning is to be performed on-line. Data-transmission strategies may also interact with the local interpretation module, in order to prioritize data and possibly to discard/compress (even in a lossy manner) some information, on a semantic basis (e.g., to avoid transmitting a piece of information that is implied by another piece of information that has already been sent). Lossy mechanism should however retain information about the amount of information that is discarded, so that later on clients can use this rate to weigh their reliance on the output of monitoring.

[4] I.e., the actual code/bytecode of the *deployed target system* must be available for modification. Open-source does not suffice.
[5] http://simple-evcorr.sourceforge.net/
[6] http://www.logwatch.org/
[7] http://www.crypt.gen.nz/logsurfer/
[8] http://swatch.sourceforge.net/

Global interpretation In a distributed setting, data collected from different nodes must be gathered eventually at one or multiple aggregation points, where it is processed by centralized filters in order to recognized non-local complex events. The goal is to make sense globally – i.e., at an aggregated level – of the information transferred from distributed nodes. This means giving a global interpretation to the data collected by local probes. The process is similar to that carried out locally, except that events and measurements coming from multiple sources are merged into one single stream for interpretation. Depending on the system structure, the node (or the nodes) where the aggregation is performed may have less strict constraints in term of resource consumption. Architectures with more than one level of aggregation are possible, and are usually adopted when enhanced scalability is necessary [138].

When moving from local to global observation, observability issues must be taken into account. Suitable timestamping and synchronization facilities must be used as appropriate.

Reporting In our view, monitoring systems should be open-ended, in that they should be independent of the particular purpose for which they are employed. In CONNECT, this principle will be pursued by devising a core monitoring infrastructure that offers monitoring capabilities through a well-defined interface. The design of this interface will include a special-purpose language to define monitoring goals, spanning both functional and non-functional monitoring tasks. The information provided as the output of monitoring can be used for a variety of purposes and should be presented in a way that is meaningful to the clients of the monitoring system. A client can be a piece of software itself or a human. In both cases the results of the final interpretation phase must undergo an elaboration in order to express output data in a suitable format. In the former case, this format should be machine-readable; in the latter, it must be shown either as a textual report or it may use interactive GUIs, graphics, animations and so on.

General implementation issues related to theoretical constraints

There exist theoretical constraints, which are intrinsic to the activity of observing a computation, that limit unavoidably any conceivable realization of a monitoring system. The two most widely recognized such constraints are represented by the *probe effect* and, in the case of distributed systems, by the *observability problem*.

These two issues represent the analogous of physical laws (such as gravity, inertia) that engineers must take into account when designing a mechanical device. Analogously, the realization of a software monitoring system must deal with the unavoidability of the probe effect, by aiming to minimize it, and with the observability problem by ensuring that, by suitable technical means, observability (or, in other words, recognizability of certain composite events) is guaranteed within a given observation error.

Probe Effect The probe effect[9] has been discussed extensively in the literature (see, for example [80, 145]). The probe effect causes the observed phenomenon to be altered by the observation itself; every observation process is, by it very nature, a destructive process to some extent. From a practical standpoing, the observation of performance-related characteristics of software systems is especially impacted by this problem. Nontheless, the probe effect is also a problem for the observation of distributed systems in general (also w.r.t. functional properties), since the process of monitoring alters the timing of events and therefore may cause wrong behaviour that otherwise wouldn't happen [104, 76]. Analogously, faults (that would have happened otherwise) can be masked as an effect of the interplay of the subject system and the monitoring.

To avoid this interference (or to minimize it), hardware-based monitoring approaches have been proposed [192], but they are not always practical and are not as flexible as software-based approaches.

Observability problem Recognizing that a given complex event has happened is not trivial in distributed loosely-coupled environments [76], as it requires establishing in which order two or more constituent events (originated from different nodes) happened.

As noted by Lamport in his well-known 1978 paper, *in a distributed system it is sometimes impossible to say that one of two events happened first* [115]. To work around the usual absence of a shared global clock, a large body of literature has been produced, tackling the problem of reconstructing causality

[9] Somewhere also referred to as "Heisenberg effect"

relations from an algorithmic/theoretic point of view [37, 119] and applying the proposed approaches to different types of distributed systems (P2P networks, ad-hoc networks, among the others). Concrete synchronization frameworks, such as the network time protocol (NTP) are widely adopted nowadays as mainstream solutions. It is important to note that the observability problem does not necessarily arise in all distributed systems. For example, if one aims to identify the service that takes the maximum average response time among a set of services, the problem is not really distributed, as the observation is in fact local and there is no need for aggregated interpretation.

7.3 Towards an Integrated CONNECT Monitoring Framework

The very vision of CONNECT, i.e., achieving automated and eternal interoperability puts on-line approaches, and therefore monitoring, in a central position in the overall project. Although WP5 is focused on dependability assessment, monitoring in CONNECT may also contribute to bridge the gap between existing approaches to behavioural learning and CONNECTor synthesis – originally conceived for off-line use – and the CONNECT world, where everything happens dynamically and thus requires approaches to work in an on-line fashion.

In CONNECT, monitoring is conceived as a common core service offered to the other Enablers to implement feedback loops whereby approaches to dependability analysis, CONNECTor synthesis, behaviour learning can be applied to an on-line setting and can be enhanced to cope with change and dynamism. Monitoring is performed alongside the functionalities of the CONNECTed System and is used to detect conditions that are deemed relevant by its clients (i.e., the other CONNECT Enablers). Upon detecting one such conditions, the monitoring system alerts the interested client which, in turn, triggers an update of the analysis, synthesis, learning respectively. In this way, powerful but expensive techniques are executed only when necessary. Although monitoring can can provide valuable support to most CONNECT Enablers, it can easily incur in feasibility problems caused by excessive overhead. Also, as we intend to realise a monitoring system that can address different purposes (spanning functional and non-functional aspects), it must be designed with special emphasis on flexibility. An evaluation of these requirements imposed on the monitoring subsystem by the other modules of CONNECT has started and has led us to identify the following key features and guidelines that a comprehensive CONNECT Monitoring Framework should possess. This exercise is meant to be continued throughout the project lifespan in order to define (and change, if necessary) the priorities of our agenda according to the new requirements that could emerge later on.

Distribution, dinamicity, heterogeneity: The monitoring system must be able to observe systems that are inherently distributed, highly dynamic and composed of heterogeneous entities that were not necessarily conceived for interoperation. Monitoring will address distribution by adopting an architecture that is itself distributed, possibly following approaches that employ mobile code and/or a hierarchical organization.

Efficiency: The performance penalty incurred because of monitoring should be minimized, while achieving the intended observation goals. Approaches for minimizing the impact of monitoring will include using statistical sampling or self-tuning algorithms for directing the focus of monitoring to certain parts of the overall monitored system that are deemed especially critical or interesting [25].

Predictable overhead: The approach to overhead reduction followed in most existing monitoring systems follows a best-effort policy, whereby overhead is kept as low as possible but is in fact unbounded.

In CONNECT we will pursue the goal of efficiency by adopting predictable strategies to estimate the computational, storage and transmission resources that are demanded by a given set of monitoring goals. This means that the load caused by monitoring will not only be limited, but also predictable and controllable, along the lines of the approach presented in [34]. Providing a reasoning framework to handle the trade-off between monitoring precision and efficiency will be part of our investigation in the remainder of the project.

Model-driven approach: Model-driven and generative approaches [178] allow software engineers to express the key concepts of a domain or system at a high level of abstraction and then use them for reasoning and for automating coding tasks that are otherwise costly and error-prone. Most of the research effort carried out in CONNECT relies on some sort of models; it is therefore natural to pursue integration at an abstract conceptual level in order to benefit from automated techniques for deriving the actual implementation code. Especially in the integration with the connector synthesis module, we will need to express the abstract behavioural model of a connector in such a way that the input to the monitoring (coming e.g., from the dependability analysis module) can assume a shared understanding of the internals of the connector is in place and therefore both parties (the monitoring and its client) can refer to it unambiguously.

Modularity and Flexibility: The CONNECT monitoring framework will be designed with modularity and flexibility in mind. One of the key goals we aim to achieve is to realize a monitoring system that can be employed for different purposes (including monitoring of functional an non-functional properties) and in different settings, possibly beyond its application to the CONNECT project. To this end, the framework will be constructed around a core module, responsible for basic primitive functionalities, and a set of special-purpose extension modules that use the services offered by the core module to target specific uses of monitoring.

Integrated with the other CONNECT Enablers: Although striving for genericity and flexibility, our framework will have the primary characteristic of being tailored after the specific needs of CONNECT and, in particular, will be matched with the other core functionalities provided by CONNECT Enablers. More on integration is discussed, although still at an abstract level, in the next three subsections.

7.3.1 Integration with CONNECTor synthesis

Deliverable 3.1 [55] thoroughly describes the approach to CONNECTor synthesis followed in the project, and can be briefly summarized as follows.

The approach aims at synthesising a CONNECTor to allow two Networked Systems with matching functionalities to communicate, despite they natively use different protocols. The approach in its initial form has been published in [183].

The synthesis process assumes that a description of the protocols executed by the two Networked Systems is available in the form of labeled transition systems. Also, the LTSs are supposed to be minimal [100].

The structural characteristics of the LTSs are taken into account to create an abstract view of the behaviours of the two parties and to identify corresponding actions. By using ontological resources, semantically corresponding actions are matched despite superficial differences (e.g., naming, ordering). This procedure leads to identifying the so-called *induced LTSs* for both protocols, the fragments of their abstracted behaviours that match (i.e., that use the same language, *modulo* ontological mediation). The existence of a *functional matching* is then checked, by verifying that the provided/required functionalities of the two protocols are similar[10].

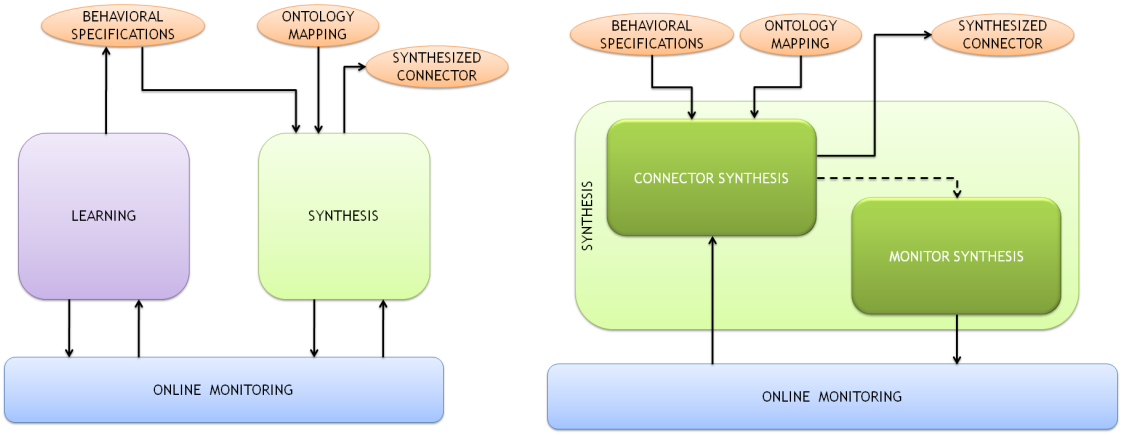
The focus of this approach is on the automated synthesis of a CONNECTor such that it *guarantees* certain properties (i.e., that the two protocols to mediated are actually capable of communicating) by working around (mediating) existing mismatches. This techniques concentrates on guaranteeing that the semantically overlapping fragments of the two protocols are mediated, but in order to do so it *assumes* that certain interactions with the context[11] are carried out correctly and that the behavioural *models* of the protocols are a correct abstraction of their *actual behaviour* [27].

Therefore, applying this approach in practice requires that the assumptions on which the synthesis builds on be checked continuously, and that violations of such assumptions be promptly reported to the CONNECTor synthesis module.

The combination of monitoring and of CONNECTor synthesis functionalities will: 1) result in an automated approach to the synthesis of monitoring agents, tailored to the synthesised connector, whose

[10] For a precise definition of similarity, please refer to Deliverable 3.1 [55]

[11] By "context" here we mean all the other entities that are involved in communications with one of the two parties bridged by the connector, where the result of such interactions is a pre-condition for the communication to be successful.



(a) Monitoring as a supporting functionality for learning and synthesis

(b) Feedback loop between CONNECTor synthesis and monitoring

Figure 7.3: Monitoring in CONNECT: An integrated view

task is to verify on-line the assumptions on which the synthesis process is based; and 2) support the *self-adaptation* of the mediated systems by enacting a feedback loop between monitoring and synthesis.

The integrated process can be automated as the CONNECTor synthesis allows us to to identify the portions of protocols (i.e., the actions) that involve "context interactions"[12] and the ones that refer to the other protocol interactions.

The monitoring will be performed so as to improve efficacy and efficiency: it will observe only some actions of one protocol instead of observing its complete behavior. In particular it will select only some among all the interaction actions with the context, where the selection could be driven by the characterization of "rich states" as presented in [183].

The loop can be closed by an automated adaptation (by triggering a new synthesis or the update of a CONNECTor model previously synthesised) in the case the monitoring should reveal that any assumption has been violated at runtime. In this case, the monitoring gives feedback to the synthesis process in a suitable format such that it can directly manipulate and interpret it in order to self-adapt the system. The adaptation may be pursued in two directions: (1) to generate a new CONNECTor that preserves the original guarantees; or (2) to maintain the previous CONNECTor recomputing the new (conceivably weaker) guarantees that will result from composing the new CONNECTor and the system.

7.3.2 Integration with off-line dependability analysis

A stochastic model of Networked Systems and CONNECTors can be generated when their functional and non-functional specification is available. In order to build the whole CONNECTed System, some additional information about the composition of the CONNECTed System and the type of interactions is also needed -e.g., the CONNECTed System is made of Networked Systems $C_1, \ldots, C_n$ which interact with an asynchronous pattern.

The constructed model is parametric w.r.t. the non-functional specification (e.g., failure rates, transmission time): in order to perform dependability analysis, parameters must be properly instantiated. Hypotheses can be made on parameters that are not available.

It can be the case that accurate values are not available for all the model parameters and approximate values (or even hypothesized ones) are used. In such cases, sensitivity analyses are very recommended. Sensitivity analyses allow identifying the critical parameters out of the many that are employed, that is those parameters to which the system is highly sensible. Indeed, sensitivity analysis allows to evaluate a range of possible system scenarios by varying the values of model parameters, to determine the trends in the consequent variations of the analyzed dependability figures. On one hand, this helps in understand-

[12] Notice that in general, context interactions can involve several protocols

ing the accuracy level that must be attained while estimating the parameters values. Since even slight variations of critical parameter values may result in relevant changes of system dependability attributes, a thorough calibration of such parameters is necessary to increase the level of confidence that can be put on the dependability evaluation itself. On the other hand, it is possible to point out which parts of the system are sensitive the most to the parameter variations, and to adopt responses such as the deployment of adequate fault tolerance techniques to achieve a proper level of dependability.

Dependability analysis provides feedback to the Enabler for estimation of design errors and deficiencies of the synthesised CONNECTor. The analysis can be done either before deployment of the CONNECTor, or in parallel. In the first case, the deployment of the CONNECTor can be made only when the analysis completes and the results point out that the requirements of the specification are met. If the requirements are not met, the CONNECTor should not be deployed and a new synthesis is needed. Analysis results can be stored in a repository to hold information about the estimated quality of service of NSs, CONNECTors and CONNECTed System.

Once the CONNECTor is deployed, the monitoring system can provide useful feedback to dependability analysis. Indeed, non-functional parameters can be monitored for the actual deployment, and more precise values can be provided to refine the stochastic model and perform a more precise dependability analysis. Also, monitoring can point out if there are mismatches between the actual quality of service and the expected quality estimated by the analysis. The mismatch could be the symptom of inaccurate analysis (maybe because model input parameters were not accurate enough), but also could reveal evolution undertaken by the CONNECTed System and so does not necessarily point out a deficiency of the stochastic analysis. In both cases, model refinement is necessary, to update the prediction of dependability levels for future usages.

7.3.3 Integration with behaviour learning

At the current stage of the project, behaviour learning, which is investigated in WP4, is an inherently *off-line* activity. It includes observing Networked Systems behaviour but assumes an approach that is typical of active-testing techniques, where the subject system is available in the lab and where artificially crafted interactions are used to explore the subject system's behaviour. Therefore, learning is carried out on a local basis, which frees monitoring of all issues coming from observing a distributed system. In general, off-line settings allow us to work around important challenges (especially w.r.t. overhead and intrusiveness) that are unavoidable on-line.

On the other hand, our focus in the project is on *on-line* monitoring, and therefore the very above-mentioned challenges will be tackled by our research in the project.

The integration of learning and monitoring will be addressed in the second year of the project. It will start by identifying specific needs of the learning module with respect to monitoring. These needs will be accommodated by enhancing the interface offered by the monitoring framework with dedicated primitives and services. The actual observation will be carried out based on the underlying core, which is shared for all the uses of monitoring in CONNECT and which is meant to provide on-line support to observing different types of characteristics, both functional and non-functional. This core, plus the support to observing non-functional properties will be realized as part of WP5 activities.

The other functionalities of the framework that are meant to provide specific support to (re-)learning, will be the focus of future research in WP4, starting in year two. Monitoring will act as a bridge between existing learning approaches – initially conceived for off-line application and to learn "stationary" behaviours – and the dynamic open world of evolving systems that is peculiar of the CONNECT vision. To this end we will investigate and implement mechanisms to observe running CONNECTed Systems *efficiently*, in order to provide feedback to the learning Enablers in such a way that existing learned behavioural models can be continuously updated based on actual observations taken in the field.

Efficiency is a major concern for on-line monitoring, and it is not our goal to exhaustively verify the conformance of observed behaviour with its (learned) specification. Therefore we will investigate specific techniques that aim at minimizing overhead by allowing lossy observation, thus trading completeness of observation with efficiency, while preserving good detection power (i.e., while minimizing the probability of missing interesting events).

7.3.4 Integration with Security-by-Contract

Referring to Chapter 5, the Security-by-Contract framework can be extended in order to deal with the measure of trust associated with an Enabler by adding a contract monitoring functionality. Indeed, trust measures assigned to security assertions can be adjusted as a result of a contract monitoring strategy in order to grant certain permissions to a CONNECTor.

Trust measures associated with an Enabler concern the contract goodness mainly. Hence, updated trust measures will influence future interactions with a CONNECTor and an Enabler. In other words, our system penalizes the Enabler when the contract does not specify CONNECTor's behaviour correctly.

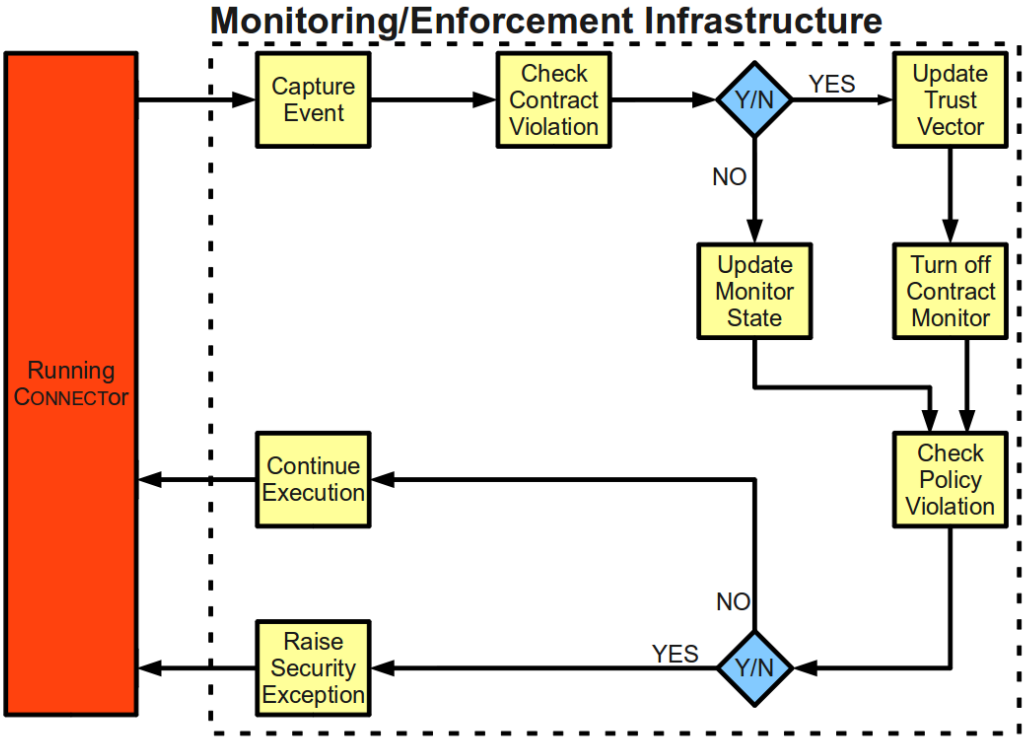


Figure 7.4: Enforcement strategy.

Here we present a possible extension of the monitoring/enforcement infrastructure model proposed in [57] by making the *policy decision point* (PDP) also responsible for the contract monitoring operations and for the trust vector updating and, from now on, we refer to it as the *Monitoring/Enforcement Infrastructure*. Roughly, in [57] a monitoring/enforcement infrastructure consists in a PDP that holds the actual security state and is responsible for accepting or refusing new actions and *Policy enforcement points* (PEPs) that are both in charge of intercepting actions to be dispatched to the PDP and preventing the execution of not allowed operations.

According to [36, 57], we assume that both contracts and policies are specified through the same formalism. Hence, the policy enforcement configuration of the PDP keeps unchanged. The PDP must load connector contracts as well as local private security policies dynamically. Moreover, it must be able to run under three different execution scenarios policy enforcement enabled, contract monitoring enabled or both.

The "policy enforcement enabled" scenario is actually unchanged w.r.t. the standard usage of the classical PDP in S×C. Hence, no contract monitoring or trust management operations are involved.

Main interest resides in the other two scenarios. The contract monitoring scenario applies to CONNECTors carrying a contract released by a trusted Enabler. The main difference w.r.t. the previous scenario is that the monitor keeps the program events trace in memory. When a signal arrives, the monitor checks whether it is consistent with the monitored contract. If the contract is respected, then the moni-

tor updates its internal state and answers by permitting the operation. Otherwise, if a violation attempt happens, the monitor reacts changing its state.

The first consequence of a contract violation is a decreasing of the trust weights of transitive security assertions. Indeed, the contract monitor detected a fake execution of a trusted application w.r.t. its declared contract.

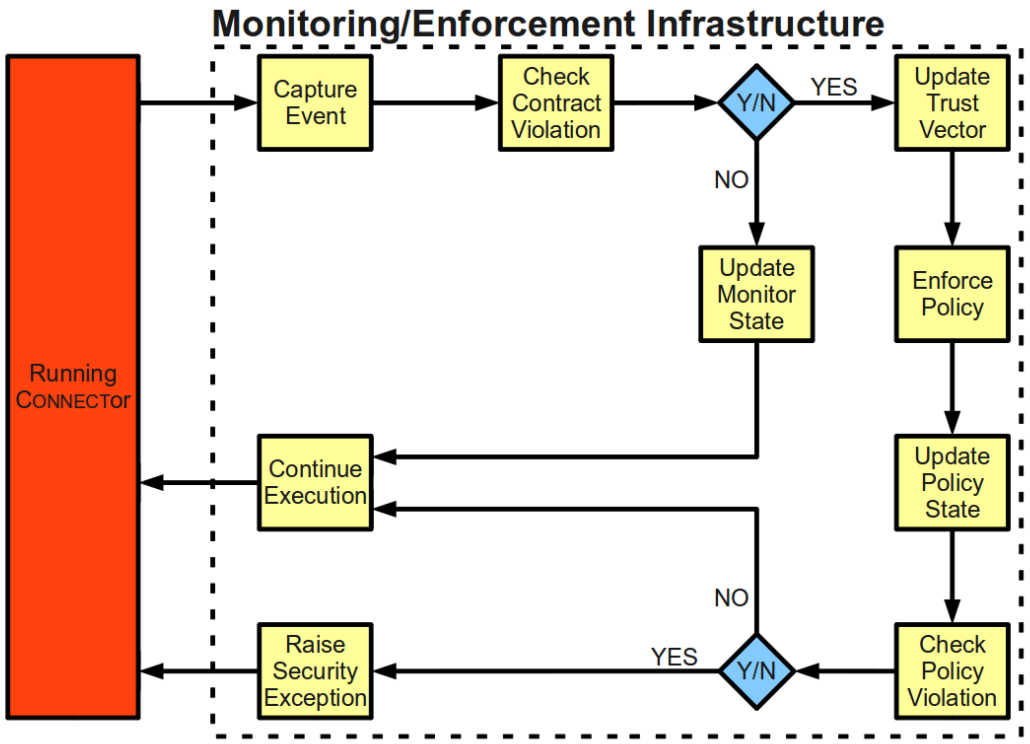


Figure 7.5: Contract monitoring strategy

Secondly, the monitor changes from contract monitoring to policy enforcement configuration. Since an instance of the policy is always present, this operation does not imply a serious computational overhead. Afterwards, the policy state is updated using the execution trace recorded during the monitoring phase. This step, that can be time consuming, is necessary for verifying whether, breaking the contract, the CONNECTor has also violated the policy. However, this computational cost, being the consequence of an extraordinary event, must be paid at most once. Indeed, when the monitor is performing both contract monitoring and policy enforcement, the current policy state is known. Finally, the execution continues with the monitor enforcing the policy starting from the last action, that is the event breaking the contract.

Figure 7.4 and Figure 7.5 show the behaviour of the monitor/enforcement infrastructure performing the contract monitoring task in the two previously discussed scenarios.

Summing up, both execution scenario 2 and 3 check contract violations through the contract monitoring strategy described above and update Enablers' trust level. Such updates will influence future interactions with CONNECTors and Enablers. Trust measures associated with Enablers concern the contract goodness mainly. In other words, our system penalizes the Enabler more when the contract does not specify CONNECTor's behaviour correctly, rather than when the CONNECTor itself contradicts the user's security policy.

7.4 Discussion and Future Directions

The focus of the CONNECT project on *future-proof interoperability* and its goal of pursuing continuous composition of Networked Systems through behaviour learning and CONNECTor synthesis techniques,

requires a sophisticated monitoring system to be put in place, that is capable of supporting activities of different kind, spanning dependability assessment, CONNECTor synthesis, behaviour (re-)learning, and trust management. This variety of needs requires a comprehensive framework to be put in place, that is powerful and flexible at the same time, coping with on-line observation of dynamic heterogenous systems, such as those envisioned in CONNECT.

During the first year of the project we have worked toward this ambitious goal by pursuing different directions. *Firstly*, we have surveyed the literature on monitoring and on related problems, in order to identify the fundamental concepts (Section 7.1) and the key components of monitoring systems (Section 7.2). *Secondly*, based on this preliminary study, we have outlined, in Section 7.3, a set of features and guidelines that will drive our future research and development efforts on the CONNECT Monitoring Framework, by taking into account the specific issues related to integrating monitoring and the other core Enablers in the CONNECT ecosystem. A high-level sketch (still tentative at this stage) of this integration plan is outlined in Sections 7.3.1 through 7.3.4. Integration with other elements of the CONNECT architecture has not been tackled yet, but is planned for the second year of the project, during which we also plan to address the following points:

Core monitoring functionalities: We will design and implement a prototype of the core (generic) module of the monitoring framework. This activity will include studying frameworks and languages for the specification of complex events and possibly for the definition of a CONNECT-specific such language. We will follow a model-driven methodology, designing a metamodel for the event specification language to be adopted in CONNECT and for defining the monitoring system itself.

Client-specific functionalities: We will pursue the integration of monitoring with learning, synthesis and dependability analysis both at the conceptual level and through concrete studies based on prototype implementations. Detailed requirements will be elicited for each of the aforementioned clients, and from other potential clients in the CONNECT ecosystem. These requirements will allow us to augment the core module with client-specific capabilities.

8 Conclusions

WP5 addresses the dependability of the eternally CONNECTed Systems. As we explained in Chapter 2, the term *dependability* has been considered in a broad sense (as illustrated in Figure 2.1) to include not only the classical attributes associated with dependability, see Figure 2.2, but also performance, security and trust concerns. WP5 targets the development of new concepts, new metrics, and new approaches for assessing and ensuring dependability, in spite of accidental or intentional faults, and also in spite of the natural evolution of Networked Systems. WP5 activities have been structured into four tasks, namely:

Task 5.1 : Dependability metrics for open dynamic systems

Task 5.2 : Dependability verification & validation in evolving, adaptive contexts

Task 5.3 : Security and privacy

Task 5.4 : Distributed trust management

As evident, the workpackage scope is quite broad. In the lifespan of the project, we pursue the ambitious aim of combining the above tasks and the different dependability concerns into a unified framework. The previous chapters have presented the models we have developed in the first year. In the following we attempt a first proposal of a workflow combining the separate concerns, i.e., dependability analysis, security enforcement and trust management, which is centered around a comprehensive monitoring framework.

8.1 WP5 Workflow Process View

In Work Package 1 an overall flow of information between all the Enablers foreseen in the CONNECT system has been produced [52]. Taking the move from that view, in Figure 8.1, which we adapted from the overall CONNECT information flow [52], we evidence the data-flow of relevance to WP5 tasks, in particular:

- we monitor the Networked Systems and the CONNECTed System; the observed behaviour is exploited within WP5 for dependability analysis (through the output provided to the "Guarantees Expected from the CONNECT System"), and for security enforcement and trust assessment (both through the output provided to the "Logs that match specified conditions"). Moreover, (through the latter) the monitoring Enabler provides also feedback to the learning Enabler and the synthesis Enabler. The monitoring is instructed by the metrics of interest and the conditions to be checked;
- we perform extensive dependability analyses, which provide feedback to the synthesis Enabler; such analyses are currently off-line, but on-line approaches are also foreseen;
- we check and enforce (interacting with the Networked Systems and the CONNECTed System) specified security properties;
- we assess and manage the trustworthiness of the Networked Systems and CONNECTors, and also of Enablers.

The activities relative to the above four tasks span over all stages of the CONNECT process, namely they cover discovery, synthesis and execution time. Combining with each other such activities (and integrating them within the overall CONNECT process) is a complex iterative endeavor, to which we will dedicate WP5 future efforts. At the current stage, we have discussed and outlined an initial high-level scheme of a WP5-centric vision of CONNECT process, which is depicted in Figure 8.2.

The process is modeled as an activity diagram with separate swim-lanes for each dependability concern described earlier, and for monitoring; we also include a generic "other CONNECT activities" swim-lane, representing the rest of the CONNECT activities other than WP5's ones. Along this generic swim-lane, the CONNECT approach is triggered by a request of communication. This request, originating from a Networked System *S1* within the CONNECTed world, is intercepted by CONNECT discovery Enablers.

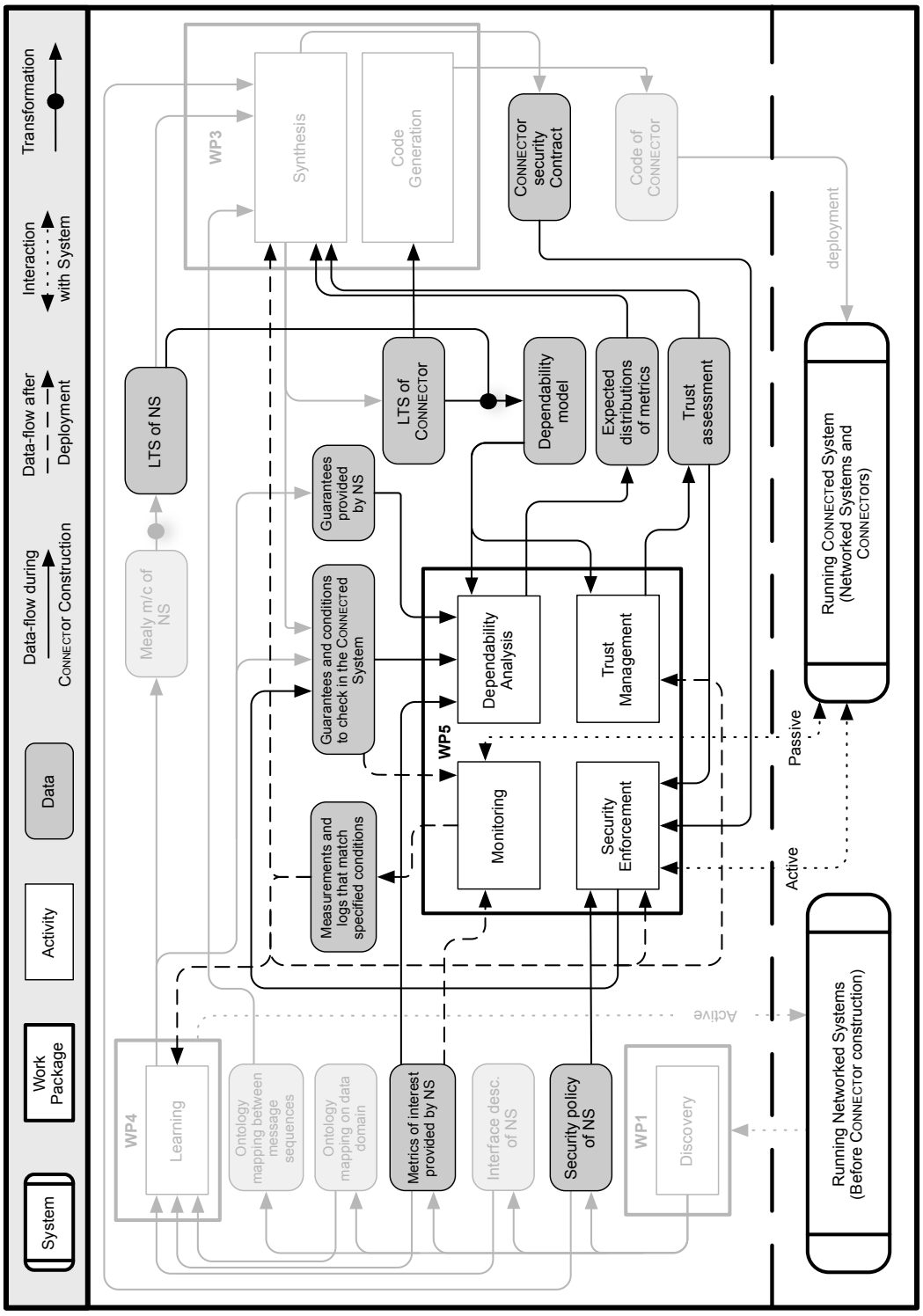


Figure 8.1: Overview of WP5 activities within CONNECT data flow

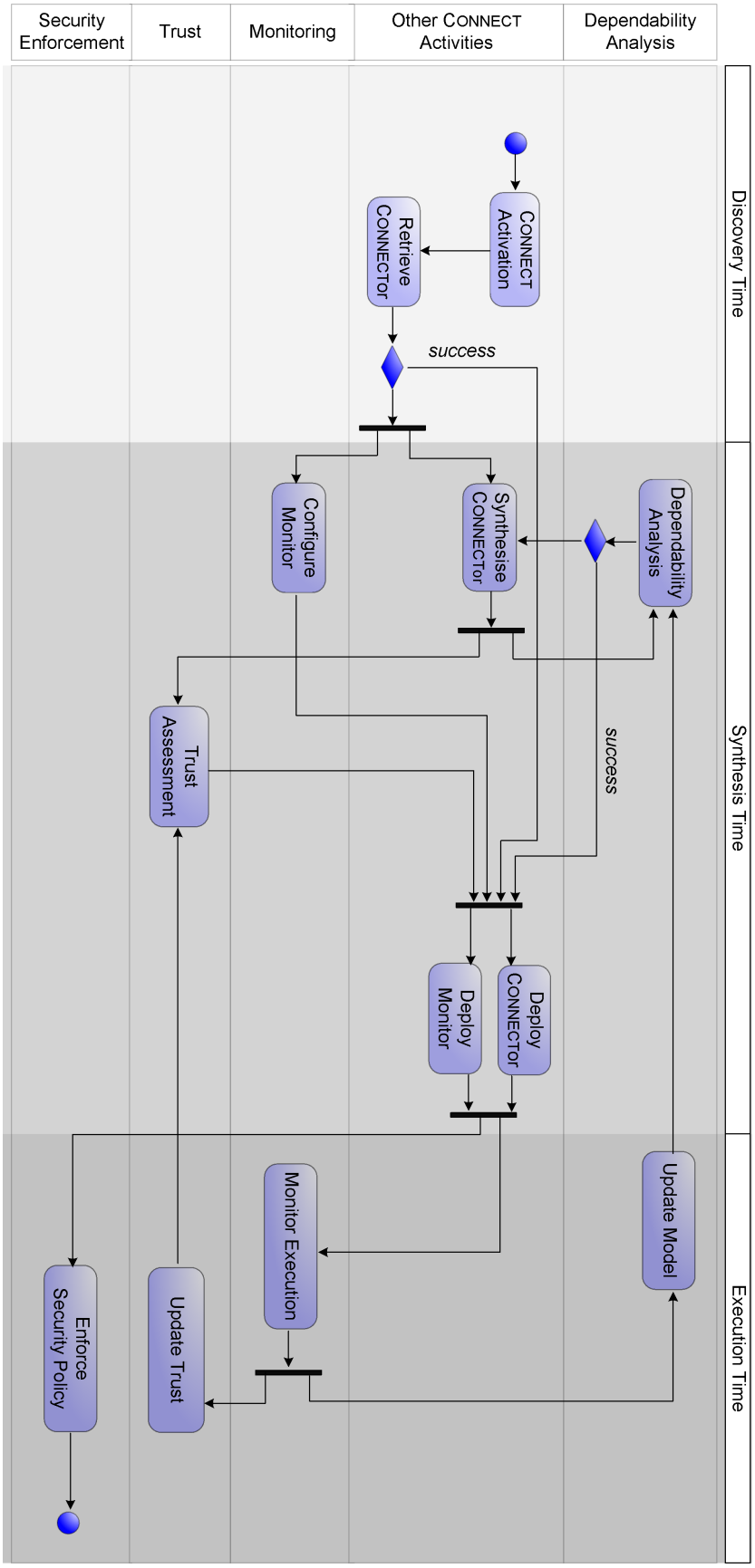


Figure 8.2: A lifecycle workflow of WP5 activities within the CONNECT process

To serve *S1* request, we have here a first decision point: does there already exist a suitable CONNECTor that can be reused? So, when a discovery Enabler *E* accepts the request, it will first search for a suitable CONNECTor already synthesised. *Suitable* here means that, among all the functional and non-functional characteristics of the CONNECTor, this yields adequate dependability properties and a trust level at least equal to the trust level associated to *E*. If the answer is positive, then the Enabler retrieves an implementation of the CONNECTor from a repository and deploys it, CONNECTing *S1* with a receiver Networked System *S2*. Moreover, the deployed CONNECTor is provided with a monitor that at execution time will warn the Enabler when and if the established communication is no longer satisfying *S1* needs (this may happen for many reasons: because either the environment or *S2* changed, or because the Trust reputation decays, or due to negative reports from on-line analysis).

If no suitable CONNECTor is available, then the synthesis process is started by the synthesis Enabler. The latter may interact with the learning Enabler to infer the desired functional behaviour of the CONNECTor, and may also obtain some dependability requirements from *S1* interface description. Hence, during the synthesis, the Enabler will interact with the dependability analysis Enablers to predict whether the built CONNECTor is satisfacing[1].

At execution time, the monitoring mechanism is activated to keep track of CONNECTor behaviour (monitoring at the Networked Systems interfaces) and of the CONNECTed System (end-to-end). Also at execution time, when security specifications are provided (security-by-contract), security enforcement mechanisms are activated. This discover and synthesis flow cycles whenever the communication is no longer satisficing.

Throughout, the Trust management model is pervasive, in that among the available Enablers, those yielding the highest trust reputation can be chosen. Trust reputation is updated at execution time according to monitoring feedback.

8.2 Summing Up and the Way Forward

This document reports the results of activity carried out in the first year within WP5 (which was launched in M4). Most significant results include:

- In Chapter 2, a conceptual framework for understanding and measuring CONNECT dependability metrics.
- In Chapter 3, an extensive survey of soft-metrics, acknowledging the idea that, to be dependable, systems need to be appropriate to potential users.
- In Chapter 4, initial dependability analysis addressing diffusion protocols in dynamic and heterogeneous environments. Specifically, a common case study has been adopted and analysed through the two main approaches to dependability analysis in CONNECT. The obtained results also point out complementarities between the two approaches.
- In Chapter 5, a security conceptual model, based on the Security-by-Contract (SxC) paradigm, guaranteeing the security of communicating systems composed by several, heterogeneous Networked Systems.
- In Chapter 6, a distributed trust model for Enablers, CONNECTors and CONNECTed System assessment.
- In Chapter 7, a detailed study of existing monitoring approaches, towards a CONNECT monitoring Enabler, and preliminary attempts of integration with synthesis, dependability analysis and security enforcement.

As evident, the workpackage scope is quite broad. We pursue the ambitious aim of combining the above outlined different approaches and concerns into a unified framework. A first proposal of a unified WP5-centric lifecycle encompassing dependability analysis, security enforcement and trust management, which is centered around the monitoring Enabler, has been outlined in Figure 8.2. WP5 activities span over discovery time, synthesis time and execution time.

[1] The word *satisfice*, coined by Herbert Simon, blends "satisfy" and "suffice", to highlight the aim to meet criteria for adequacy, rather than to identify an optimal solution.

Note that even though formally the WP involves few partners, the concern for dependability is pervasive in CONNECT and is transversal to all other WPs, and in fact partners from other WPs, WP1-WP4 (e.g., Univaq, PKU, Docomo, TUDO) have been actively involved in the WP activities.

Much discussion has been ongoing lately to understand how the complex notion of dependability, in the broad sense we mean here, adapts to the dynamic CONNECT vision, and how traditional dependability attributes (e.g., reliability, availability, security, etc) should be measured in CONNECT, leading to a two-dimension conceptual framework as described in Chapter 2 (see Figure 2.7). We will continue revising the metrics framework, since this model has presented interesting perspectives to direct dependability analysis and assurance. We also need to complement the models under development, with fault models and risks addressed.

In the first year the focus has been on devising appropriate models and background material for the various dependability concerns, building a common ground of understanding. In the remaining project years, taking stock of each partner's expertise, we intend to focus efforts on integration, looking for a synergy of approaches.

Bibliography

[1] *ISO 8402-1986(GB/T6583-1992): Quality-Vocabulary*, June 1986.

[2] *IEEE Std 610.12-1990: IEEE Standard Glossary of Software Engineering Terminology*, 1990.

[3] Dependable Systems of Systems (DSoS) EU FP5 Project, 2000–2003.

[4] Malicious-and Accidental-Fault Tolerance for Internet Applications EU FP5 Project, 2000–2003.

[5] A. Lomuscio and H. Qu and M. Solanki. Towards verifying compliance in agent-based web service compositions. In *Proceedings of The Seventh International Joint Conference on Autonomous Agents and Multi-agent systems (AAMAS-08)*, pages 265–272. IFAAMAS, 2008.

[6] A. Abdul-Rahman and S. Hailes. A distributed trust model. In *NSPW: New Security Paradigms Workshop*, pages 48–60, New York, USA, 1997. ACM Press.

[7] R. Adaikkalavan and S. Chakravarthy. Event specification and processing for advanced applications: Generalization and formalization. In R. Wagner, N. Revell, and G. Pernul, editors, *DEXA*, volume 4653 of *Lecture Notes in Computer Science*, pages 369–379. Springer, 2007.

[8] M. Ajmone Marsan, G. Balbo, and G. Conte. A class of generalized stochastic petri nets for the performance evaluation of multiprocessor systems. *ACM Transactions on Computer Systems*, 2(2):93–122, 1984.

[9] M. Ajmone Marsan and G. Chiola. On Petri nets with deterministic and exponentially distributed firing times. In G. Rozenberg, editor, *Advances in Petri Nets 1987*, volume 266 of *LNCS*, pages 132–145. Springer-Verlag, 1987.

[10] C. Alberts, S. Behrens, R. Pethia, and W. Wilson. Operationally critical threat, asset, and vulnerability evaluation framework, version 1.0. Technical Report CMU/SEI-99-TR-017, Carnegie Mellon University, 1999.

[11] R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, 2002.

[12] T. Anderson, editor. *Resilient computing systems: vol. 1*. John Wiley & Sons, Inc., New York, NY, USA, 1986.

[13] R. Aringhieri, E. Damiani, S. D. C. Di Vimercati, S. Paraboschi, and P. Samarati. Fuzzy techniques for trust and reputation management in anonymous peer-to-peer systems: Special topic section on soft approaches to information retrieval and information access on the web. *JASIST : Journal of the American Society for Information Science and Technology*, 57(4):528–537, 2006.

[14] A. Avižienis. Design of fault-tolerant computers. In *AFIPS '67 (Fall): Proceedings of the November 14-16, 1967, fall joint computer conference*, pages 733–743, New York, NY, USA, 1967. ACM.

[15] A. Avizienis, J. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, 2004.

[16] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Secur. Comput.*, 1(1):11–33, 2004.

[17] R. Axelrod. *Complexity of Co-operation: agent based models of competition and collaboration*. Princeton UP, Princeton, 1997.

[18] I. Bahar, E. Frohm, C. Gaona, G. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. In *Proc. International Conference on Computer-Aided Design (ICCAD'93)*, pages 188–191, 1993. Also available in *Formal Methods in System Design*, 10(2/3):171–206, 1997.

[19] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. On the logical characterisation of performability properties. In U. Montanari, J. Rolim, and E. Welzl, editors, *Proc. 27th International Colloquium on Automata, Languages and Programming (ICALP'00)*, volume 1853 of *LNCS*, pages 780–792. Springer, 2000.

[20] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model-checking algorithms for continuous-time Markov chains. *IEEE Transactions on Software Engineering*, 29(6):524–541, 2003.

[21] G. Balbo. Introduction to stochastic petri nets. In J.-P. Katoen, H. Brinksma, and H. Hermanns, editors, *Lectures on Formal Methods and Performance Analysis : First EEF/Euro Summer School on Trends in Computer Science Berg en Dal, The Netherlands, July 3-7, 2000, Revised Lectures*, volume 2090 of *Lecture Notes in Computer Science*, pages 84–155. Springer-Verlag, 2001.

[22] L. Baresi, C. Ghezzi, and S. Guinea. Smart monitors for composed services. In *ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing*, pages 193–202, New York, NY, USA, 2004. ACM.

[23] J. Barnat, L. Brim, I. Cerna, M. Ceska, and J. Tumova. ProbDiVinE-MC: Multi-core LTL model checker for probabilistic systems. In *Proc. 5rd International Conference on Quantitative Evaluation of Systems (QEST'08)*, pages 77–78. IEEE CS Press, 2008.

[24] A. I. Batavia and G. S. Hammer. Toward the development of consumer-based criteria for the evaluation of assistive devices. *J. of Rehabilitation Research and Development*, 27(4):425–436, 1990.

[25] A. Bertolino, G. D. Angelis, A. Sabetta, and S. G. Elbaum. Scaling up SLA monitoring in pervasive environments. In A. L. Wolf, editor, *ESSPE*, pages 65–68. ACM, 2007.

[26] D. Besnard and B. Arief. Computer security impaired by legitimate users. *Computers & Security*, 23:253–264, 2004.

[27] C. Blundell, D. Giannakopoulou, and C. S. Păsăreanu. Assume-guarantee testing. In *Proceedings of the 2005 conference on Specification and verification of component-based systems*, New York, NY, USA, 2005. ACM.

[28] A. Bobbio, A. Puliafito, M. Telek, and K. S. Trivedi. Recent developments in non-Markovian stochastic Petri nets. *Journal of Circuits, Systems and Computers*, 8(1):119–158, 1998.

[29] A. Bobbio and M. Telek. Non-exponential stochastic Petri nets: an overview of methods and techniques. *Computer Systems Science and Engineering*, 13(6):339–351, 1998.

[30] K. Boehner, J. Vertesi, P. Sengers, and P. Dourish. How hci interprets the probes. In *Proceedings of the 2007 CHI Conference on Human Factors in Computing Systems*, 2007.

[31] A. Bondavalli, S. Chiaradonna, and F. D. Giandomenico. Model-based evaluation as a support to the design of dependable systems. In H. B. Diab and A. Y. Zomaya, editors, *Dependable Computing Systems: Paradigms, Performance Issues, and Applications*, pages 57–86. Wiley, 2005.

[32] E. Borgia, M. Conti, F. Delmastro, and L. Pelusi. Lessons from an ad hoc network test-bed: Middleware and routing issues. *Ad Hoc & Sensor Wireless Networks*, 1(1-2), 2005.

[33] B. Cabral, P. Marques, and L. Silva. Rail: code instrumentation for .net. In *Proceedings of the 2005 ACM symposium on Applied computing*, pages 1282–1287, New York, NY, USA, 2005. ACM.

[34] S. Callanan, D. Dean, M. Gorbovitski, R. Grosu, J. Seyster, S. Smolka, S. Stoller, and E. Zadok. Software monitoring with bounded overhead. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–8, April 2008.

[35] J. Carroll, S. Howard, F. Vetere, J. Peck, and J. Murphy. Just what do the youth of today want? technology appropriation by young people. In *The 35th Hawaii International Conference on System Sciences*, 2002.

[36] A. Castrucci, F. Martinelli, P. Mori, and F. Roperti. Enhancing java me security support with resource usage monitoring. In *ICICS*, pages 256–266, 2008.

[37] B. Charron-Bost, F. Mattern, and G. Tel. Synchronous, asynchronous, and causally ordered communication. *Distributed Computing*, 9(4):173–191, 1996.

[38] D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1):65–75, 1988.

[39] Y. Chen and W. Tsai. Towards dependable service-orientated computing systems. *Simulation Modelling Practice and Theory*, 17(8):1361–1366, 2009.

[40] K. Cheverst, Fitton, M. Rouncefield, and C. Graham. Smart mobsŠ and technology probes: Evaluting texting at work. In *The 11th European Conference on Information Technology Evaluation (ECITE 2004)*, pages 73–80, Amsterdam, Holland, 2004.

[41] S. Chiaradonna, F. Di Giandomenico, and P. Lollini. Interdependency analysis in electric power systems. In R. Setola and S. Geretshuber, editors, *Critical Information Infrastructure security - 3rd International Workshop CRITIS 2008, Revised Papers*, volume 5508 of *LNCS*, pages 60–71, 2009.

[42] H. Choi, V. G. Kulkarni, and K. S. Trivedi. Performance modeling using Markov regenerative stochastic Petri nets. *Performance Evaluation*, 20(1–3):339–356, 1994.

[43] G. Ciardo, A. Blakemore, P. Chimento, J. Muppala, and K. Trivedi. Automated generation and analysis of markov reward models using stochastic reward nets. In C. Meyer and R. Plemmons, editors, *Linear Algebra, Markov Chains, and Queueing Models, IMA Volumes in Mathematics and its Applications, volume 48*, pages 145–191. Springer-Verlag, 1993.

[44] G. Ciardo, R. German, and C. Lindemann. A characterization of the stochastic process underlying a stochastic petri net. *IEEE Transactions on Software Engineering*, 20(7):506–515, 1994.

[45] F. Ciesinski and C. Baier. Liquor: A tool for qualitative and quantitative linear time analysis of reactive systems. In *Proc. 3rd International Conference on Quantitative Evaluation of Systems (QEST'06)*, pages 131–132. IEEE CS Press, 2006.

[46] G. Clark, T. Courtney, D. Daly, D. D. Deavours, S. Derisavi, J. M. Doyle, W. H. Sanders, and P. G. Webster. The Mobius modeling tool. In *9th Int. Workshop on Petri Nets and Performance Models*, pages 241–250, Aachen, Germany, September 2001. IEEE Computer Society Press.

[47] E. Clarke and E. Emerson. Design and synthesis of synchronization skeletons for branching-time temporal logic. In *Proceedings of Workshop on Logic of Programs*, volume 131 of *LNCS*, pages 52–71. Springer-Verlag, 1981.

[48] E. Clarke, M. Fujita, P. McGeer, K. McMillan, J. Yang, and X. Zhao. Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. In *Proc. International Workshop on Logic Synthesis (IWLS'93)*, pages 1–15, 1993. Also available in *Formal Methods in System Design*, 10(2/3):149–169, 1997.

[49] M. Cohen, M. Dam, A. Lomuscio, and H. Qu. A data symmetry reduction technique for temporal-epistemic logic. In *the 7th International Symposium on Automated Technology for Verification and Analysis, (ATVA 2009)*, LNCS 5799, pages 69–83. Springer, 2009.

[50] M. Cohen, M. Dam, A. Lomuscio, and H. Qu. A symmetry reduction technique for model checking temporal-epistemic logic. In *the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pages 721–726, 2009.

[51] M. Colombo, F. Martinelli, P. Mori, M. Petrocchi, and A. Vaccarelli. Fine grained access control with trust and reputation management for globus. In *OTM Conferences (2)*, pages 1505–1515, 2007.

[52] CONNECT Consortium. Deliverable 1.1 – Initial CONNECT Architecture.

[53] CONNECT Consortium. Deliverable 2.1 – Capturing functional and non-functional connector behaviours.

[54] CONNECT Consortium. Deliverable 6.1 – Experiment Scenarios.

[55] CONNECT Consortium. Deliverable 6.1 – Modeling of Application- and Middleware-layer Interaction Protocols.

[56] G. Costa, N. Dragoni, A. Lazouski, F.Martinelli, F. Massacci, and I.Matteucci. Extending security-by-contract with quantitative trust on mobile devices. In *IMIS'10: Proceedings of the 4th International Workshop on Intelligent, Mobile and Internet Services in Ubiquitous Computing*, 2010. To appear.

[57] G. Costa, F. Martinelli, P. Mori, C. Schaefer, and T. Walter. Runtime monitoring for next generation java me platform. *Computers & Security*, July 2009.

[58] G. Costa and I. Matteucci. Enforcing private policy via security-by-contract. *Identity and Privacy Management. Special issue of the journal UPGRADE*, 2010.

[59] A. Crabtree, T. Hemmings, T. Rodden, K. Cheverst, K. Clarke, G. Dewsbury, J. Hughes, and M. Rouncefield. Designing with care: Adapting cultural probes to inform design in sensitive settings. In *OzCHI 2003, New Directions in Interaction: Information environments, Media and Technology*, pages 4–13, Queensland, Australia, November 2003.

[60] P. D'Argenio, B. Jeannet, H. Jensen, and K. Larsen. Reachability analysis of probabilistic systems by successive refinements. In L. de Alfaro and S. Gilmore, editors, *Proc. 1st Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification (PAPM/PROBMIV'01)*, volume 2165 of *LNCS*, pages 39–56. Springer, 2001.

[61] L. Desmet, W. Joosen, F. Massacci, P. Philippaerts, F. Piessens, I. Siahaan, and D. Vanoverberghe. Security-by-contract on the .net platform. volume 13, pages 25–32, Oxford, UK, UK, 2008. Elsevier Advanced Technology Publications.

[62] G. Dewsbury. Designing dependable digital domestic environments. In *HOIT 2003: The Networked Home and the Home of the Future*, Irvine, California, 2003.

[63] G. Di Marzo Serugendo. Robustness and dependability of self-organizing systems - a safety engineering perspective. In *SSS*, pages 254–268, 2009.

[64] G. Di Marzo Serugendo, J. Fitzgerald, A. Romanovsky, and N. Guelfi. A metadata-based architectural model for dynamically resilient systems. In *SAC '07: Proceedings of the 2007 ACM symposium on Applied computing*, pages 566–572, New York, NY, USA, 2007. ACM.

[65] M. Dmitriev. Selective profiling of java applications using dynamic bytecode instrumentation. In *Proceedings of the 2004 IEEE International Symposium on Performance Analysis of Systems and Software*, pages 141–150, Washington, DC, USA, 2004. IEEE Computer Society.

[66] P. Dourish and K. Anderson. Collective information practice: Exploring privacy and security as sociasl and cultural phenomena. *Human Computer Interaction*, 21:319–342, 2006.

[67] P. Dourish, R. Grinter, J. D. de la Flor, and M. Joseph. Security in the wild: User strategies for managing security as an everyday, practical problem. *Personal and Ubiquitous Computing*, 8:391–401, 2004.

[68] N. Dragoni, F. Martinelli, F. Massacci, P. Mori, C. Schaefer, T. Walter, and E. Vetillard. Security-by-contract (SxC) for software and services of mobile systems. In *At your service - Service-Oriented Computing from an EU Perspective.* MIT Press, 2008.

[69] N. Dragoni, F. Massacci, and K. Naliuka. Security-by-contract(SxC) for mobile systems- or how to download software on your mobile without regretting it. Position Papers for W3C Workshop on Security for Access to Device APIs from the Web, December 2008.

[70] K. Edwards, E. S. Poole, and J. Stoll. Security automation considered harmful? In *NSPWŠ07*, North Conway, NH, USA., September 2007.

[71] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about Knowledge*. MIT Press, Cambridge, 1995.

[72] L. Falai, A. Bondavalli, and F. Di Giandomenico. Quantitative evaluation of distributed algorithms using the neko framework: the nekostat extension. In *2nd Latin-American Symposium on Dependable Computing*, volume 3747/2005 of *LNCS*, pages 35–51. Lecture Notes in Computer Science, 2005.

[73] A. Fehnker and P. Gao. Formal verification and simulation for performance analysis for probabilistic broadcast protocols. In *Proc. 5th International Conference on Ad-Hoc, Mobile, and Wireless Networks (ADHOC-NOW'06)*, volume 4104 of *LNCS*, pages 128–141. Springer, 2006.

[74] L. Feldkuhn and J. Erickson. *Event Management as a Common Functional Area of Open Systems Management*. 1989.

[75] C. Fetzer and K. Högstedt. Challenges in making pervasive systems dependable, 2003.

[76] C. Fidge. Fundamentals of Distributed System Observation. *IEEE Softw.*, 13(6):77–83, 1996.

[77] B. Friedman, D. Hurley, D. C. Howe, E. Felten, and H. Nissenbaum. Users' conceptions of web security: a comparative study. In *CHI '02: CHI '02 extended abstracts on Human factors in computing systems*, pages 746–747, New York, NY, USA, 2002. ACM.

[78] B. Friedman and P. Kahn. *Human Values, Ethics, and Design. The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies, and Emerging Applications.*. Lawrence Erlbaum Associates, Mahwah, NJ.

[79] B. Friedman, P. Kahn, and A. Borning. *Human-computer interaction in management information systems: Foundations*, chapter Value Sensitive Design and information systems, pages 348–372. Armonk, New York; London, England., 2006.

[80] J. Gait. A Probe Effect in Concurrent Programs. *Softw., Pract. Exper.*, 16(3):225–233, 1986.

[81] W. Gaver, A. Dunne, and E. Pacenti. Design: cultural probes. *Interactions*, 6(1):21–29, 1999.

[82] C. Geertz. *The Interpretation of Cultures*. Basic Books, New York, 1973.

[83] M. Geimer, S. S. Shende, A. D. Malony, and F. Wolf. A generic and configurable source-code instrumentation component. In *Proceedings of the 9th International Conference on Computational Science*, pages 696–705, Berlin, Heidelberg, 2009. Springer-Verlag.

[84] R. German. Non-Markovian analysis. In E. Brinksma, H. Hermanns, and J. P. Katoen, editors, *Lectures on Formal Methods and Performance Analysis*, volume 2090 of *LNCS*, pages 156–182. Springer-Verlag, 2001.

[85] J. Golbeck and J. Hendler. Filmtrust: Movie recommendations using trust in web-based social networks. In *CCNC: IEEE Consumer Communications and Networking Conference*, pages 282–286, Las Vegas, NV, USA, 2006. IEEE Computer Society.

[86] T. Grandison and M. Sloman. A survey of trust in internet applications. *IEEE Communications Surveys and Tutorials*, 3(4):2–16, 2000.

[87] P. Greci, F. Martinelli, and I. Matteucci. A framework for contract-policy matching based on symbolic simulations for securing mobile device application. In *ISoLA*, pages 221–236, 2008.

[88] G. Guirado, T. Hérault, R. Lassaigne, and S. Peyronnet. Distribution, approximation and probabilistic model checking. In *Proc. 4th International Workshop on Parallel and Distributed Methods in Verification (PDMC'05)*, volume 135(2) of *Electronic Notes in Theoretical Computer Science*, pages 19–30. Elsevier, 2005.

[89] Z. J. Haas, J. Y. Halpern, and L. Li. Gossip-based ad hoc routing. *IEEE/ACM Trans. Netw.*, 14(3):479–491, 2006.

[90] H. Hallal, S. Boroday, A. Petrenko, and A. Ulrich. A formal approach to property testing in causally consistent distributed traces. *Formal Asp. Comput.*, 18(1):63–83, 2006.

[91] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.

[92] M. Haque and S. Ahamed. An omnipresent formal trust model (FTM) for pervasive computing environment. In *Computer Software and Applications Conference, 2007. COMPSAC 2007. 31st Annual International*, volume 1, 2007.

[93] B. R. Haverkort. Markovian models for performance and dependability evaluation. In J.-P. Katoen, H. Brinksma, and H. Hermanns, editors, *Lectures on Formal Methods and Performance Analysis*, volume 2090 of *LNCS*, pages 38–83. Springer-Verlag, 2001.

[94] S. Herring, L. Scheidt, S. Bonus, and E. Wright. Bridging the gap: A genre analysis of weblogs. In *37th Annual HICSS Conference*, Big Island, Hawaii., 2004.

[95] 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 3920 of *LNCS*, pages 441–444. Springer, 2006.

[96] E. Hollnagel, D. D. Woods, and N. Leveson. *Resilience engineering: concepts and precepts*. Ashgate Publishing, Surrey, 2006.

[97] R. A. Howard. *Dynamic Probabilistic Systems: Markov Models*, volume 1 of *Decision and Control*. John Wiley and Sons, New York, 1971.

[98] J. Hughes, V. King, T. Rodden, and H. Andersen. Moving out from the control room: Ethnography in system design. In *CSCW Ś94*, Chapel Hill, North Carolina, 1994.

[99] H. Hutchinson, W. Mackay, and B. Westerlund. Technology probes: Inspiring design for and with families. In *Conference on Human Factors in Computing Systems (CHI 2003)*, pages 17–24. ACM Press, 2003.

[100] P. Inverardi and M. Nesi. Deciding Observational Congruence of Finite-State CCS expressions by Rewriting. *Theor. Comput. Sci.*, 139(1-2):315–354, 1995.

[101] D. Jansen, J.-P. Katoen, M. Oldenkamp, M. Stoelinga, and I. Zapreev. How fast and fat is your probabilistic model checker? an experimental performance comparison. In *Hardware and Software: Verification and Testing, the 3^{rd} International Haifa Verification Conference (HVC 2007)*, LNCS 4899, pages 69–85. Springer, 2007.

[102] B. Jeannet, P. D'Argenio, and K. Larsen. Rapture: A tool for verifying Markov decision processes. In I. Cerna, editor, *Proc. Tools Day, affiliated to 13th Int. Conf. Concurrency Theory (CONCUR'02)*, Technical Report FIMU-RS-2002-05, Faculty of Informatics, Masaryk University, pages 84–98, 2002.

[103] A. Jøsang and S. Pope. Semantic constraints for trust transitivity. In *APCCM: 2nd Asia-Pacific conference on Conceptual modelling*, pages 59–68, Newcastle, New South Wales, Australia, 2005. Australian Computer Society, Inc.

[104] J. Joyce, G. Lomow, K. Slind, and B. Unger. Monitoring distributed systems. *ACM Trans. Comput. Syst.*, 5(2):121–150, 1987.

[105] J.-P. Katoen, E. Hahn, H. Hermanns, D. Jansen, and I. Zapreev. The ins and outs of the probabilistic model checker MRMC. In *Proc. 6th International Conference on Quantitative Evaluation of Systems (QEST'09)*. IEEE CS Press, 2009.

[106] Y. Kim and K. Doh. Trust Type based Semantic Web Services Assessment and Selection. *Proceedings of ICACT, IEEE Computer*, pages 2048–2053, 2008.

[107] T. Kindberg, M. Spasojevic, R. Fleck, and A. Sellen. The ubiquitous camera: An in-depth study of camera phone use. *IEEE Pervasive Computing*, 4(2):42–50, 2005.

[108] H. Koshutanski, A. Lazouski, F. Martinelli, and P. Mori. Enhancing grid security by fine-grained behavioral control and negotiation-based authorization. *Int. J. Inf. Sec.*, 8(4):291–314, 2009.

[109] H. Koshutanski, F. Martinelli, P. Mori, L. Borz, and A. Vaccarelli. A fine grained and x.509 based access control system for globus. In *OTM*, pages 1336–1350. Springer, 2006.

[110] E. Kurvinen. Only when miss universe snatches me: Teasing in mms messaging. In *DPPIŠ03*, pages 98–102, Pittsburgh, Pennsylvania, USA, 2003.

[111] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with PRISM: A hybrid approach. *International Journal on Software Tools for Technology Transfer (STTT)*, 6(2):128–142, 2004.

[112] M. Kwiatkowska, G. Norman, and D. Parker. Stochastic model checking. In M. Bernardo and J. Hillston, editors, *Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM'07)*, volume 4486 of *LNCS (Tutorial Volume)*, pages 220–270. Springer, 2007.

[113] M. Kwiatkowska, G. Norman, and D. Parker. Analysis of a gossip protocol in prism. *ACM SIGMETRICS Performance Evaluation Review*, 36(3):17–22, 2008.

[114] M. Kwiatkowska, G. Norman, and D. Parker. Prism: Probabilistic model checking for performance and reliability analysis. *ACM SIGMETRICS Performance Evaluation Review*, 36(4):40–45, 2009.

[115] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.

[116] J. Laprie. Dependable computing and fault tolerance: concepts and terminology. In *Fault-Tolerant Computing, 1995, ' Highlights from Twenty-Five Years'., Twenty-Fifth International Symposium on*, pages 2+, 1995.

[117] J. Laprie. From dependability to resilience. In *38th IEEE/IFIP Int. Conf. On Dependable Systems and Networks*, 2008.

[118] J.-C. Laprie. Dependable computing: Concepts, limits, challenges. In *25th IEEE Symposium on Fault-Tolerant Computing*, pages 42–54, Pasadena, 1995. IEEE Press.

[119] C. Lenzen, T. Locher, and R. Wattenhofer. Tight bounds for clock synchronization. In *PODC '09: Proceedings of the 28th ACM symposium on Principles of distributed computing*, pages 46–55, New York, NY, USA, 2009. ACM.

[120] J. Liu and V. Issarny. An incentive compatible reputation mechanism for ubiquitous computing environments. *Int. J. Inf. Secur.*, 6(5):297–311, 2007.

[121] J. Liu and V. Issarny. An incentive compatible reputation mechanism for ubiquitous computing environments. *Int. J. Inf. Sec.*, 6(5):297–311, 2007.

[122] P. Lollini, A. Bondavalli, and F. Di Giandomenico. A modeling methodology for hierarchical control system and its application. *Journal of the Brazilian Computer Society (JBCS)*, 10, N. 3:57–69, April 2005.

[123] P. Lollini, A. Bondavalli, and F. Di Giandomenico. A decomposition-based modeling framework for complex systems. *IEEE Transactions on Reliability*, 58, N. 1:20–33, March 2009.

[124] A. Lomuscio and W. Penczek. Ldyis: a framework for model checking security protocols. *Fundamenta Informaticae*, 85(1-4):359–375, 2008.

[125] A. Lomuscio, H. Qu, and F. Raimondi. MCMAS: A model checker for the verification of multi-agent systems. In *Proceedings of CAV 2009*, LNCS 5643, pages 682–688. Springer, 2009.

[126] A. Lomuscio, H. Qu, M. J. Sergot, and M. Solanki. Verifying temporal and epistemic properties of web service compositions. In *ICSOC*, volume 4749 of *LNCS*, pages 456–461. Springer-Verlag, 2007.

[127] A. Lomuscio and M. Sergot. Deontic interpreted systems. *Studia Logica*, 75(1):63–92, 2003.

[128] A. Lomusico, H. Qu, and M. Solanki. Towards verifying contract regulated service composition. In *IEEE International Conference on Web Services (ICWS 2008)*, 2008.

[129] N. Luhmann. *Trust and Power*. Chichester Wiley, 1979.

[130] N. Luhmann. Familiarity, confidence, trust: problems and alternatives. In D. Gambetta, editor, *Trust: Making and Breaking Co-operative Relations*, pages 94–107. Department of Sociology, University of Oxford, 1990.

[131] Z. Malik and A. Bouguettaya. Reputation Bootstrapping for Trust Establishment among Web Services. *Internet Computing, IEEE*, 13(1):40–47, Jan.-Feb. 2009.

[132] D. Manchala. Trust metrics, models and protocols for electronic commerce transactions. In *International Conference on Distributed Computing Systems*, volume 18, pages 312–321. IEEE Computer Society, 1998.

[133] M. Mansouri-Samani and M. Sloman. Monitoring distributed systems. pages 303–347, 1994.

[134] M. Mansouri-Samani and M. Sloman. GEM: a generalized event monitoring language for distributed systems. *Distributed Systems Engineering*, 4(2):96–108, 1997.

[135] S. Marsh. *Formalising Trust as a Computational Concept*. PhD thesis, University of Stirling, Scotland, 1994.

[136] F. Martinelli and M. Petrocchi. A uniform framework for security and trust modeling and analysis with crypto-ccs. *Electron. Notes Theor. Comput. Sci.*, 186:85–99, 2007.

[137] P. M. Masci, S. Chiaradonna, and F. D. Giandomenico. Dependability analysis of diffusion protocols in wireless networks with heterogeneous node capabilities. Technical report, 2009.

[138] M. L. Massie, B. N. Chun, and D. E. Culler. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(7):817 – 840, 2004.

[139] R. McDougall, J. Mauro, and B. Gregg. *Solaris(TM) Performance and Tools: DTrace and MDB Techniques for Solaris 10 and OpenSolaris (Solaris Series)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2006.

[140] J. F. Meyer. Performability: A retrospective and some pointers to the future. *Perform. Eval.*, 14(3-4):139–156, 1992.

[141] M. K. Molloy. Performance analysis using stochastic Petri nets. *IEEE Transactions on Computers*, 31(9):913–917, 1982.

[142] R. J. Moore. Dynamic probes and generalised kernel hooks interface for linux. In *Proceedings of the 4th annual Linux Showcase & Conference - Volume 4*, pages 35–35, Berkeley, CA, USA, 2000. USENIX Association.

[143] A. Movaghar and J. F. Meyer. Performability modelling with stochastic activity networks. In *1984 Real-Time Systems Symposium*, pages 215–224, Austin, TX, December 1984. IEEE Computer Society Press.

[144] J. K. Muppala, R. M. Fricks, and K. S. Trivedi. Techniques for system dependability evaluation. In W. K. Grassmann, editor, *Computational Probability, Vol. 24 of Operations Research and Management Science*, pages 445–480. Kluwer Academic Publishers, The Netherlands, 2000.

[145] O. Naím and A. Hey. Invasiveness of performance instrumentation measurements on multiprocessors, 1994.

[146] S. Naqvi, P. Massonet, B. Aziz, A. Arenas, F. Martinelli, P. Mori, L. Blasi, and G. Cortese. Fine-Grained Continuous Usage Control of Service Based Grids - The GridTrust Approach. In *ServiceWave '08: Proceedings of the 1st European Conference on Towards a Service-Based Internet*, pages 242–253, Berlin, Heidelberg, 2008. Springer-Verlag.

[147] B. Nardi, D. Schiano, and M. Gumbrecht. Blogging as social activity, or, would you let 900 million people read your diary? In *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, Chicago, Illinois, USA, November 2004.

[148] S. Nepal, Z. Malik, and A. Bouguettaya. Reputation Propagation in Composite Services. In *Proceedings of the 2009 IEEE International Conference on Web Services-Volume 00*, pages 295–302. IEEE Computer Society, 2009.

[149] D. M. Nicol, W. H. Sanders, and K. S. Trivedi. Model-based evaluation: from dependability to security. *IEEE Transactions on Dependable and Secure Computing*, 1:48–65, January-March 2004.

[150] G. Norman and V. Shmatikov. Analysis of probabilistic contract signing. *Journal of Computer Security*, 14(6):561–589, 2006.

[151] P. Nurmi. A bayesian framework for online reputation systems. In *Telecommunications, 2006. AICT-ICIW '06. International Conference on Internet and Web Applications and Services/Advanced International Conference on*, pages 121–121, Feb. 2006.

[152] A. Nusayr. Aop as a formal framework for runtime monitoring. In *Proceedings of the 2008 Foundations of Software Engineering Doctoral Symposium*, pages 25–28, New York, NY, USA, 2008. ACM.

[153] T. Ojala, J. Korhonen, T. Sutinen, P. Parhi, and L. Aalto. Mobile karpat Ű a case study in wireless personal area networking. In *MUM 2004*, College Park, Maryland USA, 2004. ACM.

[154] D. Okabe. Emergent social practices, situations and relations through everyday camera phone use. In *Mobile Communication and Social Change, the 2004 International Conference on Mobile Communication*, Seoul, Korea, October 2004.

[155] S. Paradesi, P. Doshi, and S. Swaika. Integrating Behavioral Trust in Web Service Compositions. In *Proceedings of the 2009 IEEE International Conference on Web Services*, pages 453–460. IEEE Computer Society, 2009.

[156] D. Parker. *Implementation of Symbolic Model Checking for Probabilistic Systems*. PhD thesis, University of Birmingham, 2002.

[157] N. W. Paton and O. Díaz. Active database systems. *ACM Comput. Surv.*, 31:63–103, March 1999.

[158] F. Perich, J. Undercoffer, L. Kagal, A. Joshi, T. Finin, and Y. Yesha. In reputation we believe: Query processing in mobile ad-hoc networks. *IEEE International Conference on Mobile and Ubiquitous Systems: Networking and Services*, pages 326–334, 2004.

[159] P. R. Pietzuch, B. Shand, and J. Bacon. A framework for event composition in distributed systems. In *Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware*, pages 62–82, New York, NY, USA, 2003. Springer-Verlag New York, Inc.

[160] K. Plummer. *Documents of Life*. London: George Allen & Unwin., 1983.

[161] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science*, pages 45–57, 1977.

[162] S. Porcarelli, , F. Di Giandomenico, A. Bondavalli, M. Barbera, and I. Mura. Service level availability estimation of gprs. *IEEE Transactions on Mobile Computing*, 2, N. 3:233–247, 2003.

[163] A. Rahman and S. Hailes. Supporting trust in virtual communities. *IEEE Hawaii International Conference on System Sciences*, page 6007, 2000.

[164] F. Raimondi and A. Lomuscio. Automatic verification of multi-agent systems by model checking via OBDDs. *Journal of Applied Logic*, 5(2):235–251, 2005.

[165] B. Randell. Facing up to faults. *Computer J.*, 45(2):95–106, 2000.

[166] J. Rasmussen. *New Technology and Human Error*, chapter The definition of human error and a taxonomy for technical system design. John Wiley and Sons: Chichester, 1987.

[167] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, New York, NY, USA, 2001. ACM Press.

[168] J. Reason. *Human Error*. Cambridge University Press, 1990.

[169] H. K. P. M. Repo, P. and T. P. Mobile video. Technical Report 2003:5, National Consumer Research Centre, 2003.

[170] ReSIST NOE. Resilience for survivability in ist, 2006.

[171] M. Riedl, J. Schuster, and M. Siegle. Recent extensions to the stochastic process algebra tool CASPA. In *Proc. 5th International Conference on Quantitative Evaluation of Systems (QEST'08)*, pages 113–114. IEEE CS Press, 2008.

[172] F. Romani, S. Chiaradonna, F. Di Giandomenico, and L. Simoncini. Simulation models and implementation of a simulator for the performability analysis of electric power systems considering interdependencies. In *10th IEEE High Assurance Systems Engineering Symposium (HASE'07)*, pages 305–312, Dallas, Texas, November 2007. IEEE Computer Society Press.

[173] R. Saadi, J. M. Pierson, and L. Brunie. T2D: A Peer to Peer trust management system based on Disposition to Trust. In *25th ACM Symposium On Applied Computing (SAC)*. ACM Press, 2010 (to be appear).

[174] W. H. Sanders and L. M. Malhis. Dependability evaluation using composed SAN-based reward models. *Journal of Parallel and Distributed Computing*, 15(3):238–254, 1992.

[175] W. H. Sanders and J. F. Meyer. A unified approach for specifying measures of performance, dependability, and performability. *Dependable Computing and Fault-Tolerant Systems: Dependable Computing for Critical Applications*, 4:215–237, 1991.

[176] W. H. Sanders and J. F. Meyer. Stochastic activity networks: formal definitions and concepts. pages 315–343, 2002.

[177] J. Sandhu. *Universal Design*, chapter Multi-Dimensional Evaluation as a tool in Teaching Universal Design. Hausbanken, Norway, 2002.

[178] D. C. Schmidt. Guest editor's introduction: Model-driven engineering. *Computer*, 39:25–31, 2006.

[179] B. P. Shah. Analytic solution of stochastic activity networks with exponential and deterministic activities. Master's thesis, University of Arizona, USA, 1993.

[180] R. Silverstone. *Television and Everyday Life,*. London, Routledge, 1994.

[181] C. Sodergard. Mobile television - technology and user experiences, report on the mobile tv project. Technical report, VTT Publications, 2003.

[182] S. Song, K. Hwang, R. Zhou, and Y. Kwok. Trusted P2P transactions with fuzzy reputation aggregation. *IEEE Internet Computing*, 9(6):24–34, 2005.

[183] R. Spalazzese, P. Inverardi, and V. Issarny. Towards a formalization of mediating connectors for on the fly interoperability. In *Proceedings of the Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture (WICSA/ECSA 2009)*, pages 345–348, 2009.

[184] M. Spezialetti and J. P. Kearns. A General Approach to Recognizing Event Occurences in Distributed Computations. In *ICDCS*, pages 300–307, 1988.

[185] W. J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton, 1994.

[186] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 149–160, New York, NY, USA, 2001. ACM Press.

[187] T. Suganuma, T. Yasue, and T. Nakatani. An empirical study of method in-lining for a java just-in-time compiler. In *Proceedings of the 2nd Java™ Virtual Machine Research and Technology Symposium*, pages 91–104, Berkeley, CA, USA, 2002. USENIX Association.

[188] A. Taylor and R. Harper. Age-old practices in the Śnew worldŠ: a study of gift-giving between teenage mobile phone users. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 439–446. . ACM Press, 2002.

[189] G. Theodorakopoulos and J. S. Baras. Trust evaluation in ad-hoc networks. In *3rd ACM workshop on Wireless security*, pages 1–10, New York, NY, USA, 2004. ACM Press.

[190] M. Tribastone. The PEPA plug-in project. In *Proc. 4th International Conference on Quantitative Evaluation of Systems (QEST'07)*, pages 53–54. IEEE Computer Society, 2007.

[191] K. S. Trivedi. *Probability and Statistics with Reliability, Queueing and Computer Science Applications*. John Wiley & Sons, New York, second edition, 2002.

[192] J. J. P. Tsai, K. Y. Fang, and H. Y. Chen. A noninvasive architecture to monitor real-time distributed systems . *Computer*, 23(3):11–23, Mar 1990.

[193] L. Viljanen. Towards an ontology of trust. In S. K. Katsikas, J. Lopez, and G. Pernul, editors, *TrustBus*, volume 3592 of *Lecture Notes in Computer Science*, pages 175–184. Springer, 2005.

[194] Wikipedia. Beta distribution. http://en.wikipedia.org/wiki/Beta_distribution.

[195] L. Xiong and L. Liu. A reputation-based trust model for peer-to-peer ecommerce communities. In *4th ACM conference on Electronic commerce*, pages 228–229, 2003.

[196] H. Younes. Ymer: A statistical model checker. In *Proc. 17th International Conference on Computer Aided Verification (CAV'05)*, volume 3576 of *LNCS*, pages 429–433. Springer, 2005.

[197] R. Zhou, K. Hwang, and M. Cai. Gossiptrust for fast reputation aggregation in peer-to-peer networks. *IEEE Transactions on Knowledge and Data Engineering*, pages 1282–1295, 2008.

[198] C. Ziegler and J. Golbeck. Investigating correlations of trust and interest similarity-do birds of a feather really flock together. *Decision Support Systems*, 2005.

[199] D. Zimmer. On the semantics of complex events in active database management systems. In *Proceedings of the 15th International Conference on Data Engineering*, pages 392–, Washington, DC, USA, 1999. IEEE Computer Society.

[200] P. R. Zimmermann. *The official PGP user's guide*. MIT Press, Cambridge, MA, USA, 1995.