Revised reference model

J. Arlat, M. Kaâniche, A. Bondavalli,
M. Calha, A. Casimiro, A. Daidone,
L. Falai, G. Huszerl, M-O. Killijian,
A. Kövi, Y. Liu, P. Lollini,
E.V. Matthiesen, M. Radimirsch, T. Renier,
N. Rivière, M. Roy, H-P. Schwefel,
I-E. Svinnset, H. Waeselynck

DI–FCUL TR–07–20

September 2007

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

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

Project no.: **IST-FP6-STREP- 26979**

Project full title: **Highly dependable ip-based networks and services**

Project Acronym: **HIDENETS**

Deliverable no.: D1.2

Title of the deliverable: Revised reference model

Contractual Date of Delivery to the CEC:	30th June 2007
Actual Date of Delivery to the CEC:	30th June 2007
Organisation name of lead contractor for this deliverable	LAAS-CNRS
Authors: Jean Arlat and Mohamed Kaâniche (Editors), Andrea Bondavalli, Mario Calha, Antonio Casimiro, Alessandro Daidone, Lorenzo Falai, Gabör Huszerl, Marc-Olivier Killijian, András Kövi, Yaoda Liu, Paolo Lollini, Erling Vestergaard Matthiesen, Markus Radimirsch, Thibault Julien Renier, Nicolas Rivière, Matthieu Roy, Hans-Peter Schwefel, Inge-Einar Svinnset, Hélène Waeselynck	
Participants: AAU, BME, Carmeq, FCUL, LAAS-CNRS, Telenor, UniFi	
Work package contributing to the deliverable:	WP1
Nature:	R
Version:	2.0
Total number of pages:	86
Start date of project:	1st Jan. 2006 **Duration:** 36 month

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination Level		
PU	Public	**X**
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Abstract:

This document contains an update of the HIDENETS Reference Model, whose preliminary version was introduced in D1.1. The Reference Model contains the overall approach to development and assessment of end-to-end resilience solutions. As such, it presents a framework, which due to its abstraction level is not only restricted to the HIDENETS car-to-car and car-to-infrastructure applications and use-cases.

Starting from a condensed summary of the used dependability terminology, the network architecture containing the *ad hoc* and infrastructure domain and the definition of the main networking elements together with the software architecture of the mobile nodes is presented. The concept of architectural hybridization and its inclusion in HIDENETS-like dependability solutions is described subsequently. A set of communication and middleware level services following the architecture hybridization concept and motivated by the dependability and resilience challenges raised by HIDENETS-like scenarios is then described.

Besides architecture solutions, the reference model addresses the assessment of dependability solutions in HIDENETS-like scenarios using quantitative evaluations, realized by a combination of top-down and bottom-up modelling, as well as verification via test scenarios. In order to allow for fault prevention in the software development phase of HIDENETS-like applications, generic UML-based modelling approaches with focus on dependability related aspects are described.

The HIDENETS reference model provides the framework in which the detailed solution in the HIDENETS project are being developed, while at the same time facilitating the same task for non-vehicular scenarios and applications

Keyword list:

Reference model, network and node architectures, middleware-level and communication-level services, dependability and performance assessment (evaluation and testing), design methodologies, etc.

Version information

Version	Date	Comments
0.0	07.02.2007	Table of Contents
0.1	05.03.2007	Integration of material from WPs and broadcasting to WP1 list
0.15	10.04.2007	Integration of all material received and broadcasting within LAAS
0.2	30.04.07	Document restructuring and integration of material received since previous release
0.3	09.05.07	Updating of document by integration of new subsections according to new structuring
0.4	23.05.07	Integration and consolidation of inputs received, inclusion of a list of abbreviations
0.45	24.05.07	Partial update (including major structural changes) for final changes integration
0.5	25.05.07	Comprehensive editing work
1.0	13.06.07	First Revision according to Review team 1 and Advisory Board comments
1.5	20.06.07	Second Revision according to Review team 2 and additional Advisory Board comments
2.0	22.06.07	Final version

Table of Content

Bibliography

[1] ITU-T Recommendation X.200 (1994) | ISO/IEC 7498-1:1994, Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model (and corresponding references therein).

[2] ITU-T Rec. X.901 | ISO/IEC 10746-1: Information technology — Open Distributed Processing — Reference model: Overview (and corresponding references therein).

[3] A. Avizienis, J.C. Laprie, "Dependable computing: from concepts to design diversity", Proceedings of the IEEE, vol. 74, no. 5, May 1986, pp. 629-638.

[4] A. Avizienis, J.C. Laprie, B. Randell, C. Landwer, "Basic Concepts and Taxonomy of Dependable and Secure Computing", IEEE Transactions on Dependable and Secure Computing, vol. 1, no. 1, January-March 2004, pp. 11-33.

[5] W.C. Carter, "A time for reflection", in Proc. 12th IEEE Int. Symp. on Fault Tolerant Computing (FTCS-12), Santa Monica, California, June 1982, p. 41.

[6] J.C. Laprie, A. Costes, "Dependability: a unifying concept for reliable computing", Proc. 12th IEEE Int. Symp. on Fault Tolerant Computing (FTCS-12), Santa Monica, California, June 1982, pp. 18-21.

[7] J.-C. Laprie (Ed.), Dependability: Basic Concepts and Terminology, Springer-Verlag, Vienna, 1992.

[8] IEEE 802.11 WG, "Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specification", IEEE 1999.

[9] IEEE 802.11 WG, "Draft Supplement to Part 11: Wireless Medium Access Control (MAC) and physical layer (PHY) specifications: Medium Access Control (MAC) Enhancements for Quality of Service (QoS)", IEEE 802.11e/D13.0, Jan. 2005.

[10] J. Moy, "OSPF Version 2". IETF RFC 2328 (STD 54), April 1998.

[11] R.W. Callon, "Use of OSI IS-IS for routing in TCP/IP and dual environments". IETF RFC 1195, December 1990

[12] Y. Rekhter. "A Border Gateway Protocol 4 (BGP-4)". IETF RFC 4271, January 2006.

[13] T. Clausen, P. Jacquet, "Optimized Link State Routing Protocol (OLSR)". IETF RFC 3626, October 2003

[14] P. Spagnolo et al. "OSPFv2 Wireless Interface Type". Internet draft 'draft-spagnolo-manet-ospf-wireless-interface-01, May 2004.

[15] M. Chandra, "Extensions to OSPF to Support Mobile Ad Hoc Networking". Internet draft 'draft-chandra-ospf-manet-ext-02', October 2004.

[16] C. Perkins, E. Belding-Royer, S.Das, "Ad hoc On-Demand Distance Vector (AODV) Routing". IETF RFC 3561, July 2003.

[17] X.Y. Li and I. Stojmenovic, "Broadcasting and topology control in wireless ad hoc networks", in Handbook of Algorithms for Mobile and Wireless Networking and Computing, (A. Boukerche and I. Chlamtac, eds.), CRC Press, to appear.

[18] X. Chen and J. Wu, Multicasting techniques in mobile ad hoc networks, in The handbook of ad hoc wireless networks, CRC presss. Pages 25-40, 2003.

[19] I. Stojmenovic, Geocasting in ad hoc and sensor networks, in Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless and Peer-to-Peer Networks (Jie Wu, ed.), Auerbach Publications (Taylor & Francis Group), 2006, 79-97.

[20] C. Perkins, "IP Mobility Support for Ipv4", IETF RFC 3344, August 2002

[21] J. Rosenberg, et al., "Session Initiation Protocol", IETF RFC 3261, June 2002

[22] R. Stewart,, et al., "Stream Control Transmission Protocol", IETF RFC 2960, Oct. 2000

[23] E. Perera, V. Sivaraman, and A. Seneviratne, "Survey on Network Mobility Support", ACM SIGMOBILE Mobile Computing and Communications Review, 8(2):7-19, Apr 2004.

[24] A. Autenrieth, A. Kirstädter "Fault Tolerance and Resilience Issues in IP-Based Networks", Second International Workshop on the Design of Reliable Communication Networks (DRCN2000), Munich, Germany, April 9-12, 2000

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

[26] T. Chandra, S. Toueg. Unreliable failure detectors for reliable distributed systems. Journal of the ACM, 43(2):225–267, March 1996.

[27] IEEE 802.11p draft standard, http://www.ieee802.org/11/Reports/tgp_update.htm

[28] IEEE 802.11s draft standard, http://www.ieee802.org/11/Reports/tgs_update.htm

[29] 3GPP TS 23002-710: "Network Architecture", V7.1.0, March 2006

[30] B Aboba et al., "*Link-local Multicast Name Resolution (LLMNR)*", < draft-ietf-dnsext-mdns-47.txt >, August 2006

[31] ETSI ES 282 003: "*Resource and Admission Control Sub-system (RACS); Functional Architecture*", Release 2, September 2006.

[32] I-E. Svinnset et al, "*Report on resilient topologies and routing (preliminary version)*", EU FP6 IST project HIDENETS, deliverable D3.1.1 December 2006

[33] S. Rank, HP Schwefel: "Transient analysis of RED queues: a quantitative analysis of buffer-occupancy fluctuations and relevant time-scales", Performance Evaluation 63, pp. 725-742, 2006.

[34] HP Schwefel, L. Lipsky, M. Jobmann "On the Necessity of Transient Performance Analysis in Telecommunication Networks". In Souza, Fonseca, Silva (eds.), "Teletraffic Engineering in the Internet Era", pp. 1087-1099. Elsevier, 2001.

[35] K. Nagel and D. E. Wolf and P. Wagner and P. Simon, "*Two-lane traffic rules for cellular automata: {A} systematic approach*" LA-UR 97-4706, Los Alamos, 1997.

[36] K. Nagel and M. Schreckenberg, "A cellular automaton model for freeway traffic", Journal de Physique pp 2221-2229, September 1992

[37] D. M. Nicol, W. H. Sanders, K. S. Trivedi, "Model-based Evaluation: From Dependability to Security. IEEE Transactions on Dependable and Secure Computing, Vol. 1, No. 1, pp 48-65, 2004.

[38] K. Kanoun et al., http://www.laas.fr/DBench, Project Reports section, project full final report.

[39] K. Kanoun, Y. Crouzet, A. Kalakech, A. E. Rugina and P. Rumeau, "Benchmarking the Dependability of Windows and Linux using Postmark Workloads", in *Proc. 16th IEEE Int. Symp. on Software Reliability Engineering (ISSRE 2005)*, (Chicago, IL, USA), pp.11-20, IEEE CS Press, 2005.

[40] I. Majzik, A. Pataricza, A. Bondavalli: Stochastic Dependability Analysis of System Architecture based on UML Models. In R. de Lemos, C. Gacek, A. Romanovsky: Architecting Dependable Systems. LNCS-2677, pp 219-244, Springer Verlag, Berlin, 2003.

[41] P. Lollini, "On the modeling and solution of complex systems: from two domain-specific case-studies towards the definition of a more general framework". PhD Thesis, University of Florence, Computer Science Department, 2005.

[42] S. Zuyev, F. Bacelli and K. Tchoutmatchenko, "*Markov paths on the Poisson-Delaunay graph with applications to routeing in mobile networks*". Adv. Appl. Probab., 32(1), 1-18, 2000

[43] S. Zuyev, F. Baccelli, M. Klein and M. Leborges "*Stochastic geometry and architecture of communication networks*". Journal of Telecommunication Systems, 7, 209-227, 1997

[44] RL Olsen, MB Hansen, HP Schwefel: 'Quantitative analysis of access strategies to remote information in network services', Proceedings of IEEE GLOBECOM, November 2006

[45] I. Mura and A. Bondavalli, "Markov Regenerative Stochastic Petri Nets to Model and Evaluate the Dependability of Phased Missions", IEEE Transactions on Computers, 50(12): 1337-1351, 2001.

[46] Object Management Group, "*UML Profile for Schedulability, Performance, and Time"*. Final adopted specification. http://www.omg.org/, 2001

[47] A. Klar, R. Kuehne and R. Wegener, "*Mathematical models for vehicular traffic*", Surv. Math. Ind. pp 215, 1996.

[48] F. Bause, P. Buchholz, and P. Kemper. A toolbox for functional and quantitative analysis of deds. In *Lecture Notes in Computer Science*, number 1469, pages 356.359. R. Puigjaner, N. N. Savino, and B. Serra, 1998.

[49] C. Betous-Almeida, and K. Kanoun, "*Stepwise construction and refinement of dependability models*". In Proc. IEEE International Conference on Dependable Systems and Networks DSN 2002, Washington D.C., 2002.

[50] A. Bondavalli, M. Dal Cin, D. Latella, I. Majzik, A. Pataricza and G. Savoia: Dependability Analysis in the Early Phases of UML Based System Design. International Journal of Computer Systems - Science & Engineering, Vol. 16 (5), Sep 2001, pp. 265-275.

[51] P. Lollini, A. Bondavalli et al., "Evaluation methodologies, techniques and tools (preliminary version)", EU FP6 IST project HIDENETS, deliverable D4.1.1. December 2006.

[52] G. A. Di Caro, *Analysis of simulation environments for mobile ad hoc networks*, Technical Report No IDSIA-24-03, Dalle Molle Institute for Artificial Intelligence, Switzerland, December 2003

[53] L. Falai and A. Bondavalli. Experimental evaluation of the QoS of Failure Detectors on Wide Area Network. Proceedings of the International Conference on Dependable Systems and Networks (DSN 2005). 2005.

[54] T.H. Tse, Stephan S. Yau, W.K. Chan, Heng Lu. Testing Context-Sensitive Middleware-Based Software Applications, Proceedings of the 28th Annual International Computer Software and Application Conference (COMPSAC 2004), pp.458-466, IEEE CS Press, 2004.

[55] Satyajit Achrya, Chris George, Hrushikesha Mohanty. Specifying a Mobile Computing Infrastructure and Services, 1st International Conference on Distributed Computing and Internet Technology (ICDCIT 2004), LNCS 3347, pp.244-254, Springer-Verlag Berlin Heidelberg, 2004

[56] Satyajit Acharya, Hrushikesha Mohanty, R.K Shyamasundar. MOBICHARTS: A Notation to Specify Mobile Computing Applications. Proceedings of the 36th Hawaii International Conference on System Sciences (HICSS'03), IEEE CS Press, 2003.

[57] Vincenzo Grassi, Raffaela Mirandola, Antonino Sabetta. A UML Profile to Model Mobile System, UML 2004,

[58] Hubert Baumeister et al. UML for Global Computing. Global Computing: Programming Environments, Languages, Security, and Analyisis of Systems, GC 2003, LNCS 2874, pp. 1-24, Springer-Verlag Berlin Heidelberg, 2003

[59] F. Ngani Noudem and C. Viho. "Modeling, Verifying and Testing the Mobility Management In the Mobile Ipv6 Protocol," 8th International Conference on Telecommunications (ConTEL 2005), Vol.2, pp. 619-626, IEEE CS Press, 2005.

[60] A. Cavalli et al. "A validation Model for the DSR protocol, " in Proc. of the 24th International Conference on Distributed Computing Systems Workshops (ICDCSW'04), pp.768-773, IEEE CS Press, 2004

[61] W.K. Chan, T.Y. Chen, Heng Lu. A Metamorphic Approach to Integration Testing of Context-Sensitive Middleware-Based Applications, Proceedings of the 5th International Conference on Quality Software (QSIC'05), pp.241-249, IEEE CS Press, 2005

[62] Karl R.P.H Leung, Joseph K-Y Ng, W.L. Yeung. Embedded Program Testing in Untestable Mobile Environment: An Experience of Trustworthiness Approach, Proceedings of the 11th Asia-Pacific Software Engineering Conference, pp.430-437, IEEE CS Press, 2004

[63] de Bruin, D.; Kroon, J.; van Klaverem, R.: Nelisse, M.. Design and test of a cooperative adaptive cruise control system, Intelligent Vehicles symposium, pp.392-396, IEEE CS Press, 2004

[64] Christoph Schroth et al. Simulating the traffic effects of vehicle-to-vehicle messaging systems, Proceedings of ITS Telecommunication, 2005

[65] Ricardo Morla, Nigel Davies. Evaluating a Location-Based Application: A Hybrid Test and Simulation Environment, IEEE Pervasive computing, Vol.3, No.2, pp.48-56, July-September 2004

[66] Rimon Barr, Zygmunt J. Haas, Robbert van Renesse. Scalable Wireless Ad hoc Network Simulation. Handbook on Theoretical and Algorithmic Aspects of Sensor, Ad hoc Wireless, and Peer-to-Peer Networks, ch. 19, pp. 297-311, CRC Press, 2005

[67] J.Barton, V. Vijayaragharan. Ubiwise: A Simulator for Ubiquitous Computing Systems Design, Technical report HPL-2003-93, Hewlett-Packard Labs, 2003

[68] Kumaresan Sanmiglingam, Geogre Coulouris. A Generic Location Event Simulator, UbiComp 2002, LNCS 2498, pp.308-315, Springer-Verlag Berlin Heidelberg, 2002

[69] P. Thévenod-Fosse, H. Waeselynck and Y. Crouzet, “Software statistical testing”, in Predictably Dependable Computing Systems, Springer Verlag, pp. 253-272, 1995

[70] P. Veríssimo. Travelling through Wormholes: a new look at Distributed Systems Models, ACM SIGACT news (ACM Special Interest Group on Automata and Computability Theory), 37(1):66-81, 2006.

[71] T. Chandra, V. Hadzilacos, S. Toueg, and B. Charron-Bost. On the impossibility of group membership. In Proceedings of the 15th ACM Symposium on Principles of Distributed Computing, pages 322–330,May 1996.

[72] E. Anceaume, B. Charron-Bost, P. Minet, and S. Toueg. On the formal specification of group membership services. Technical Report RR-2695, INRIA, Rocquencourt, France, November 1995.

[73] I. de Bruin et al., “Specification HIDENETS laboratory set-up scenario and components”, EU FP6 IST project HIDENETS, deliverable D6.1. March 2007.

[74] Flaviu Cristian, Christof Fetzer. The timed asynchronous system model. In Proceedings of the 28th Annual International Symposium on Fault-Tolerant Computing, pp.140-149, Munich, Germany, June 1998. IEEE CS Press.

[75] Paulo Veríssimo, António Casimiro. The timely computing base model and architecture. IEEE Transactions on Computers, 51(8):916–930, 2002.

[76] M. Radimirsch et al., “Use case scenarios and preliminary reference model”, EU FP6 IST project HIDENETS, deliverable D1.1. September 2006.

[77] S. Lee, R. Sherwood, B. Bhattacharjee. "Cooperative peer groups in NICE". In INFOCOM'03, April 2003.

[78] N. Asokan, M. Schunter, and M. Waidner. Optimistic Protocols for Fair Exchange. In Proceedings of the 4th ACM Conference on Computer and Communications Security, Zurich, April 1997.

[79] M.-O. Killijian, R. Cunningham, R. Meier, L. Mazare, and V. Cahill, "Towards Group Communication for Mobile Participants”, presented at Principles of Mobile Computing (POMC'2001), Newport, Rhode Island, USA, 2001.

[80] L. Courtes, O. Hamouda, M. Kaaniche, M.-O. Killijian, D. Powell, “Assessment of cooperative backup strategies for mobile devices”, LAAS report #06817.

[81] W. K. Lin, D. M. Chiu, Y. B. Lee. Erasure Code Replication Revisited. In Proc. of the 4th P2P, pp. 90–97, 2004.

[82] M. Mitzenmacher. Digital Fountains: A Survey and Look Forward. In Proc. of the IEEE Information Theory Workshop, pp. 271–276, October 2004.

[83] H. Weatherspoon, J. Kubiatowicz. Erasure Coding vs. Replication: A Quantitative Comparison. In Revised Papers from the 1st Int. Workshop on Peer-to-Peer Systems, pp. 328–338, Springer-Verlag, 2002.

[84] L. Xu. Hydra: A Platform for Survivable and Secure Data Storage Systems. In Proc. of the ACM Workshop on Storage Security and Survivability, pp. 108–114, ACM Press, November 2005.

[85] L. Xu, V. Bohossian, J. Bruck, D. G. Wagner. Low Density MDS Codes and Factors of Complete Graphs. In IEEE Transactions on Information Theory, 45(1), November 1999, pp. 1817–1826.

[86] Y. Deswarte, L. Blain, J-C. Fabre. Intrusion Tolerance in Distributed Computing Systems. In Proc. of the IEEE Symp. on Research in Security and Privacy, pp. 110–121, May 1991.

[87] The SUMO open source traffic simulation package, http://sumo.sourceforge.net.

[88] Q. Huang, C. Julien, G. Roman. Relying on Safe Distance to Achieve Strong Partitionable Group Membership in Ad Hoc Networks. In IEEE Transactions on Mobile Computing, 3 (2), April 2004, pp. 192-205.

[89] H. Yu and A. Vahdat, “Design and Evaluation of a Continuous Consistency Model for Replicated Services,” Proc. Fourth Symp. Operating Systems Design and Implementation, Oct. 2000.

[90] H. Waeselynck, Z. Micskei, M.D. N’Guyen, N. Rivière, “Mobile Systems from a Validation Perspective: A Case Study”, *Proc. 6th International Symposium on Parallel and Distributed Computing (ISPDC’2007)*, Hagenberg, Austria, July 5-8, 2007, (to appear).

Abbreviations

AO: Authentication Oracle

API: Application Programming Interface

C2C: Car-to-Car

C2I: Car-to-Infrastructure

CA: Certification Authority

CAC: Connection Admission Control

COTS: Commercial Off-The-Shelf

CRC: Cyclic Redundancy Coding

DM: Diagnostic Manager

DoS: Denial of Service

ETSI: European Telecommunications Standards Institute

GMP: Group Membership Protocol

GPRS: General Packet Radio Service

GPS: Global Positioning System

GSPN: Generalized Stochastic Petri Nets

HW: Hardware

IEEE: Institute of Electrical and Electronics Engineers

IFIP: International Federation of Information Processing

IM: Intermediate Model

IMS: IP Multimedia Subsystem

IP: Internet Protocol

ISO: International Organization for Standardization

J2SE: Java 2 Standard Edition

JVM: Java Virtual Machine

LLC: Logical Link Control

MAC: Medium Access Control

MIP: Mobile IP

MSC: Message Sequence Chart

MW: Middleware

NeMo: Network Mobility

ODP: Open Distributed Processing

OS: Operating System

OSI: Open System Interconnection

OTS: Off-the-Shelf

PCO: Points of Control and Observation

PDA: Personal Digital Assistant

PLCP: Physical Layer Convergence Protocol

PKI: Public-Key Infrastructure

QoS: Quality of Service

RACS: Resource and Admission Control Subsystem

RM: Reference Model

RecM: Reconfiguration Manager

RepM: Replication Manager

RTP: Real-time Transport Protocol

R&SA Clock: Reliable and Self-Aware Clock

SAF: Service Availability Forum

SCTP: Stream Control Transmission Protocol

SDL: Specification and Design Language

SINR: Signal to Interference-plus-Noise Ratio

SIP: Session Initiation Protocol

SNMP: Simple Network Management Protocol

SRN: Stochastic Reward Nets

SW: Software

TCO: Trust and Cooperation Oracle

TCP: Transmission Control Protocol

TPH: Tamper Proof Hardware

TTP: Trusted Third Party

UDP: User Datagram Protocol

UML: Unified Modelling Language

UMTS: Universal Mobile Terrestrial Access

VoIP: Voice on IP

V&V: Verification and Validation

WIMAX: Worldwide Interoperability for Microwave Access

WLAN: Wireless Local Area Network

1. Executive Summary

HIDENETS addresses the provision of available and resilient distributed applications and mobile services in highly dynamic environments characterized by unreliable communications and components due to the occurrence of accidental and malicious faults (attacks and intrusions). Our investigations include networking scenarios consisting of *ad hoc*/wireless multi-hop domains as well as infrastructure network domains. Applications and use case scenarios from the automotive domain, based on car-to-car communications with additional infrastructure support are used as driving examples to identify the key features (challenges, threats, and resilience requirements) that are relevant in the context of the project. Based on these features, the project aims at developing appropriate fault tolerance mechanisms, at the middleware and communication layers, as well as methodologies to support their evaluation and testing.

The HIDENETS Reference Model synthesizes the main solutions that are promoted by the project for the design, development support, evaluation and testing of resilient mobile and *ad hoc* based applications and services, based on the results and achievements obtained in the course of the project. The terminology from the dependability related community is used as a starting point for the concepts contained in this Reference Model. Both the terminology as well as the proposed technical solutions aim to have some generic applicability, namely they are meant to be applicable beyond the context of car-to-car and automotive scenarios, applications, and use-cases. Tailoring to a specific system development is always required, i.e., the reference model contains the above elements only to a certain degree of concreteness.

Figure 1 illustrates the scope of the technical work and solutions developed in the context of HIDENETS with respect to a typical layered communication model. In particular, the results covering resilient architecture and communication, and methodologies to assist design and testing and quantitative evaluation provide the main inputs for the HIDENETS Reference Model. It is noteworthy that HIDENETS does not develop new technologies for the physical layer.

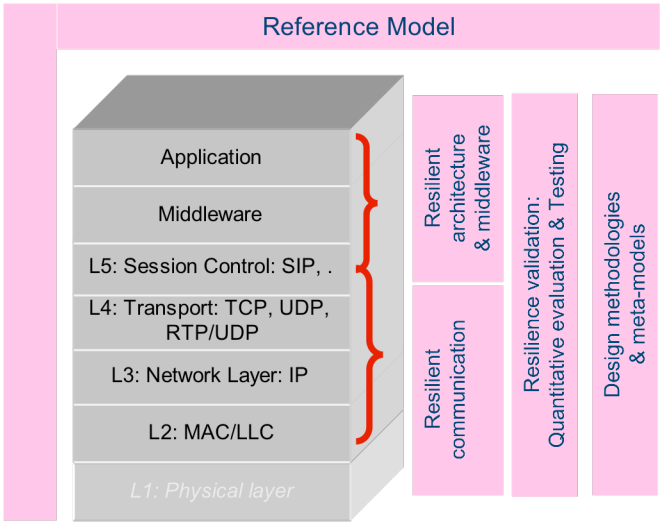


Figure 1: Illustration of scope of HIDENETS and reference model with respect to OSI model

Preliminary definitions of the scope and of the concepts behind the HIDENETS Reference Model have been presented in deliverable D1.1 [76]. This deliverable contains a refined version including a more detailed description of the main results obtained during the first 18 months of the project. Adaptations will likely still occur according to the progress and detailed results of the technical WPs. These modifications and adaptations will be contained in the final WP1 deliverable in Month 36.

The remaining part of this deliverable is structured into 9 sections. Section 2 presents dependability and resilience terminology that gives precise definitions of key concepts concerning the threats to address, the properties to satisfy and the means that can be used to achieve the target dependability and resilience

requirements. Section 3 presents an overview of the HIDENETS network and node architecture including an identification of the different layers investigated by the project. In section 4, the architectural hybridization and wormhole model underlying the HIDENETS architecture solutions is presented. The main services proposed by HIDENETS at the middleware and communication levels are described in Sections 5 and 6 together with the challenges addressed at these levels. Section 7 deals with the analysis of faults and their propagation considering a bottom-up approach from lower level communication layers up to the middleware. Section 8 focuses on the development of modelling and experimental techniques for the quantitative evaluation of dependability and resilience properties. Section 9 briefly presents the testing framework suitable for the applications investigated by HIDENETS and in particular in addressing the specific challenges raised by mobility. Section 10 deals with design methodology and meta-models needed to support the engineering and development processes. In particular, this design methodology and the meta-model are aimed to provide basic notations and modelling facilities for the description of the architecture level services described in Sections 5 and 6, and also to provide support for the quantitative evaluation and testing activities outlined in Sections 8 and 9. Finally, Section 11 concludes and presents future developments.

The HIDENETS Reference Model provides the framework in which the detailed solutions in the project are being developed, while at the same time facilitating the same task for non-vehicular scenarios and applications.

2. The Dependability and Resilience Conceptual Framework

This section introduces some basic concepts and terminology related to dependability and resilience issues that will be used in this document to characterize the HIDENETS reference model. The related concepts will be useful to define the properties, the threats, and the resilience and fault tolerance related requirements.

2.1 Basic Concepts and Terminology

The definitions presented in the following are based on the dependability concepts that have been developed and updated since the mid-seventies by the Fault-Tolerant Computing community, and especially the IFIP Working Group 10.4 [3-7]. It is noteworthy that other concepts similar to dependability exist, such as survivability, trustworthiness and resilience (e.g., see [4] for a definition of some of these concepts and a comparison with dependability). Among these, the concept of resilience extends the classical notion of fault tolerance usually applied to recover system functions in spite of operational faults, to some level of adaptability, so as to be able to cope with system evolution and unanticipated conditions[1]. Throughout this report, however, in most cases dependability, resilience and trustworthiness will be used interchangeably to refer to the ability to deliver a service that can justifiably be trusted.

The *service* delivered by a system (in its role as a service provider) is its behaviour as perceived by its user(s). The *function* of a system is what the system is intended to do and is described by the *functional specification* in terms of functionality and performance. *Correct service* is delivered when the service implements the system function. A *service failure* occurs when the delivered service deviates from correct service. A failure is thus a transition from correct service to *incorrect service*. The period of delivery of incorrect service is a *service outage*. The transition from incorrect service to correct service is a *service restoration*. Based on the definition of failure, an updated definition of dependability, which complements the initial definition in providing a criterion for deciding if the service is dependable, is as follows: the ability of a system to avoid service failures that are more frequent and more severe than is acceptable.

A systematic exposition of dependability consists of three main parts: the *threats* to, the *attributes* of and the *means* by which dependability is attained. The dependability threats correspond to faults, errors and failures that might affect the service(s) delivered by the system. The dependability attributes define the main facets of dependability that are relevant for the target system and applications. The dependability means correspond to the methods and techniques used to support the production of a dependable system. These means can be classified into four major categories:

- *fault prevention*: to prevent the occurrence or introduction of faults,
- *fault tolerance*: to avoid service failures in the presence of faults,
- *fault removal*: to reduce the number and severity of faults,
- *fault forecasting*: to estimate the present number, the future incidence, and the likely consequences of faults.

Fault prevention and fault tolerance aim to provide the ability to deliver a service that can be trusted, while fault removal and fault forecasting aim to reach confidence in this ability by justifying that the functional and the dependability and security specifications are adequate and that the system is likely to meet them.

---

[1] This interpretation is actually in line with the related on-going terminology work being carried out within the ReSIST project (www.resist-noe.org): *Resilience is the ability to deliver, maintain and improve service when facing threats (accidental or malicious) and evolutionary changes.* Such evolutionary changes could be of various types: functional, environmental or technological (hardware and software), or might occur in short term (related to dynamicity or mobility of the system components of its environments), in medium term (related to the introduction of new versions or reconfigurations) or in long term (e.g., as a result of reorganizations).

Fault prevention is part of general engineering and can be attained through the use of rigorous development techniques, high-level specification and design methodologies, structured programming, information hiding, modularization, etc.

Fault tolerance which is aimed at failure avoidance is generally implemented by error detection and subsequent system recovery. More details about these techniques are provided in Section 2.4.

Fault removal is performed both during the development phase and the operational life of a system. During the development, it consists of three steps: verification, diagnosis, and correction. Verification is the process of checking whether the system adheres to given properties, termed the verification conditions. If it does not, the other two steps are applied. Verification activities are generally implemented using a combination of static analysis, model checking, theorem proving, testing, etc.

Finally, fault forecasting is conducted by performing an evaluation of the system behaviour with respect to fault occurrence or activation. Evaluation has two aspects: a) qualitative, or ordinal evaluation which aims to identify, classify and rank the failure modes or the combinations of events that would lead to system failures, and b) quantitative, or probabilistic, evaluation, which aims to evaluate in terms of probabilities the extent to which some of the attributes of dependability are satisfied; those attributes are then viewed as measures of dependability. Various methods can be used to support these evaluations, including analytical modelling, simulation, experimental measurements as well as judgements.

The solutions investigated in the HIDENETS project cover various dimensions of dependability taking into account the four classes of dependability means (fault prevention, fault tolerance, fault removal, and fault forecasting). The development of these solutions is based on the analysis of the specific requirements and challenges characterizing various applications and use case scenarios in particular from car-to-car and automotive domains.

In the following, we present more detailed concepts related to: 1) the dependability properties, 2) the threats to be addressed to satisfy these properties, and 3) the fault tolerance mechanisms that can be used to cope with the threats.

2.2 Dependability Related Properties

Depending on the applications considered, different facets of dependability may be important, i.e., different emphasis may be put on different attributes of dependability. Basic dependability attributes are defined as follows:

- *availability*: readiness for correct service,
- *reliability*: continuity for correct service,
- *safety*: absence of catastrophic consequences on the user(s) and the environment,
- *confidentiality*: absence of unauthorized disclosure of information,
- *integrity*: absence of improper system alterations,
- *maintainability*: ability to undergo modifications and repairs,

Several other dependability attributes can be obtained as combinations or specialization of the primary attributes listed above. In particular, *security* is defined as the concurrent existence of a) availability for authorized users only, b) confidentiality and c) integrity where ‘improper’ means ‘unauthorized’.

The attributes of dependability may be emphasized to a greater or a lesser extent depending on the application: availability, integrity and maintainability are generally required, although to a varying degree depending on the application, whereas reliability, safety and confidentiality may or may not be required. The extent to which a system possesses the attributes of dependability should be considered in a relative, probabilistic sense, and not in an absolute, deterministic sense. Due to the unavoidable presence or occurrence of faults, systems are never totally available, reliable, safe or secure.

Integrity is a prerequisite for availability, reliability and safety, but may not be so for confidentiality (for instance, attacks via covert channels or passive listening can lead to a loss of confidentiality, without impairing integrity). The definition given above for integrity — absence of improper system alterations

extends the usual definition as follows: (a) when a system implements an authorization policy, 'improper' encompasses 'unauthorized'; (b) 'improper alterations' encompass actions that prevent (correct) upgrades of information; (c) 'system state' encompasses hardware modifications or damages.

Besides the attributes listed above, other secondary attributes can be considered to refine the primary attributes. An example of such a secondary attribute is *robustness*, i.e., dependability with respect to external faults, which characterizes a system's reaction to a specific class of faults.

The notion of secondary attributes is especially relevant for security, when we distinguish among various types of information. Examples of such secondary attributes are:

- *accountability*: availability and integrity of the identity of the person who performed an operation
- *authenticity*: integrity of a message content and origin, and possibly of some other information, such as the time of emission.
- *non-repudiability*: availability and integrity of the identity of the sender of a message (non-repudiation of the origin), or the receiver (non-repudiation of reception)

Variations in the emphasis on the different attributes of dependability directly affect the appropriate balance of the techniques (fault prevention, tolerance, removal, forecasting) to be employed in order to make the resulting systems dependable. This problem is all the more difficult as some attributes conflict (e.g., availability and safety, availability and security), necessitating design trade-offs.

2.3 Threats

The dependability threats mainly correspond to the faults, errors, and failures that should be covered by the target applications to satisfy the desired dependability properties.

A service may fail either because it does not comply with the functional specification, or because this specification did not adequately describe the system function. A service failure occurs when at least one or more external state(s) of the system deviate from the correct service state. The deviation is called an *error*. The adjudged or hypothesized cause of an error is called a *fault*.

A system may not, and generally does not, always fail in the same way. The ways a system can fail are its *failure modes*, which may be characterised according to four viewpoints: 1) the failure domain, 2) the detectability of failures, 3) the consistency of failures, and 4) the consequences of failures on the environment.

The *failure domain* viewpoint leads to the distinction of *content failures* (e.g., incorrect values) and *timing failures*. *Value failures* are a particular case of content failures. Timing failures may be of two types: *early* or *late* depending on whether the service was delivered too early or too late. Failures when both content and timing are incorrect fall into two classes:

- *halt failure*, or simply halt, when the service is halted (the external state becomes constant, i.e., system activity, if there is any, is no longer perceptible to the users); a special case of halt is *silent failure*, or simply silence, when no service at all is delivered at the service interface (e.g., no messages are sent in a distributed system).
- *erratic* failures otherwise, i.e., when a service is delivered (not halted), but is erratic (e.g., babbling).

The *detectability of failures* viewpoint addresses the signalling of the service failures to the users. Signalling at the service interface originates from detection mechanisms in the system that check the correctness of the delivered service. When the losses are detected and signalled by a warning signal, then *signalled failures* occur. Otherwise, they are *unsignalled failures*. The detection mechanisms themselves have two failure modes: 1) signalling a loss of function when no failure has actually occurred, that is a *false alarm*, 2) not signalling a function loss, that is an *unsignalled failure*. When the occurrence of service failures result in reduced modes of service, the system signals a degraded mode of service to the user(s). Degraded modes may range from minor reductions to emergency service and safe shutdown.

The *consistency of failures* viewpoint when two or more service users are involved leads to the distinction of *consistent failures* (when the incorrect service is perceived identically by all the users) from *inconsistent failures*, also called *Byzantine failures*, (when some or all users perceive an incorrect service differently),

The *consequences of failures* on the environment viewpoint leads to the grading of failure modes according to different *failure severities*. The failure modes are ordered into severity levels, to which are generally associated maximum acceptable probabilities of occurrence. The number, the labelling, and the definition of the severity levels, as well as the acceptable probabilities of occurrence, are application-related, and involve the dependability and security attributes for the considered application(s).

When designing a dependable system, it is very important to identify which fault classes are to be taken into account because different means are to be used to deal with different fault classes. Thus, fault assumptions influence directly the design choices, and also the level of dependability that can be achieved.

Faults and their sources are very diverse. They can be classified according to different criteria: the phase of creation (development *vs.* operational faults), the system boundaries (internal *vs.* external faults), their phenomenological cause (natural *vs.* human-made faults), the dimension (hardware *vs.* software faults), the persistence (permanent *vs.* transient faults), the objective of the developer or the humans interacting with the system (malicious *vs.* non malicious faults), their intent (deliberate *vs.* non-deliberate faults), or their capability (accidental *vs.* incompetence faults).

Malicious faults are human-made faults that are generally introduced with the malicious objective to alter the functioning of the system during use. The goals of such faults are: 1) to disrupt or halt service, causing denials of service; 2) to access confidential information; or 3) to improperly modify the system. They can be grouped into two classes: 1) malicious logic faults that encompass faults introduced during the development phase such as Trojan horses, logic or timing bombs, and trapdoors, as well as operational faults such as viruses, worms or zombies (see e.g., [4] for a precise definition of these terms); and 2) intrusion attempts that are operational external faults. The external character of intrusion attempts does not exclude the possibility that they may be performed by system operators or administrators who are surpassing their rights.

The list of failures and faults assumptions to be addressed in the development process should be completed by the specification of the acceptable degraded operation modes as well as of the constraints imposed on each mode, i.e., the maximal tolerable service interruption duration and the number of consecutive and simultaneous failures to be tolerated, before moving to the next degraded operation mode. The analysis of the impact of the simultaneous loss or degradation of multiple functions and services requires particular attention. Depending on the dependability needs and the system failure consequences on the environment, the need to handle more than one nearly concurrent failure modes could be vital. Such an analysis is particularly useful for the specification of the minimal level of fault tolerance that must be provided by the system to satisfy the dependability objectives. It also provides preliminary information for the minimal separation between critical functions that is needed to limit their interactions and prevent common mode failures.

2.4 Fault Tolerance

Fault tolerance is aimed at failure avoidance. It is generally implemented by *error detection* and subsequent *system recovery* (or simply recovery).

There exist two classes of error detection techniques:

- *concurrent error detection* which takes place during service delivery
- *preemptive error detection* which takes place while service delivery is suspended; it checks the system for latent errors (i.e., that are not yet detected) and dormant faults (i.e., that are not yet activated).

Recovery transforms a system state that contains one or more errors (and possibly faults) into a state without detected errors and faults that can be activated again. Recovery consists of error handling and fault handling.

Error handling eliminates errors from the system state. It may take three forms:

- *rollback,* where the state transformation consists of returning the system back to a saved state that existed prior to error detection; that saved state is a *checkpoint*,

- *compensation,* where the erroneous state contains enough redundancy to enable error elimination,
- *rollforward,* where the state without detected errors is a new state.

Fault handling prevents faults from being activated again. It involves four steps:

- *fault diagnosis,* which identifies and records the cause(s) of error(s) in terms of both location and type,
- *fault isolation,* which performs physical or logical exclusion of the faulty components from further participation in service delivery,
- *system reconfiguration,* which either switches in spare components or reassigns tasks among non-failed components,
- *system reinitialization,* which checks, updates and records the new configuration and updates system tables and records,

Usually, fault handling is followed by corrective maintenance that removes faults isolated by fault handling.

Systematic usage of compensation may allow recovery without error detection. This form of recovery is called fault masking. However, such simple masking will conceal a possibly progressive and eventually fatal loss of protective redundancy; thus practical implementations of masking generally involve error detection (and possibly fault handling), leading to *masking and recovery*.

The choice of error detection, error handling and fault handling techniques, and of their implementation is directly related to and strongly dependent upon the fault assumptions. The classes of faults that can actually be tolerated depend on the fault assumptions considered in the development process. Various techniques for achieving fault tolerance can be used such as performing multiple computations in multiple channels, either sequentially or concurrently, where the channels may be of identical design (if the objective is to tolerate independent physical faults or elusive design faults) or may implement the same function via separate designs and implementations, i.e., through *design diversity* (if the objective is to tolerate solid design faults). Other techniques include the use of self-checking components which provide the ability to define error confinement areas.

Fault tolerance is a recursive concept: it is essential that the mechanisms that implement fault tolerance should be protected against the faults that might affect them. Examples of such protection are voter replication, self-checking checkers, stable memory for recovery programs and data.

The notion of *coverage*, in particular attached to the efficiency of the fault tolerance techniques and mechanisms especially with respect to the failure assumptions they rely upon, is essential to ensure the overall ability to actually achieve the targeted dependability and security levels.

Systematic introduction of fault tolerance is often facilitated by the addition of support systems specialized for fault tolerance (e.g., software monitors, service processors, dedicated communication links).

Fault tolerance is not restricted to accidental faults. Some mechanisms of error detection are directed towards both malicious and non-malicious faults (e.g., memory access protection techniques) and schemes have been proposed for the tolerance of both intrusions and physical faults, via information fragmentation and dispersal, as well as for tolerance of malicious logic, and more specifically of viruses, either via control flow checking, or via design diversity. It is noteworthy that the extension and adaptation to security of traditional techniques for tolerating accidental faults, led to the emergence of the *intrusion tolerance* concept. The focus of intrusion tolerance is on ensuring that systems will remain operational (possibly in a degraded mode) and will continue to provide core services despite faults due to intrusions.

3. HIDENETS Architecture Overview

This section presents first an overview of the HIDENETS network architecture and application context to clarify the various types of scenarios and interactions investigated by the project. Then, we introduce the basic models and assumptions underlying the design of the HIDENETS architecture. Finally, we present a simplified and high-level description of the architecture itself, considering the architecture of the nodes that will be implementing the basic dependability services and mechanisms needed to provide the level of resilience required for the HIDENETS applications.

3.1 HIDENETS Network Architecture and Application Context Description

The HIDENETS network architecture introduces the relevant network elements and domains as illustrated in Figure 2. We distinguish two fundamentally different domains: 1) the *ad hoc domain* in which service access and service deployment are performed in a wireless setting, and 2) the *infrastructure domain* that consists of a back-bone IP network connecting both service providers as well as service clients. Parts of the *ad hoc* domain may be connected to the infrastructure domain via cellular access (GPRS/UMTS) or via WLAN hot-spots.

As illustrated in Figure 2, mobile nodes communicate with other mobile nodes directly, or via the infrastructure domain. In the HIDENETS scenarios, these nodes will typically be cars (or terminals in cars, either integrated or portable), but they may also be car-external devices. Mobile nodes may also communicate with nodes in the infrastructure domain. In fact, three main classes of scenarios are studied:

1) All communicating entities are located in the *ad hoc* domain. Note that this includes scenarios in which the infrastructure domain is needed for connectivity, when the entities may not be within *ad hoc* connectivity of each other.
2) The service accessing entities are located in the *ad hoc* domain and the service provisioning entities are in the infrastructure domain.
3) The service accessing entities are in the infrastructure domain and the service provisioning entities are in the *ad hoc* domain.

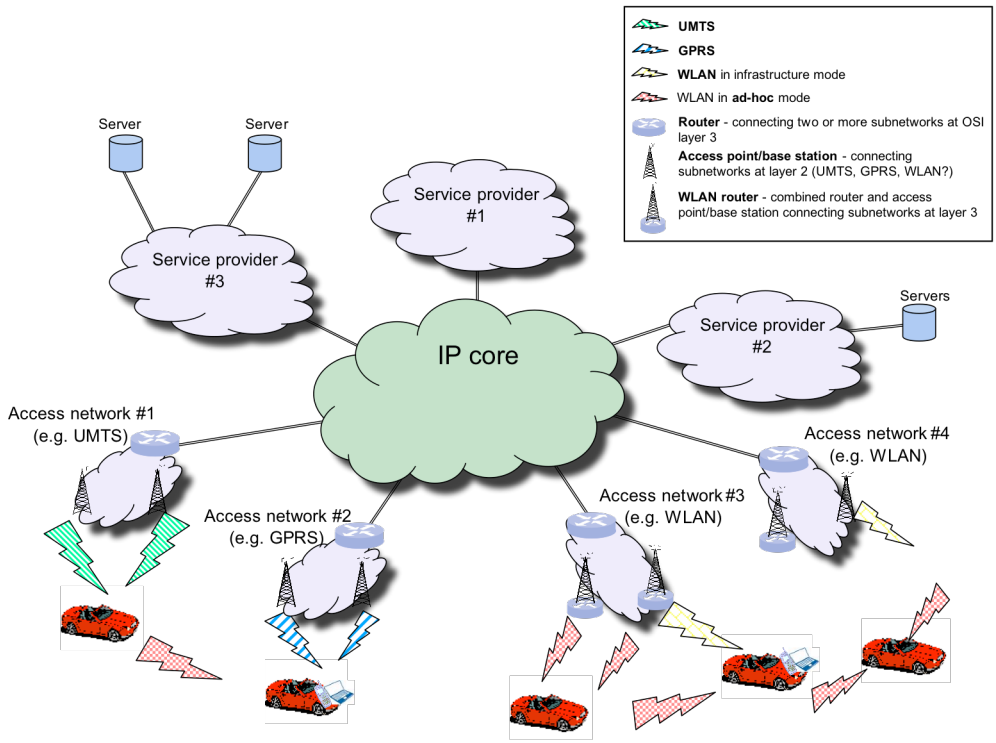


Figure 2: HIDENETS network architecture – infrastructure and *ad hoc* domains

A *mobile node* is a node communicating via wireless technologies and protocols so that it can potentially move without losing connection. In the figure we have a set of mobile nodes (the cars) that are communicating directly with other mobile nodes, or via a fixed network, via different types of wireless links (Access Link or *Ad hoc* Link).

When mobile nodes communicate or are ready to communicate directly without an infrastructure, i.e., within the *ad hoc* domain, we call it an *ad hoc network*. The nodes may then run applications that are peer-to-peer in nature, or where the server-part of the application is implemented in a wireless node. They may also communicate via an access network connecting the mobile nodes to the infrastructure domain. We assume that several service providers are connected to the IP core part of the infrastructure domain. These provide, besides applications/services running in the *ad hoc* network, additional applications/services for the *ad hoc* nodes.

When an *ad hoc* network is connected to the infrastructure domain, acting as an extension to the infrastructure domain, there will be one or more devices functioning as *gateways* between the two domains. There are several technologies that could be used for such a gateway. An important example is a *WLAN access point* that connects hosts with WLAN interfaces operating in Infrastructure mode together, forming a wireless network (WLAN). In the case that the WLAN access points are connected to the infrastructure domain (normally the case, and making it a gateway), they also forward data between the wireless hosts and servers or hosts connected to the wired network. A WLAN access point operates at OSI layer 2, but it can also be integrated with a router, in which case it is called a WLAN router.

Another important gateway technology is a GSM/GPRS/UMTS base station. This is more specifically a network element in the radio access network responsible for radio transmission and reception to and from the user equipment. It is as such always connected to the infrastructure domain, and communication between the wireless hosts is transmitted via the mobile core network. The operation of GSM/GPRS/UMTS base stations is defined by 3GPP standards (e.g., see [29]). The coverage area of a GSM/GPRS/UMTS base station is termed a **cell**. A device moving from one cell to another will automatically be handed over from one base station to another.

Note that, according to our definitions, a car can be an *ad hoc* node even if it is not connected to other cars in an *ad hoc* manner. Second, *an ad hoc node can also function as an ad hoc gateway at the same time*, i.e., the *ad hoc* node acts as an interface to the infrastructure domain (Fixed-Wireless *Ad hoc* Gateway or Wireless-Wireless *Ad hoc* Gateway). In summary, an *ad hoc* node can be in several states: *ad hoc* connected only; *ad hoc* disconnected, but infrastructure connected; gateway (both *ad hoc* and infrastructure connected); both *ad hoc* and infrastructure disconnected.

Wireless Technologies: Except for the Layer-2 mechanisms, most of the dependability solutions that are introduced as part of the HIDENETS reference model in subsequent sections are in fact independent of the underlying link-layer technology that is used for *ad hoc* connectivity and for the connection to the infrastructure domain. The link-layer technology however strongly influences the communication properties (as expressed by neighbour discovery and link establishment delays, link throughput, L2-frame delays, L2-frame loss probability, availability of L2 broadcast functions) and hence will influence the quantitative performance and dependability metrics on and above the link-layer. Therefore, it is important to identify relevant candidate technologies, so that they can be used in the quantitative analysis and testing. For dependability functionality placed on the link-layer (such as multi-channel MAC), candidate technology selection is even mandatory, as it directly influences the conceptual design of such functionalities.

The main candidates for the *ad hoc* link-layer connectivity are the Wireless Local Area Networks (WLAN) described by the IEEE 802.11 standards [8]. Several varieties are likely candidates in HIDENETS: common 802.11a/b/g networks where unlicensed frequency bands are available, or 802.11p [27] networks for vehicular communication (draft standard). Due to the presence of an additional control channel in 802.11p and the use of licensed spectrum, 802.11p can show advantages in particular for safety-critical applications. The original WLAN standards provide a best effort service, and when the offered traffic load is too high, the overall network performance drops. The extended 802.11e [9] provides functions for differentiated (not guaranteed) QoS.

For non-vehicular *ad hoc* communication beyond the scope of HIDENETS, also short-range technologies such as Bluetooth and the IEEE 802.15 family or upcoming Ultra-Wide-Band communication can be

interesting candidates. This is in particular the case, if small distances, low mobility, and scarceness of battery-energy are present; relevant example scenarios include Personal (Area) Networks and sensor network scenarios. In HIDENETS, these technologies are not considered.

For the connection to the infrastructure, in addition to the packet-switched transport services of cellular networks GPRS and UMTS, WIMAX (mobile versions of the IEEE 802.16 family) and WLAN 802.11-like link-layer protocols in so-called infrastructure mode are the most interesting candidates. While the long-range cellular technologies operate through the already installed radio access networks, dedicated road-side access points will need to be deployed in most cases for the WLAN-based infrastructure access.

For more general, non-vehicular, communication scenarios, short-range technologies can also be used for the infrastructure connectivity, e.g., via the use of Bluetooth access points, sometimes even deployed using meshed networks technology. Although such scenarios are out of scope for HIDENETS, the solutions as presented in this reference model could be relevant and should be adapted and tuned to take into account the specific requirements inherent to these scenarios.

3.2 HIDENETS Applications

This reference model shall help to develop systems which can be used to run applications with a level of dependability, as required by the application. Various applications and use cases with different characteristics have been considered in the context of HIDENETS to identify the main dependability and resilience requirements that need to be addressed and the challenges for which middleware and communication level services have to be advised. Such characteristics can be translated into a number of measures, some of which are described by the middleware and communication level properties presented in Sections 5 and 6.

The applications considered include pure information and entertainment applications (e.g., web browsing, voice and audio streaming, video, audio conferencing and on-line gaming), more car-related services (e.g., Platooning, traffic sign extension and floating car data), as well as safety-relevant applications (e.g., hazard warnings, distributed black box and communication with medical experts). In fact, some of these applications exhibit similar characteristics and could therefore be grouped accordingly. The advantage of considering groups, or classes of applications, is that it may be possible to devise the solutions for improved dependability in a more generic way, for classes of applications instead of individual ones. Additional details on some possible classes of applications are provided in Section 4.3.2

It is obvious that there will always be a mixture of applications in a given scenario. This is reflected by the HIDENETS use cases which represent collections of a number of applications that are put into a wider context and which interact with each other. Interaction here can mean that the information exchanged by different applications depends on each other, but equally it may mean that they have different priorities and may supersede each other (for details, see D1.1 [76]).

For the sake of illustration, the following use cases described in [76] present three typical examples of applications that are relevant in the context of HIDENETS:

Platooning: A platoon is formed by two or more vehicles following each other closely, controlled by the vehicle at the head of the platoon. This application provides both positional and velocity control of vehicles in order to operate safely as a platoon on a highway. Besides improving safety, the objective is to optimize highway traffic flow and capacity. Platooning requires vehicle-to-vehicle communication and may include vehicle-to/from-infrastructure communication. The dependability needs include the satisfaction of timeliness and fail-safe requirements, and also some requirements in terms of security and authenticity of data.

Car accident: This use-case covers situations that occur before, during or after an accident happens. Before an accident, time-stamped information characterizing the state of a car and its environment can be collected, backed up to other cars as they pass, as well as to fixed-network servers. Such information can be used as a virtual black-box for investigating the conditions that led to the occurrence of the accident. Thus, efficient means for ensuring data availability, integrity and confidentiality are needed in this context. Right after the accident, dependable and timely communications with the emergency services and medical teams, including text, voice and multimedia messages, could also be necessary.

Assisted transportation: This use case covers the situations in which driving by car is subject to general constraints, like time constraints, route constraints (the need to pass by specific locations), or both. Several applications might be used, independently or in combination, in order to better assist the user in achieving its goals. These include for example: a) applications that collect and disseminate to other vehicles information concerning floating car data and hazard warnings that is useful to plan an adequate route and to prevent accidents, or b) traffic sign extension application which consists in using intelligent signs to allow centralized control of the information indicated by each sign and proactive dissemination of information between signs and to vehicles passing by. As for the dependability needs, it is fundamental to ensure that the information processed within a car and disseminated through the network and wireless links are consistent with real conditions of the environment. Thus timing failures have to be addressed carefully and reliable communication solutions have to be provided. Other requirements related to security such as ensuring the authenticity and trustworthiness of the disseminated data are also important.

The application and use cases properties as well as the corresponding challenges are the main driver for the development of this reference model. So the reference model is a collection of possible means and methods that can be used to develop a system tailored to support a set of applications in a given scenario with certain dependability requirements.

3.3 HIDENETS Node Architecture – Simplified Description

In this section, we present a simplified description of the proposed architecture of a HIDENETS mobile node that will include the software and hardware components and services needed to run a HIDENETS application in an ad-hoc based mobile environment and to satisfy its dependability and resilience requirements. A more detailed description of the services and building blocks of the proposed architecture is presented in Sections 4, 5 and 6.

A first version of the simplified node architecture is shown in Figure 3, in which three distinct layers are shown: the *hardware layer*, the *operating system layer* and the *user space layer*.

The node consists of some hardware (HW) that may be installed in a mobile node (e.g., a car) or be part of a separate terminal, and some software running on it. One particular piece of hardware is the network interface card that allows the transmission of information out on the network. Other relevant hardware parts may for instance be GPS devices.

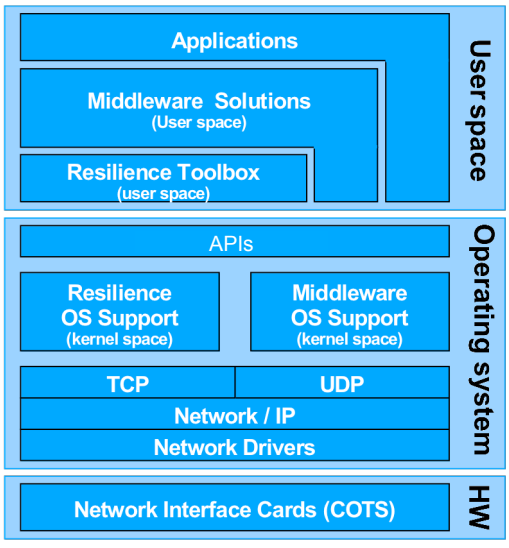


Figure 3: Simplified node architecture

The node software may be part of the operating system or it may be implemented in user space. Regular applications that may be installed and run by users are always implemented in user space. Since user space applications are thought of as potentially *untrusted*, they are only allowed to access the operating system

functions through well-defined Application Programming Interfaces (APIs). On the other hand, software that is included into the operating system is thought of as *trustworthy* and is allowed to use operating system functions, read variables and even interact with low level hardware.

Resilience functions in user space may be implemented as separate functions, included within middleware, or built into the applications themselves. In the operating system, provided functions may be categorized in three main blocks: Middleware OS support, Resilience OS support, and Communication/Networking support. Resilience OS support functions are provided as part of the general Middleware OS Support. They are, however, categorized as a separate block to highlight the possible existence of specific functions within the operating system to support resilience. The third block concerns Communication/Networking support, including general OS provided communication related functions, typically implementing OSI layers 2 to 4.

In fact, the figure can be drawn differently, to make more explicit that Resilience OS support does not necessarily need to be developed on top of the "standard" network layers implemented in the OS. While resilience support functions in the OS may not need communication support, they may have to rely on other system resources for which low-level access must be granted. As depicted in Figure 4, from an OS perspective these resilience support functions are now drawn as a service block that is located side-by-side with communication/networking functions. Therefore, resilience support will have independent access to low-level devices and hardware that may be used to improve some resilience aspects. For instance, the interaction with a GPS device connected to the node, which provides accurate timing information, is relevant for dependability purposes. Interactions with other components, like hardware device controllers, could also be envisaged. All these interactions are independent of the existent OS communication support. This architecture would also allow considering solutions in which the communication stack would be able to access resilience support functions.

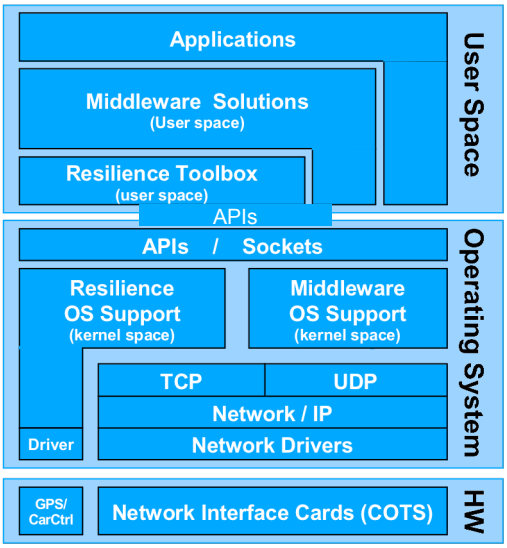


Figure 4: Simplified node architecture – OS perspective

This view of Resilience OS Support functions as a special domain within the OS has some limitations. In particular, it does not express the possibility of endowing resilience support functions with stronger properties than those exhibited by the "normal" system and OS. A perspective in which this is expressed is provided in Figure 5, which introduces the simplified view of a hybrid system architecture.

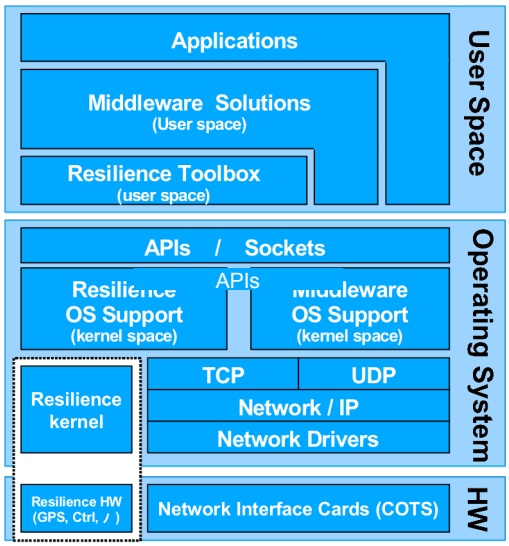


Figure 5: Simplified node architecture – hybrid system perspective

The simplified node architecture in Figure 5 includes a special part, clearly separated from the remaining system, referred to as a *Resilience Kernel*. This resilience kernel is a subsystem that has better properties than the rest of the system (user space and OS). Typically, this means that it can be timelier, more secure and/or more reliable than the rest of the system. These better properties represent a potential for the improvement of the overall node resilience.

Looking at the node as a whole, the existence of these two parts with different sets of properties prefigures a system that is well characterized by the *Wormholes model* [70], in contrast with other distributed systems models that assume homogeneous properties for the entire system. Therefore, in the following section (Section 4) we discuss the wormhole model and the implications of adopting such an hybrid system architecture in HIDENETS, in particular concerning resilience improvements. The concrete services and solutions that we envision for the HIDENETS architecture are presented in Sections 5 and 6, where we focus on "middleware services" (Section 5), which include functionalities or services not specifically related to communication aspects, and "communication services and protocols" (Section 6), including communication-related services. The model presented in Figure 5 will be considered as a basis for these discussions.

3.4 Middleware Interfaces and Standardization

HIDENETS applications will run on different HIDENETS nodes that – because of the possibly different HW platforms — may have different implementations of the HIDENETS services. In order to support the execution of applications in this environment without modifications, specific interfaces are created for each service. The applications access the services through these interfaces.

Furthermore, it is important for the application programmers to create code that is reusable for different solutions, thus, reduce the development time. Reusability is highly facilitated by the application of standards. The specifications of the Service Availability Forum (SA Forum)[2] define a complete set of standard interfaces for accessing the services of highly available middleware implementations. In addition, the development of HIDENETS applications can be based on the best practices and experiences of SA Forum application development projects.

By taking the above mentioned advantages, in HIDENETS, we decided to provide services on the basis of SA Forum's interfaces. However, there are additional HIDENETS service interfaces that are made directly accessible for applications since the functionality they provide is out of the scope of the SA Forum specifications.

[2] http://www.saforum.org.

The structure of the HIDENETS middleware is depicted in Figure 6. It is shown that the applications access the services through the standardized SA Forum interfaces and the predefined HIDENETS interfaces and do not directly use the HIDENETS service implementations.

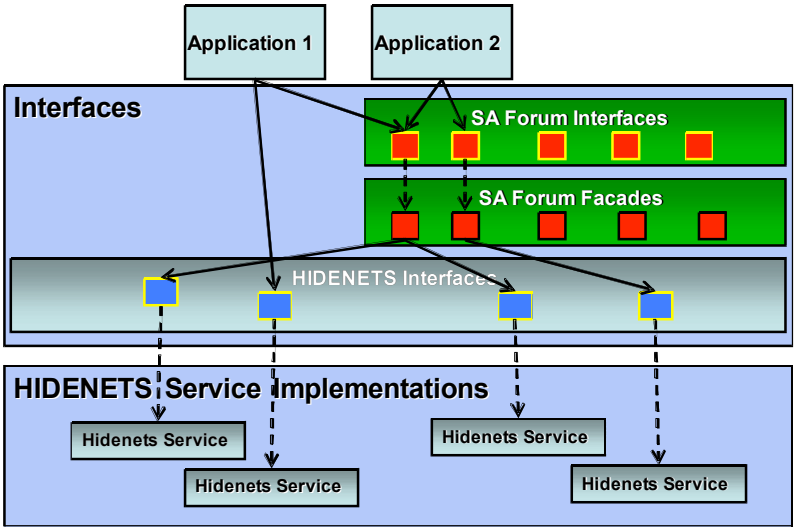


Figure 6: Structure of the HIDENETS middleware

Relation between SA Forum services and HIDENETS services. Although, applications use the SA Forum interfaces, those services do not have separate implementations. Rather they are implemented as façade objects that use the existing HIDENETS services to carry out a specific operation. Generally, a *façade object* is an object that provides a simplified interface to a larger body of code, such as a class library. In our case, we use the façade objects to hide the HIDENETS service calls and to provide higher-level operations to applications. We also plan to propose HIDENETS developed or extended services to the SA Forum for possible impact on related standardization activities.

4. Architectural Hybridization

This section introduces first the basic models and assumptions underlying the design of the HIDENETS architecture. Then, it outlines briefly the main concepts behind the architectural hybridization and the wormhole models and their application to the HIDENETS architecture design.

4.1 Modelling the Synchrony of the System

The definition of a system model implies making assumptions about a certain number of aspects. In the context of HIDENETS, we are interested in modelling distributed systems. In this kind of systems, where computations involve several processes or nodes that need to exchange information among them, a major concern has to do with modelling the communication between processes. Modelling local computations is also important, namely when dealing with timeliness issues, in which case synchrony aspects become fundamental.

With respect to communication, in particular, a system model can be characterized by the assumptions it makes about the following four concerns: *failure,* s*ynchrony*, *network topology* and *message buffering*. It is also possible to characterize other attributes of a system, like, for instance, the *interaction* among processes or the system *complexity*.

In this section we address architectural hybridization as a proposed approach for the design of systems with improved resilience. Despite the several dimensions in which it may be possible to characterize a system model, architectural hybridization has particular implications concerning *synchrony* assumptions. Therefore, before delving into the discussion of the architectural hybridization approach proposed for HIDENETS, we review existing models of synchrony and make the case for the use of a partially synchronous model.

Synchrony models allow characterizing a system with respect to the existence of notions of *time* and *timeliness*. The weakest model is the *asynchronous* model, also called *time-free*. A time-free model makes absolutely no assumptions related to time, which means that message delivery and process response time can be arbitrarily large. Furthermore, in time-free models there are neither clocks nor any other form of measuring time intervals. On the other extreme, the strongest model is the *synchronous* model, in which communication and processing delays are always bounded, and the bounds are known. Intermediate models are known as *partially synchronous* models. They do only the necessary time and/or timeliness assumptions that will allow satisfying some required properties with the appropriate dependability. For example, the Timed Asynchronous model, [74], only assumes the availability of local clocks with a bounded rate of drift, and the Timely Computing Base model, [75], assumes that some parts of the system are synchronous while other parts may be asynchronous. The former is powerful enough to eventually allow the detection of timing failures, while the latter also provides the means for a timely detection and a controlled and timely reaction to those failures. The use of partially synchronous models is attractive for the work in HIDENETS, since these models allow capturing the dynamics and heterogeneity of the operational environments characterizing the target applications and use cases.

4.2 Architectural Hybridization and the Wormholes Model

Computing systems and computer networks are inherently heterogeneous. This is true both in terms of the functional and non-functional properties they exhibit. In fact, we reason about systems in terms of their functional attributes, such as speed, power consumption or memory capacity, and also in terms of non-functional ones, such as their reliability, security, dependability or resilience, which as we know, are typically not homogeneous in a distributed system, i.e., the required properties and the level of resilience and dependability might not be the same for all system components and subsystems. On the other hand, it is interesting, and perhaps peculiar, to note the contrast with distributed system models, which are generally homogeneous, that is, which assume the same properties for the whole system. This is particularly noticeable in what concerns synchrony properties, where two typical system models can be distinguished: a) purely

asynchronous models that do not make any time-related assumption or b) purely synchronous models assuming that timeliness bounds will always be satisfied throughout the system. Some partially synchronous system models also exist, which assume some local synchrony (e.g., reliable clocks) or that synchrony is eventually achieved and will remain for sufficiently large amounts of time to finalize computations.

Differently from previous approaches, in HIDENETS we follow a hybrid distributed system model approach, in which different parts of the system have different sets of properties and can rely on different sets of assumptions, namely in terms of faults and synchronism. This has a number of advantages when compared to approaches based on homogeneous models, as we briefly explain in what follows (a detailed discussion can be found in [70]).

Hybrid distributed systems models are:

- *Expressive models with respect to reality*— Real systems are partially synchronous in the time dimension. But further to that, they generally have components with different degrees of synchronism, i.e., in the space dimension: different bounds or absence thereof, different speeds, different tightness and steadiness (metrics of synchronism, *see* [25]). Homogeneous models simply cannot take advantage from this, being confined to use the worst-case values or bounds (e.g., of the least synchronous component), which ultimately— and safely— mean asynchrony.
- *Sound theoretical basis for crystal-clear proofs of correctness*— Why were some problems found in earlier-generation asynchronous algorithms [71][72]? One of the reasons was because timing assumptions were made for the system, that were not in agreement with the model. Artificially restricting such assumptions to 'parts' of an asynchronous system does not improve the situation much if it follows a homogeneous model: we may fall into the same kind of contradictions. However, by using a hybrid model, the heterogeneous properties of the different *loci* of the system (the space dimension) are by nature represented, and we are in consequence forced to explicitly make correctness assertions about each of these loci, and about the interfaces to one another.
- *Naturally supported by hybrid architectures*— Sisters to hybrid distributed systems models, hybrid architectures accommodate the existence of actual components or subsystems possessing different properties than the rest of the system. Hybrid models and architectures provide feasibility conditions for powerful abstractions which are to a large extent unimplementable on canonical (homogeneous) asynchronous models: failure detectors; *ad hoc* synchronous channels; timely execution triggers (also known as watchdogs) or timely executed actions (periodic or event-triggered)[3]. Hybrid models and architectures may drastically increase the usefulness and applicability of all these abstractions.
- *Enablers of concepts for building totally new algorithms*— A powerful yet simple concept behind the first experiments with hybrid models was: use the weakest possible model for the generic system; imagine that a "toolbox" of simple but stronger low-level services is available, locally accessible to processes (e.g., timely execution triggers; timely executed actions; trusted store); these local services can be distributed via alternative channels, to obtain further strength (e.g., synchronous channels; trusted global time; trusted binary agreement); devise algorithms which, by working between the two space-time realms, the generic and the enhanced subsystem containing the "toolbox", achieve new properties (e.g., making an asynchronous process enjoy timely execution).

One example of a hybrid distributed systems model is the *Wormholes mode,* named in this way after the astrophysics theory. This theory argues that one could take shortcuts, through, say, another dimension, and re-emerge safely at the desired point, apparently much faster than what is allowed by the speed of light. Those shortcuts received the inspiring name of *Wormholes*. In essence, Wormholes prefigure an ancillary theory which coexists with the classical theory, predicting subsystems which present exceptional properties allowing to overcome fundamental limitations of the systems under the classical theory.

Having described the fundamental advantages of hybrid models, of which the Wormholes model is an example, we do not provide a description of the Wormholes model here, for which the reader can refer to [70].

[3] The same could be said of security-related abstractions concerning tamper-proofness *vs.* Byzantine models.

From a more practical point of view, a Wormhole can be instantiated by following the architectural hybridization paradigm. This paradigm not only explores the inherent heterogeneity of practical systems, when possible, but also defines a few principles that pave the way for the construction of the required hybrid and modular systems.

In fact the architectural hybridization paradigm is based on the following principles:

Heterogeneity principle: Systems can be composed of several realms with different properties, in particular non-functional ones, such as quality-of-service, synchronism, security, etc.

Construction principle: In order to follow the architectural hybridization paradigm for building distributed systems it is possible to do more than just simply exploit the natural hybrid nature of the underlying infrastructures. The designer can be proactive in ensuring that a set of desired properties is provided by each of the subsystems that compose the overall system. This means that specific engineering measures can be taken to create realms in the system with (typically better) properties (e.g., multiple communication channels, multiple priority levels for the execution of application processes, mechanisms for differentiating the access to some specific system functions, etc), or it may be even possible to add new components as necessary to create these realms (e.g., additional network interfaces, watchdog cards, additional embedded processors or controllers, etc.).

Encapsulation and interfacing principle: Given the existence of realms with different properties, it is fundamental to ensure the encapsulation of these realms and the provision of well-defined interfaces, as the only mean through which each of the realms can manifest their properties.

We may say that architectural hybridization is an enabler of the construction of realistic hybrid distributed systems. In fact, the HIDENETS node architecture presented in Figure 5 provides an illustration of the architectural hybridization paradigm: the Resilience kernel corresponds to the subsystem with the better properties, which should be constructed according to the above-mentioned construction and encapsulation principles.

4.3 Middleware Oracles in the HIDENETS Architecture

The HIDENETS architecture is a hybrid architecture, thus implementing the Wormholes distributed systems model [70]. It is divided into two (logical) subsystems:

- *Wormhole subsystem*: in HIDENETS this corresponds to the Resilience kernel, on which some simple but critical services will reside. We call to these services *timeliness and trustworthiness middleware oracles (MW oracles)*.
- *Payload subsystem*: in HIDENETS this corresponds to the rest of the system. The services residing in the payload subsystem will be *complex middleware services, OS services and network services*.

MW oracles are trusted components ("more" trusted with respect to the Payload subsystem). They offer services useful for applications with strict real-time and/or with strict security requirements. The behaviour of these components must be very strict and predictable: to not impact reliability and safety of the whole system their services have to be provided with certain guarantees at the interfaces. The identification of the role of all MW oracle services is important: what kind of guarantees exactly they provide, and what are the requirements from the hardware/operating system/network in order to maintain such guarantees with a given probability.

The communication between payload and wormhole parts of the architecture is based on a wormhole gateway (see Figure 7). The only way for payload processes to communicate with the MW oracles is through these wormhole gateways, with well-defined interfaces (thus following the encapsulation and interfacing principle). Likewise, for payload processes the properties offered by any wormhole are defined and enjoyed at a wormhole gateway. The payload processes do not have to know how wormholes are implemented, and vice-versa.

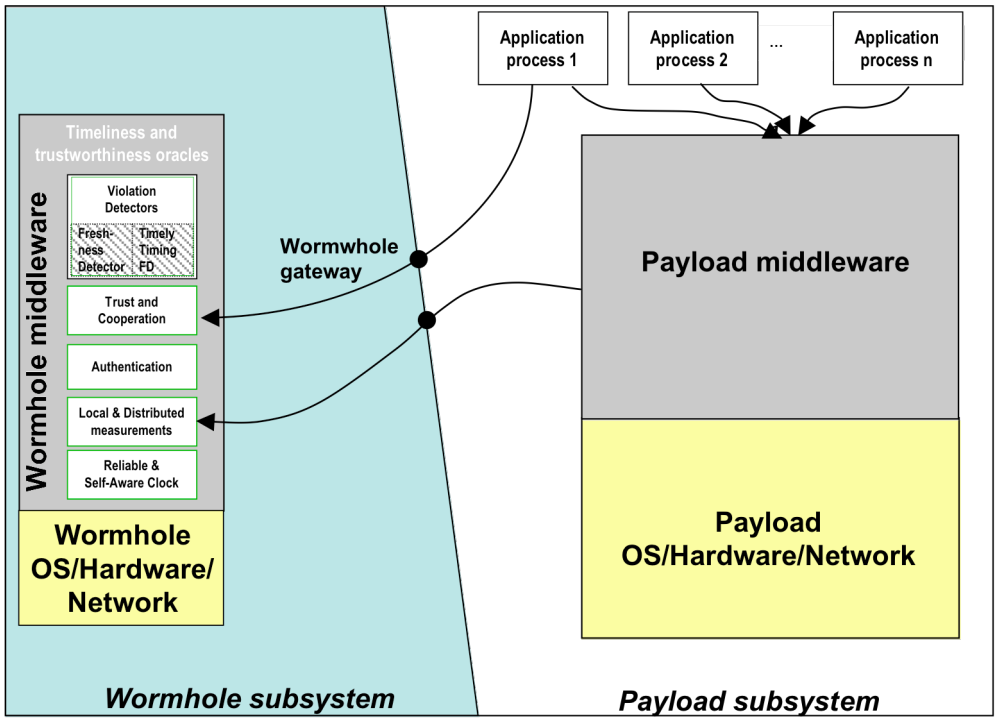


Figure 7: Wormhole and Payload subsystems in HIDENETS and their interconnections

The wormhole part of the architecture provides services with different quality with respect to the payload part. In particular, it assists the execution of fault-tolerant algorithms, offering a minimal set of trusted and timely services, implemented by the MW oracles. In order to be able to offer such services in the wormhole subsystem (with a certain probability), we have more stringent requirements from the execution environment of the wormhole subsystem.

4.3.1 Classification of the Middleware Oracles

The functionalities offered by MW oracles may be classified along several axes. Three interesting dimensions can be identified as follows (Figure 8):

1) *Scope*: the nature of the offered service; e.g., is the MW oracle implemented locally or is it a distributed MW oracle requiring the execution of a distributed protocol and thus requiring communication resources?
2) *Resilience requirements*: the level of resilience required by the service provided by the MW oracle.
3) *Timeliness requirements*: the level of timeliness required by the service provided by the MW oracle.

This classification can be seen as a tentative classification of *non-functional requirements* of the MW oracles. From Figure 8, we can see that the difficulty of satisfying the requirements of each service is directly proportional with the distance of the service from the origin of the axes. For example, a MW oracle with a distributed scope will be more difficult to construct than a MW oracle with only local scope, since it will require the satisfaction of additional properties for the communication among oracle instances in different nodes. Also, it is clear that higher resilience and timeliness requirements will imply an extra effort in the construction of these oracles in order to satisfy those requirements.

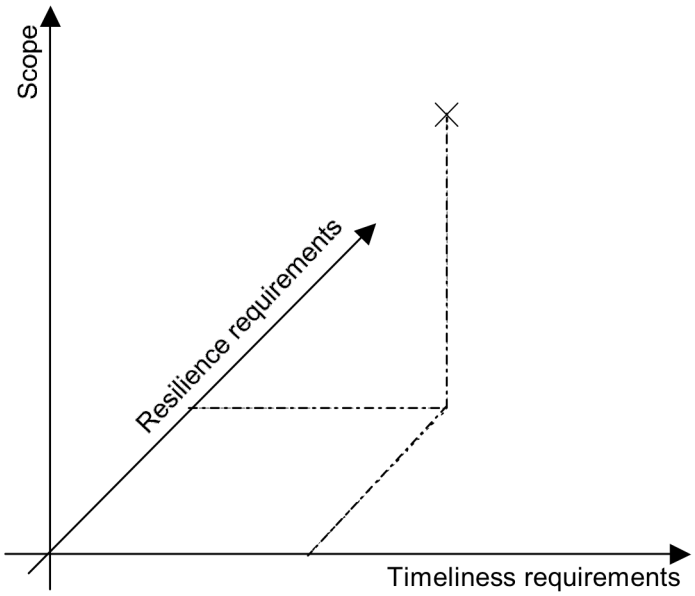


Figure 8: Classification of the functionalities of the MW oracles

The architectural solutions to address (local) resilience and timeliness requirements can be mainly identified as (see Figure 9):

- *separation* of wormhole and payload hardware and operating system;
- same hardware for wormhole/payload, wormhole and payload operating system separated through *hardware virtualization*;
- same hardware and same operating system for wormhole and payload; in this case we need an OS with real-time scheduling and mechanisms to guarantee that payload subsystem can interact with wormhole *only* through the wormhole gateways.

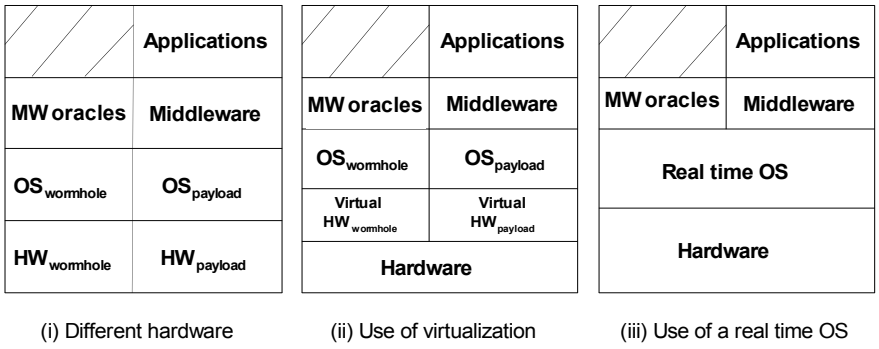


Figure 9: Three different methods to address local resilience and timeliness requirements of the MW oracles

MW oracles based on functionalities with a distributed scope also need a special communication channel (wormhole channel), with better guarantees (in terms of losses, delays, stability, etc.) with respect to the payload network.

4.3.2 Classification of the Applications

As explained in the previous sub-section, in HIDENETS we propose to use a hybrid distributed systems model and we construct the system following the architectural hybridization paradigm. This is important for the dependability objectives of HIDENETS because it allows developing new approaches to deal with the assumed faults, and it allows achieving increased guarantees on the correct behaviour of applications. However, the approaches and mechanisms that may be used to achieve these improvements also depend on the characteristics of the considered applications. In fact, depending on the properties of the application it

might be possible to employ different mechanisms, or combinations thereof, in order to implement those applications with the increased dependability guarantees.

We provide some examples of classes of applications that could be implemented in a more dependable way, provided they could use the help of some services, implemented as MW oracles, namely: Fail-safe, Time-elastic and Time-safe.

Fail-safe class: Includes applications that can switch at any moment to a fail-safe state and remain there permanently, such that the system's safety properties remain valid. This kind of applications cannot be implemented in generic distributed systems with uncertain timeliness, since in these systems it is not possible to limit the time it takes for the application to switch to the fail-safe state. However, with the existence of an MW oracle service devoted to the timely detection of failures and execution of handler functions, this would become possible and, therefore, this class of application could be implemented with the necessary dependability guarantees.

Time-elastic class: Includes applications whose bounds can be increased or decreased dynamically, such as QoS-driven applications. In HIDENETS some applications can be considered to be part of this class. They can be implemented in a dependable way if they adapt according to some well known criteria. In concrete, a relevant criterion is ensuring that an assumed bound is always satisfied with a stable coverage, even if this requires the bound to be increased or decreased over time. To achieve such a dependable behaviour it is important to have correct and actual context information, that is, about the state of the environment, which can be provided if there is an MW oracle that provides this accurate measurements and monitoring information through the wormhole gateway.

Time-safe class: Includes applications whose safety does not depend on any timeliness property. This is a relevant application class in the sense that it characterizes applications in which it is possible to fully separate logical safety properties from timeliness properties. In this kind of applications the code implementing logical safety properties does not depend on time (e.g., timeouts), which means that it is possible to guarantee that if any timeliness property is violated (e.g., due to a timing fault), this will have no impact on any of the safety properties, which will always be secured. If an application of the time-safe class is also of the time-elastic class (see above), it will be able to cope with faults during the process of adaptation, and still ensure that some other important properties (usually QoS-related properties) still remain valid as an outcome of adaptation.

5. Middleware Level Challenges and Services

Middleware solutions can be used as building blocks for the development of complex systems and applications. They provide enriched functionalities and an abstraction of the underlying processing and communication infrastructures properties.

In this section, we first address the main challenges attached to the middleware level. These challenges allow identifying the fundamental services to be provided in order to meet the requirements attached to the considered challenges.

We reason about the fundamental services of HIDENETS middleware, from a dependability perspective. The objective is to identify and discuss the relevance of each of these services for the dependability and resilience goals of HIDENETS. For that, we focus on specific dependability attributes that can be improved with the help of each service and we explain how it can be used and what it does in order to achieve such improvements. We also address possible requirements that might have to be fulfilled by an application in order to benefit from the provided services.

In the presentation, we follow the architecture used in HIDENETS, which, as explained in Section 4, considers that services can reside in two different subsystems: MW oracle services in the wormhole subsystem and general middleware and communication-related services in the payload subsystem. The text addresses in the first place the services belonging to the more trusted and timely group, the MW oracles, and then the remaining middleware services. This structuring is also convenient because it is in accordance with the dependency flow that is observed between some of the described services.

5.1 Challenges for the Middleware

The middleware level is the place within the system architecture in which a variety of services can be constructed to abstract specific details of lower levels and to enrich the functionalities provided to applications. From a dependability point of view, the middleware level can play an important role in the management of existing resources as to meet application demands or expectations with respect to dependability attributes.

In HIDENETS, a number of services is proposed for the middleware level, which have been found to be necessary to provide important dependability properties, such as availability or reliability. These services are addressed in this section, where a discussion is provided about the relevance of each of these services from a dependability point of view.

Important issues related to the design and development of middleware services for dependability purposes, are the characterization of the environment in which these middleware services will be used and the identification of concrete properties that the designer wants to secure. While the former essentially corresponds to making assumptions about the execution environment (hardware infrastructure, operating system, networks, number of entities, geographic scale, mobility, and so on), the latter is determined by the applications that are supposed to directly make use of the middleware. Meeting the latter while taking into account the former can be a very challenging task.

In the following subsection, we describe a set of properties that were identified in HIDENETS as relevant for the set of applications and use cases that are included and described in deliverable D1.1 [76]. The challenge in HIDENETS is to meet these properties while assuming communication environments made of *ad hoc*/wireless multi-hop domains as well as infrastructure network domains. In such environments it is necessary to account for the possible unreliable communication, for the potentially high number of users, which requires scalable solutions, and for the openness of the environments, which makes systems subject to both accidental and malicious interferences.

5.2 Middleware Level Properties

Middleware level properties describe issues that can be used as Quality-of-Service measures for middleware performance. The properties identified in HIDENETS are:

Timeliness of data: Refers to the freshness of data, which is of most relevance when this data is received from entities in the environment, whose value is continuously varying over time. This property is necessary when some application needs to ensure that it is up-to-date (within some given bound), with respect to the context or environment in which it is being deployed. Timeliness of data is also necessary for real-time applications in general (e.g., like video conferencing, on-line gaming). Timeliness imposes constraints on communication delays to lower layers. Quite clearly, it also requires a minimum level of connectivity. Timeliness of data is compromised if pre-established bounds are not met in run-time.

Logical consistency: This property is essentially relevant when considering replication of data. It ensures that the state (the value) of every replica is consistent with each other. Consistency can be further defined as strong consistency, when every replica provides the same response to a distributed query, or as weak consistency, which allows for some replicas to be delayed with respect to others.

Temporal consistency: This property is essentially relevant in the context of real-time data representation. In brief, it ensures that at any point in time the value of some (real-time) entity stored at a replica is not too far apart from the real value of that entity at that same point in time [25]. This applies to so-called *time-value entities*, that is, entities whose time-domain and value-domain correctness are inter-dependent. The notion of temporal consistency implies the definition of a validity constraint for the computer representations of the time-value entities (the values stored at a replica). This constraint expresses the maximum allowed difference between the representation of the time-value entity and its real value and depends on the required consistency, which is imposed by the application.

Trustworthiness of data: The concept of trustworthiness refers to the degree of confidence a service user may have that the service will perform as expected and, in particular, that it will satisfy a set of security properties. In [5], trustworthiness is considered as a similar concept for dependability. A trustworthy service is dependable with respect to security properties. In the context of communication, trustworthiness of data may be regarded as the confidence one may have that security properties for transmitted data are preserved. Some important secondary attributes of dependability that should also be considered in this context include *accountability*, *authenticity* and *non-repudiability* (see Section 2.2 for precise definitions). The threats to trustworthiness include attacks from crackers or insiders, environmental disruptions or human and operator errors. The typical means by which dependability is achieved (see Section 2.4) can also be applied here.

Robustness: This specialised secondary attribute of dependability characterizes systems that are dependable with respect to external faults. Therefore, the robustness of middleware solutions is especially meaningful when external faults constitute a relevant threat.

Message ordering: Different properties with respect to message ordering can be considered. FIFO ordering guarantees that messages are delivered in the same order they have been sent. Causal ordering guarantees that if a precedence relation exists between any two delivered messages, they will be delivered according to that precedence. Total ordering guarantees that any two messages are delivered to any recipients in the same order. Temporal ordering guarantees that messages are delivered in the order of their send timestamps (which requires some form of clock synchronization to implement a useful solution). All these properties are of relevance in the context of group communication.

Completeness, accuracy and timeliness (of failure detection): The requirement for failure detection can be specialized with particular properties that failure detectors must exhibit. The notions of completeness and accuracy to classify different classes of (crash) failure detectors have been introduced more than a decade ago by Chandra and Toueg [26], and are widely used in the distributed systems community. *Completeness* refers to the ability of a failure detector to detect every failure that occurs. *Accuracy* refers to the ability of the failure detector to not make mistakes, that is, wrongly detect failures when they do not occur. Another property that is relevant to further distinguish classes of failure detectors, in particular in systems that are not purely asynchronous, is the *timeliness* property. Timeliness refers to the ability of the failure detector to detect failures within given time bounds.

5.3 From Challenges/Properties to Services

An appropriate way to identify potentially interesting and relevant services for the HIDENETS middleware consists in following a top-down approach, considering use cases and their associated dependability requirements, and understanding how they can be addressed. This exercise has been done in HIDENETS, and now we summarize the most important conclusions, leading to the identification of a number of relevant services, the ones that we address in this section. In any case, the specific relevance of each service is further detailed hereafter.

We use the three use cases presented in Section 3.2, which we consider to be the most representative for this purpose: the *Platooning* the *Car accident* and the *Assisted transportation* (See also Deliverable D1.1. [76] for more details).

Platooning: in this use case, dependability is significantly dictated by the satisfaction of timeliness requirements and consistency requirements (which may also depend on the timeliness aspects). This application has also some requirements in terms of security aspects, in particular authenticity of data. Finally, the application should be robust to failures, reacting to them in a fail-safe manner. This calls for services aimed at providing improved timeliness guarantees and improved awareness of the temporal properties of the system and communication. For instance, accurate clock reading and measurement services may provide such improved awareness, as well as services aimed at evaluating the freshness of data. Improved timeliness may be achieved if services are implemented in a timelier part of the system. Failure detection, for instance, should be done in a timely manner in order to better secure the requirements for a fail-safe reaction to failures. Authenticity calls for a service providing authentication functionalities.

Car accident: in this use case we have to consider the dependability of a Black Box application and of an Emergency communications application. To improve the dependability of the Black Box application it is fundamental to improve availability aspects and ensure privacy and confidentiality. This may be achieved with the help of services aimed at developing trust relations and improving cooperation (and hence possibilities of increased redundancy and availability). Consistency is also important, and thus the possibility of evaluating consistency constitutes a mean to improve this aspect of dependability. While the Black Box application does not have stringent requirements for timeliness, these appear as one of the most important aspects for the dependability of the Emergency communications application. However, since there are no safety-critical aspects involved (in contrast with the Platooning application), dependability improvements may be achieved by better ensuring some desired quality of service levels, which may be done with the help of appropriate QoS oriented services.

Assisted transportation: consistency of information (both on the value and temporal domains) is fundamental to improve the dependability of most applications included in this use case. The authenticity of information is also important for the dependability of these applications (e.g., for the Maintenance and Software updates application). It is clear that, once again, the availability of services providing improved awareness of timeliness aspects and improved precision of time-related metrics can be relevant to improve dependability. The use case also calls for the availability of an authentication service.

Given the above observations, the following services have been considered as the fundamental ones to be developed as MW oracle services, due to their timeliness or security requirements, and also due to their characteristics (simple, generic):

- Reliable and self-aware clock
- Duration measurement
- Timely timing failure detector
- Freshness detector
- Authentication
- Trust and cooperation

Another set of services includes typically more complex services, not requiring the satisfaction of timeliness or security requirements. These are considered as general middleware services, and are the following:

- Diagnostic manager

- Reconfiguration manager
- QoS coverage manager
- Replication manager
- Inconsistency estimation
- Proximity map
- Cooperative data backup

The following subsections address each of these services from a dependability perspective. We try to keep the discussion as general as possible by abstracting from the specific applications and use cases for which the services could be useful. We start by addressing MW oracle services, which may be characterized as basic building blocks, and then, in sequence, we address the remaining middleware services, some of which may use the previously described MW oracles.

5.3.1 Reliable and Self-Aware Clock

In distributed, open, dynamic and pervasive systems, many applications may have critical aspects to deal with, in order to provide a dependable (e.g., safe) service. Examples of such aspects are: i) temporal order delivery (for example, the physical time of sensor readings in data fusion process); ii) temporal consistency; reduced and reliable transmission delay. These applications are usually time-dependent and intensively use timestamps. Timestamps can be obtained by reading the local clock of the nodes of the distributed system. Time measurements can be obtained through these timestamps. Clock synchronization is thus a fundamental process. Clocks, due to their imperfections, drift from real time, and their drifts may vary over time due to several causes. There is thus the need of synchronizing the local clocks among them or to a time reference, in order to enforce and maintain accuracy and precision bounds. Usually, systems assume *worst-case bounds* that are necessary constraints that allow distinguishing unreliable biased data due to poorly-synchronized clocks, from reliable data collected when clock synchronization is good. However these bounds are usually pessimistic values, far from the medium case.

The *Reliable and Self-Aware Clock* (R&SA Clock) is the component that provides an abstraction of the local clock. Its main task is monitoring the synchronization level of the local clock with respect to a global time reference (e.g., TAI, International Atomic Time). The R&SA Clock provides information about time and current *precision level* of the information provided.

The R&SA Clock service is available both for the applications and for other higher level middleware services.

The provided information is such that the actual real-time is within the interval [MinTime; MaxTime], and LocalClockTime is the most probable value of actual time. LocalClockTime is always within the interval [MinTime; MaxTime].

The time interval [MinTime; MaxTime] is a dynamic variable: it changes over time; in fact it depends on the synchronization mechanism(s) used and on the current quality of such synchronization. The interval is usually *symmetric* with respect to the medium (LocalClockTime value); the behaviour of the most used external clock synchronization mechanism is in fact to synchronize the local clock with a given level of accuracy (*Accuracy*).

In this case we have:

- MinTime = LocalClockTime - Accuracy
- MaxTime = LocalClockTime + Accuracy

In general we can imagine to use synchronization mechanisms characterized by asymmetric interval [MinTime; MaxTime].

In order to quantify the improvements in resilience of a system that uses the Reliable and Self-Aware Clock we can proceed in this way: we can compare an *a priori* estimation of the quality of used clock synchronization mechanism, with respect to the dynamic estimation made by the R&SA Clock. Let us assume that we may estimate that clock synchronization mechanism used in the system allows

synchronization with a worst-case accuracy *alpha* (with some assumption on the system behaviour). We can compare the estimation of the quality of the synchronization made by the R&SA Clock (that is a variable over time) with the constant value of accuracy *alpha*. Using R&SA Clock we obtain that actual time *t* is within an interval [*MinTime; MaxTime*], interval in general smaller with respect to [*C(t)-alpha; C(t)+alpha*] (where C(t) is the time seen by the local clock in the instant of time *t*). We can also have situations in which the assumption made for the clock synchronization mechanism is not respected, and we can thus have situations in which the actual accuracy of the clock may be over the assumed maximum *alpha*. This is really critical in case of a distributed application with hard real-time requirements: in this case this incorrect behaviour of the clock may also result in a *catastrophic failure.* Just to make a simple example, we are in this situation during the initialization of the clock synchronization mechanism. The R&SA Clock can correctly report the particular situation, e.g., during initialization it reports a value C(t) (the current view of the local clock), with special limit values for *MinTime* and *MaxTime* as [-∞;∞]. Furthermore, using appropriate estimation of the *MinTime* and *MaxTime* values, we may need less assumptions on the system behaviour: for example, in order to be able to *a priori* estimate a maximum value for the accuracy of a distributed clock synchronization mechanism, we have to assume a maximum delay in the communication, whereas this may be not the case if we use a suitable R&SA Clock.

In fact, the R&SA Clock should be implemented as an MW oracle, within the Resilience kernel. The most important difference between having a service in the wormhole or in the payload subsystem is related to the *trustworthiness attached to the information provided*. In the case of the R&SA Clock we need to assume that the component is able to provide information on which we can place *more confidence* with respect to a time service in the payload subsystem. We can imagine that the service provided by R&SA Clock can suffer only of a specific class of failure (e.g., crash failure): in this case, the information, if provided, is correct. This is the most important difference with respect to a time service in the payload part of the architecture, which can suffer of less restrictive classes of failures (e.g., timing failure, value failure, etc.).

5.3.2 Duration Measurement

The objective of this service is to provide duration measurements of local and distributed activities, with a *bounded precision* on the measured durations. We distinguish between local measurements, associated to the execution of local functions, and distributed measurements, associated to distributed executions involving communication activities through some network.

The ability to measure durations, or time intervals, is usually important in every system and in every situation in which the notion of time is relevant. For instance, it is likely necessary in applications with timeliness requirements or performance requirements, whose satisfaction may be better or easily achieved if the application is able to measure durations and become aware of how well or how timely it is performing. In practice it is usually simple to measure time intervals using the timestamps provided by a single local clock. We would call this a “normal” duration measurement service. However, to measure durations with certain *guarantees about the measurement precision*, and to do it both in a local and in a distributed way, is a different and more complex problem. So there should be some benefits in doing this.

There are in fact potential benefits from a dependability point of view, if one can use a measurement service that provides guarantees on the precision of returned values. The following examples illustrate those benefits:

- A duration measurement service can be used as the basis for the implementation of different kinds of failure detectors, namely those concerned with the detection of faults in the time domain (crash, omission or timing). The accuracy of these failure detectors clearly depends on how well they do their job (on the ability to avoid mistakes while quickly detecting all failures), which in turns depends on how well they are able to measure the durations. A precise notion of the measurement precision can be important to provide awareness of the quality of failure detection. It should be noted that failure detection is a basic service required by most applications with dependability concerns. Therefore, these dependability needs can be addressed more easily if one is able to characterize the precision of the measurements upon which these applications rely.

- Many applications rely on monitoring services to make decisions about when and how they should adapt. When monitoring is about measuring delays or durations, then it may be interesting, from a dependability point of view, to rely on the advocated duration measurement service, which will allow to characterize how precise is the information provided by the monitoring service and hence to make better adaptation decisions.

While the provision of guarantees on the measurement precision is a fundamental characteristic from the point of view of dependability improvement, it is possible to argue about additional benefits of a duration measurement service, if we take into account that this service is to be implemented, as advocated in the HIDENETS reference model, as a more trusted and timely service. In this case, we have in addition that some guarantees can be given on the timeliness of the measurements, which means that as soon as a measurement is completed it is made available at the service interface without any other delays. Such timeliness is particularly relevant when it is ensured at the interface. For instance, if accessed or used by another service within the same subsystem, (e.g., the timing failure detection service), the timeliness is not lost and the measured duration becomes not only precise, but also timely (thus allowing, for instance, timely timing failure detection). In general, the availability of timely information is also relevant when it can be used in a timely way. Assuming that systems and applications are built according to the proposed reference model, then this will be important from a dependability point of view because it will allow to provide the timeliness guarantees for the effectively important or critical parts using modular and simple constructs, partially based on this duration measurement service.

5.3.3 Timely Timing Failure Detector

Timing failure detection allows applications or middleware services relying on it to know if timed actions have been executed in a timely way or if they have incurred in a timing failure. In many works in the literature, failure detection refers to crash failures. The detection of *timing failures*, on the other hand, is particularly relevant when considering applications that impose some timeliness constraints and when, at the same time, the considered operational environments are uncertain and do not allow establishing upper bounds with the necessary coverage on the time it takes to execute basic operations, be it communication or processing related.

To understand the importance of detecting timing failures from a dependability point of view, it is necessary to observe what can possibly happen to an application if these failures are not detected. The immediate effect of timing failures may be twofold: unexpected delay and/or incorrectness by contamination.

Unexpected delay is defined as the violation of a timeliness property. That can sometimes be accepted, if applications are prepared to work correctly under increased delay expectations, or it can be masked, by using timing fault tolerance.

However, there can also be *contamination*, which is defined as the incorrect behaviour resulting from the violation of safety properties as a result of the occurrence of timing failures. Clearly, the dependability of an application can be improved if one is able to address this contamination problem. In fact, if the application is constructed using an adequate algorithm structure, then the availability of a timely timing failure detection service can be useful to avoid contamination. To provide an intuition on this, suppose that the system does not make progress without having an assurance that a previous set of steps were timely. Now observe that "timely" means that the detection latency is bounded. In consequence, if it waits more than the detection latency, absence of failure indication means "no failure", and thus it can proceed. If an indication does come, then it can be processed before the failure contaminates the system, since the computation has not proceeded.

For instance, in applications involving replicated components whose state must be kept consistent, a timing failure on one of the replicas when updating the state of the replica set will necessarily affect their consistency. In such cases, the availability of a timely timing failure detection service is important to allow the affected replica(s) to detect that a timing failure occurred and take some action to prevent the inconsistency to be externally observed, like simply stopping to produce further outputs.

In a more general sense, it is possible to say that any application with a fail-safe state, to which it can switch upon the occurrence of a timing failure, can take advantage of a timing failure detection service. We say that these applications belong to a *fail-safe class* of applications (see Section 4.3.2). In the case of safety-critical

applications, switching to the fail-safe state must usually be done in a timely way when the failure occurs. In such cases, the timeliness of the timing failure detector becomes crucial. This is one of the reasons why we consider, in HIDENETS, that this service must be implemented in the more trusted and timely part of the system. We should note, however, that in order to obtain a dependable system it is not sufficient to provide a timely indication of the failure at the service interface. Any fail-safety procedures that are executed in reaction to this failure indication must also be implemented with the same timeliness assurances.

In addition to the previously mentioned effects of timing failures, we also mention the *decreased coverage* effect. However, since it cannot be addressed with the help of a timing failure detection service, we do not elaborate any further here. The means to address this effect are mentioned in Section 5.3.9 concerning the QoS coverage service.

5.3.4 Freshness Detector

The detection of messages delayed more than a given threshold — called "message freshness detection" — is an important requirement for many distributed critical real-time systems. The Freshness Detector is a specific software component that supports the detection of the freshness of the exchanged messages. This service detects if fresh-enough data are available, in order to allow the application/middleware to react to the absence of fresh-enough information.

When the Freshness Detection is used to inform the application of the absence of fresh-enough data, specific interfaces and real-time mechanisms must be used in order to guarantee a *prompt and timely reaction*. This service can be needed also in safety-critical applications: the absence of fresh-enough data, without a prompt reaction from the application, can bring the system to an unsafe state. It is thus important to obtain proper estimation of the freshness of available data (e.g., for such applications, a pessimistic estimation, always greater or equal to the real freshness value, can be the proper one).

Several distributed systems with real-time requirements need to use fresh information and, in addition, they need to detect the *freshness level* of available data. Message freshness detection is an instance of the more general problem of *failure detection* in which *timing failures* are considered (instead of more traditional class of crash failures). Examples of systems for which it is necessary to check the freshness of the exchanged data are transportation systems that use communication for data (like position/speed) exchange. For instance, considering the Assisted Transportation use case, and the hazard warnings application in particular, we have requirements on the freshness of data received by the cars "geographically" near to the car that raised the alarm. An old information (e.g., old information about car position or speed) that reaches the application level can lead to incorrect choices, that may eventually lead to catastrophic consequences. The level of freshness of the data received has to be assessed to guarantee the safety.

The Freshness Detector can be used to instantiate a *logical connection* between two end-hosts with freshness requirements and to check the freshness of data exchanged on this connection. The Freshness Detector can also check violations of freshness of data exchanged between a group of processes (multicast and broadcast messages). The Freshness Detector is an end-to-end service: we can define complex freshness requirements, using logical predicates of simple requirements based on freshness as seen by single participants.

The Freshness Detector obtains the real-time requirements of the logical connections directly from the involved entities. The same application or middleware component can require the Freshness Detector to instantiate several data flows, and each dataflow can be defined with its specific freshness requirements. When the predicate on the freshness of the connection is not satisfied, the application has to be informed to allow proper reaction through its own fault management policy; therefore, it is of utmost importance to define the temporal behaviour of the service. The Freshness Detector must be able to inform the application of absence of fresh enough data in a timely way, respecting the real-time requirements of the application, which can be strict. For example, let us suppose that an application, in order to guarantee the safety of the system, needs a data flow respecting a real-time requirement on the freshness of the data. This application can require the Freshness Detector to instantiate a logical connection between the two end-hosts, and can define the predicate of freshness on this data flow. When the Freshness Detector detects the absence of fresh enough data on the flow, it has to react, informing the application, in a timely way (e.g., respecting some

predefined real-time limit). In other terms, from the application level perspective, it is always needed to be able to switch, in a limited and known time, to a fail-safe state.

This service is useful for many applications with real-time requirements, e.g., applications for mobile systems in which exchanged information has a predefined "lifetime".

As for other failure detection mechanisms, the quantification of the service provided by a Freshness Detector may be obtained by estimating the following metrics:

1) Probability of false positive.
2) Probability of false negative.
3) Time between the detection of a freshness failure and the reaction of the system: this is really important for critical applications, in which a correct detection of a freshness failure must be immediately (timely) followed by a prompt reaction by the system (e.g., switching the system to a fail-safe state, if it exists).

5.3.5 Authentication

Proper authentication of participating entities in a communication session is a fundamental feature in an environment such as the one provided by HIDENETS. Entities involved in a HIDENETS-like scenario (e.g., cars) have to trust the collected information in order to make safe operational decisions which, gone awry, can have catastrophic consequences such as car accidents. This implies a need for some mechanism that enforces authentication of participating entities, and facilitates the integrity and confidentiality properties in communication.

The Authentication MW Oracle (AO) pertains to the trustworthiness of information. To target this high-level problem, the pivotal functionality of the AO is to guarantee that communicating entities are really who they claim to be. If some car says it is A, then it has to show some proof that it is really A. The same is true for the servers in the infrastructure network. If some server says it is B, it has to present some proof that it is B.

This should be accomplished with the use of digital certificates. HIDENETS should employ a public-key infrastructure (PKI) with a certification authority (CA) responsible for issuing X.509 certificates that bind public keys to entities. Assuming that every entity trusts this CA, and can verify its signature, then it can also verify that a certain public-key belongs to that entity which claims so.

Additionally, the AO exposes auxiliary functionalities in order to ease the enforcement of secure communication. It can sign/verify messages, and encrypt/decrypt them using either public-key or symmetric cryptography. It can also negotiate a symmetric key between a group of entities for either secure point-to-point or secure group communication sessions.

The existence of this service as a MW oracle is justified by the improved trustworthiness it provides if designed as such. The AO is responsible for holding sensitive information such as secret cryptographic keys. Even if the system (in which an AO is deployed) is compromised, this ensures that this sensitive information is never leaked to intruders. An intruder may have the ability to use the functionalities provided by the AO as long as it controls the system, but as soon as the effects of the intrusion are removed, there is no additional need to re-negotiate secure communication sessions, or ask a CA for a new private/public key-pair. A second reason for the MW oracle approach is the fact that this service is supposed to be used by other MW oracles (e.g., the Reliable and Self-Aware Clock of Section 5.3.1), and this forces the authentication service to provide, at least, the same dependability coverage as these MW oracles.

5.3.6 Trust and Cooperation

The Trust and Cooperation MW Oracle (TCO) is a basic service for cooperative applications. A cooperative service emerges from the cooperation of entities that are generally unknown to one another. Therefore, these entities have no a priori trust relationship and may thus be reluctant to cooperate. In cooperative systems without cooperation incentives, entities tend to behave in a rational way in order to maximize their own benefit from the system. The goal of the trust and cooperation oracle is therefore to evaluate locally the level of trust of neighbouring entities and to manage cooperation incentives.

Synergy is the desired positive effect of cooperation, i.e., that the accrued benefits are greater than the sum of the benefits that could be achieved without cooperation. However synergy can only be achieved if nodes do indeed cooperate rather than pursue some individual short-term strategy, i.e., being rational. Therefore, cooperative systems need to have cooperation incentives and rationality disincentives.

In the literature, one can find many cooperation incentive schemes that are diverse not only in terms of the applications for which they are employed, but also in terms of the features they implement, the type of reward and punishment used, and their operation over time. Cooperation mechanisms can be classified into: remuneration-based and reputation-based mechanisms.

The estimation of reputation can be performed either centrally (as with the eBay system for example) or in a distributed fashion, e.g., see [77]. In a centralized reputation system, the central authority that collects information about peers typically derives a reputation score for every participant and makes all scores available online. In a distributed reputation system, there is no central authority for submitting ratings or obtaining reputation scores of others. However there might be some kind of distributed stores where ratings can be submitted. Most of the time in a distributed architecture, each peer estimates ratings autonomously. Each peer records ratings about its experiences with other peers and/or tries to obtain ratings from other parties who have had experiences with a given target peer.

The centralized approach to reputation management is not fault-tolerant and furthermore not scalable. On the other hand, with the decentralized approach, it is often impossible or too costly to obtain cooperation evaluations resulting from all interactions with a given peer. This is the reason why reputation is generally based on a subset of these evaluations, usually obtained from the neighbourhood.

In contrast to reputation-based mechanisms, remuneration based incentives [78] are an explicit counterpart for cooperation and provide a more immediate penalty to misconduct. Remuneration brings up requirements regarding the fair exchange of the service for some form of payment. This requirement generally translates to a more complex and costly implementation than for reputation mechanisms. In particular, remuneration based mechanisms require trusted third parties (TTP) such as banks to administer remuneration of cooperative peers; these entities do not necessarily take part in the online service, but may be contacted in case of necessity to evaluate cooperation. Tamper proof hardware (TPH) like smart cards and related secure operating systems have been suggested or used to enforce in a decentralized fashion the fair exchange of the remuneration against a proof that the cooperative service was undertaken by a peer node.

In the automotive application context, it makes sense to consider a solution based on specialized hardware. Henceforth we envision that solutions based on the use of TPH will be preferred for their simplicity.

The solution proposed in HIDENETS for the TCO proposes a cooperation policy based on the use of a smartcard. The smartcard is used for generating, storing and verifying cryptographic signatures. The TCO thus uses the smartcard for verifying that: 1) a device follows the cooperation policy defined, and 2) that the software it runs is genuine, i.e., that the software is signed using the appropriate signature.

5.3.7 Diagnostic Manager

The Diagnostic Manager (DM) is the first “non-oracle” service we are connsidering in the following subsections.

The DM significantly contributes to dependability improvement in the system, being in charge of managing all the activities necessary to judge whether the system (or parts of it) is (are) working properly[4] or not. DM contribution to dependability becomes more evident considering that DM feeds the Reconfiguration Manager service.

Three fundamental classes of faults are considered, which may affect any kind of component in the node architecture (hardware, software or communication link):

[4] An “improper” (unsuitable) behaviour could be the result of: i) the manifestation of a fault affecting the component, which leads the service provided by such a component to depart from its functional specification, or ii) it could be determined by a change in the requirements of the application using the service provided by the component itself, or iii) by changes in the environment (e.g., system load) which leads to a change in the QoS provided.

- omissive: a service suffers an *omissive* fault when it is expected to do something, but it never does;
- assertive: a service suffers an *assertive* fault when it provides a service, but the service provided does not satisfy its specifications (e.g., because it produces an out of bound value, or it does not meet its timing requirements, if any);
- arbitrary: a service suffers an *arbitrary* fault when no assumptions are made on the way it fails (e.g., inconsistent/Byzantine faults); this class includes also the previous ones.

Basing on the supposed fault model, error/deviation-detection activities produce information supporting the DM; that information is made available in specific repositories or directly sent to the DM itself. Here, a reactive vs proactive scheme can be considered: in a reactive scheme, detection is explicitly requested by the DM; in a proactive scheme, the DM gathers available detection results as soon as the error/deviation-detection produces them.

Considering the available error/deviation-detection messages, diagnosis has to continuously assess the status of certain components/services. Diagnosis is classified using the following classes:

1. *Local diagnosis on local resources/services*: each node performs local diagnosis in order to judge the status of some components/services inside the same node.
2. *Local diagnosis on remote components/services*: each node aims to quantify the local perception of some remote services, basing on information available in the node.
3. *Distributed diagnosis*: is aimed to build a common view about the status of other nodes in the system. Distributed diagnosis has to be resilient to Byzantine behaviours, so it is very resource consuming (both in time and communication); therefore distributed diagnosis has to be performed only when strictly necessary, involving only a "group" of nodes in the system. The group definition is application dependent, and raises the problem of membership and (possible) leadership.

System nodes can be classified into two categories: fixed and mobile ones; each category has its functionalities and capabilities, so specific diagnostic solutions need to be developed.

5.3.8 Reconfiguration Manager

Reconfiguration is a fundamental activity in fault-tolerant systems, because dynamically reacting to diagnosed problems by performing proper system reconfiguration increases system availability and dependability. Reconfiguration may be triggered as a reaction to either the presence of diagnosed faults or to diagnosed changes in the environment. Therefore reconfiguration effectiveness is primarily influenced by correct diagnosis and secondly by the selection of the proper reconfiguration.

The "Reconfiguration Manager" (RM) is specifically in charge of deciding both when some reconfiguration is needed and which reconfiguration policy has to be applied. The RM, basing on information coming from the DM and from the "QoS Coverage Manager", has to:

1. bring back the system to provide correct (although possibly degraded) services after the occurrence of some malfunctions (some form of redundancy should be in place to replace faulty components);
2. properly manage system resources in order to provide the required QoS levels after the occurrence of some deviations from expected QoS;
3. decide on the application of pre-defined preventive maintenance policies, possibly alerting the operator service in order to physically repair/replace modules which fail to satisfy the maintenance tests and, in such a case, implementing the appropriate reconfiguration of the involved subsystem.

Reconfiguration activities are classified into two categories:

1. *local reconfiguration*, triggered by local diagnosis to satisfy local concerns;
2. *global reconfiguration*, triggered by distributed diagnosis to satisfy global concerns; when global reconfiguration is requested, each node in the involved group has to perform local reconfiguration basing on its competency.

Reconfiguration can be performed according to different approaches; the extremes are described in the following and intermediate solutions are possible as well:

- *static approach*: a set of reconfiguration strategies are defined at system design time, and statically associated to specific patterns of faults/deviations of a number of system components. This association is performed through a look-up table, which is accessed on-line to retrieve the appropriate reconfiguration strategy. The problem arises of how to sort out from the situation where the fault/deviation pattern, as assessed by the DM subsystem, is not listed in the look-up table (situation not foreseen beforehand).
- *dynamic approach*: many strategies are applicable for the same diagnosed scenario, the choice of the reconfiguration to apply is performed on line through a proper evaluation support, which is fed with the specific system and environment conditions at the time the reconfiguration action is triggered by the DM subsystem. This is of course more accurate than the previous approach, but more costly in terms of time and resources needed to perform the on-line choice. Evaluation relying on a model-based approach looks like a promising solution in this respect; however, it requires methodological advancements to cope with complexity and fast solution methods, to make the approach feasible in practice. So, although very appealing, the dynamic selection still requires deep research investigations. Its possible implementation will be investigated in HIDENETS, with the on-line evaluator support developed in the context of the technical work focusing on quantitative evaluation of HIDENETS applications and services (see Section 8).

5.3.9 QoS Coverage Manager

As previously mentioned, when referring to timely timing failure detection service from a dependability perspective, timing failures can have several effects, one of which is a decreased coverage. Whenever we design a system under the assumption of the absence (i.e., prevention) of timing failures, we have in mind a certain coverage, which is the degree of correspondence between system timeliness assumptions and what the environment can guarantee. In a system with uncertain timeliness, such as those we consider in HIDENETS, the above-mentioned correspondence is not constant, it varies during system life. If the environment conditions start degrading to states worse than assumed, the coverage incrementally decreases, and thus the probability of timing failure increases. If on the contrary, the coverage is better than assumed, we are not taking full advantage from what the environment can offer. Both situations are undesirable, and this is a generic problem for any class of system relying on the *assumption/coverage binomial*: if coverage of failure mode assumptions does not stay stable, a fault-tolerant design based on those assumptions will be impaired. A sufficient condition for that not to happen consists in ensuring that coverage stays close to the assumed value, over an interval of mission time.

Applications that rely on the assumption/coverage pair, do in fact expect a certain quality of service (QoS) to be delivered (e.g., an assumed timeliness bound, a throughput, etc) with an associated coverage. In that sense, they will express such QoS requirements by means of <bound, coverage> pairs.

The objective of the QoS coverage service is to provide enhanced support for applications that are able to adapt to the available QoS. In other words, the idea is to provide the means for the adaptation to be done in a more dependable way than it would be done by relying on the information provided by simpler monitoring service, disregarding any coverage needs. This kind of applications, which can adapt their behaviour and dynamically assume different timeliness bounds according to the actual conditions of the environment, can be said to belong to a *time-elastic class* of applications (see Section 4.3.2). For these applications, rather than adapting in some *ad hoc* fashion, for instance by doubling timeout values as soon as a timeout is exceeded, it is much more interesting to perform the adaptation based on rigorous information provided by some underlying service. From a dependability point of view, the adaptation process should be done in a controlled way, that is, with exact knowledge of *how much* and *when* it should take place. This is only possible with an accurate characterization of the actual conditions of operation and the available resources at a certain moment.

The impact of the QoS coverage service in terms of dependability improvements will depend on the concrete mechanisms that are used to determine the state of the environment and to calculate the bounds that should be used in order to achieve some required coverage. These improvements might be evaluated by comparing the behaviour of an application in the two situations: using and without using the QoS coverage manager service.

5.3.10 Replication Manager

The RM supports the most common way to improve dependability of an application, i.e., to replicate it. If one instance of the service (usually referred to as master or primary instance/process) fails, backup instances can take over the workload from the failed one.

Naturally, there is some overhead involved in managing replicated services, detecting faults and triggering failover. It will always be a tradeoff between different costs: on one hand, the service replication overhead and on the other hand, the cost for a fault in the system leading to a service failure. In many cases the cost of replication is small compared to a complete service breakdown. To reduce the application development cost it can be beneficial for the application development process that functionalities to support the complex management of replicated services be included in the middleware supporting the high-availability application.

For stateful services, it is not enough to just run several instances. An example of a stateful service is a billing service used by a telephone operator to keep track of how many minutes the costumers have been using the service. The charging information has to be kept up to date also on the replica servers. Otherwise a service failure could potentially lead to loss of revenue if the latest charging information is lost during the failover to a backup server. To combat this situation, the state information must be shared between the instances so that all instances have an up to date picture of the application state. This in turn involves more complex management routines.

Some services are in such high demand that they are replicated not only because of availability reasons but also to share the workload on several servers. In such networks with a common load balancing function splitting the workload, it is beneficial to divide the servers into groups that share their state. The best group size can be determined based on information about the server mean time to failure and mean time to repair plus the target level of availability. Grouping servers in small groups is advantageous especially if the servers are stateful and need to share state updates with the rest of the group. This leads to savings not only in network resources, but also in storage on the nodes. If the set of servers that share state variables is very large, the resources needed to store all states can become a major problem due to hardware limitations of the servers.

In addition when using servers in state sharing groups it is important to keep track of the group memberships for the individual servers. Functionality to determine and control membership information becomes a crucial part of the middleware. The added complexity in management is earned in reduced network utilization because the servers only need to send state updates to their other group members. This reduction in network load is especially important in wireless networks.

Having split the servers in smaller groups also helps improve the probability of reading a state that is inconsistent compared to the state at the master server. Based on measurements of certain performance metrics such as network delays the probability of inconsistency can be determined. The inconsistency estimation model can be implemented in the network nodes where it can help decide on the selection of primary backup servers.

5.3.11 Inconsistency Estimation

Stateful applications are applications that manage states about ongoing communications, e.g., interactive games, video conferencing, etc. For dependability purposes, replicated stateful servers have to share the service state(s) so that if the primary server fails, the service can be continued at a backup server without losing all information related to previous communications. When the stateful service reads an inconsistent state value, dependability levels can be impacted in different ways: an ongoing session is dropped (inconsistent session state) or the operator loses revenue (inconsistent charging state). Therefore, the state information needs to be propagated timely so that when a secondary server can take over the service provisioning with up-to-date, or consistent, state values.

Each application requires a specific type(s) of logged information to generate state updates, e.g., billing-related logs store different information about the communication than logs used for server failover situations. So in order to evaluate inconsistency properly, we first need to carefully define what inconsistency is in a

given scenario. There are many definitions of inconsistency, because each stateful application has different requirements in terms of state replication and state retrieval. For example, for one application those requirements imply that consistency is the probability that all states are identical at a given time, while another application might require that only the state replica used by a read process is consistent with the expected state value (i.e., the value read is not identical to the value written when the last state update was committed).

The goal of the inconsistency evaluation is to quantitatively estimate inconsistency for the replicated, stateful service. The inconsistency estimation helps determine whether the replicated service at a server is up-to-date (at a given time) or not. The knowledge about inconsistency levels at a given time can be used by other services (e.g., replication manager) to failover the service more efficiently by picking an optimal replicated server or, typically in the distributed black box scenario, to retrieve a consistent image of the road and traffic conditions at accident time. The inconsistency levels estimated by this component can be used by the reconfiguration manager to choose among the replicated servers and pick one as a backup only if it has consistent service state or if the probability for the service state to be consistent is above a certain threshold. The same applies with the cooperative data backup module. The replication manager can also use inconsistency as an input to logically split a large set of replicated servers into subsets. To generalize, this component is relevant to all use cases with replicated stateful services that can use the knowledge about inconsistency levels to optimize performance and/or dependability. Typically, those services are deployed in the infotainment use-case.

Inconsistency levels greatly depend on network characteristics and traffic models; therefore, information about the network context is essential in the inconsistency estimation process. For instance, it is possible to use stochastic models to derive inconsistency levels approximation based on the distribution of read and write events.

5.3.12 Proximity Map

The goal of the proximity map service is to provide a map of neighbouring[5] nodes (along with estimated distance, position and speed). Thus, the proximity map represents the local knowledge a node has about its vicinity. This can vary according to wideness (the number of communication hops represented on the map) and according to accuracy (how often is the map updated? Does it use a reactive or a proactive protocol?).

For many applications, and especially for cooperation-based applications, a node needs to interact with its neighbours. Furthermore, the quality of service that may be provided by a given component can vary according to the vicinity, e.g., the quantity of neighbours, their density, etc. It is then necessary to formalize this view of the vicinity into a proximity map.

The solution proposed in HIDENETS for the establishment of proximity maps is based on the following idea [79]: each node possesses a GPS that is used for establishing its own location, then it uses the following algorithm for exchanging location information, constructing a proximity map and evaluating the accuracy of the information contained in this map.

Each node keeps a map of its knowledge of the location and connectivity of other nodes, which is represented as a graph as shown in Figure 10. This graph is regularly updated when the node receives new location information and is also regularly sent to the node's neighbours in its proximity messages. In this way, a node's knowledge increases over time:

- first, the node knows its own location (level *0*),
- second, when the node receives its neighbours' beacons, including their locations, it knows about its one hop proximity (level *1*),
- then, the node receives beacons including its neighbours knowledge (level *L-1*) and updates its own graph with this information (level *L*).

If messages are sent each τ time units, level L information is ($L*\tau$) old. Because high-level knowledge is older and because it is not desirable to have the knowledge of the whole network, the maximum level of

[5] The notion of neighborhood here refers to geographic vicinity and not networking vicinity

knowledge is bounded. This bound, *Lmax*, is determined dynamically according to the size and the density of the network.

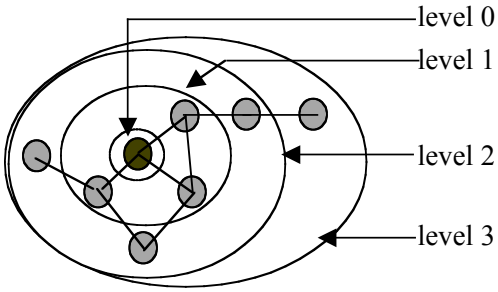


Figure 10: Location knowledge levels

This algorithm is pro-active and enables a node to know the other nodes physically present within the area if *Lmax* is sufficiently large. When *Lmax* is not large enough, or the coverage obtained is not sufficient, a reactive protocol is used to collect location information further than *Lmax* hops.

It should be noted that at time *t*, every node in the network has a different view of the connectivity since its level *1* information is τ time units old, its level n is ($n*\tau$) time units old, etc. If this connectivity information is to be used for some protocol where consensus is necessary, one would consider that at time t each node knows the exact connectivity graph that we had at time $t-(Lmax * \tau)$.

5.3.13 Cooperative Data Backup

The problem of cooperative backup of critical data consists essentially in: 1) discovering storage resources in the vicinity (using the proximity map service), 2) negotiating a contract with the neighbouring cars for the use of their resources (using the trust and cooperation MW oracle), 3) handling a set of data chunks to backup and 4) assigning these chunks to the negotiated resources according to some data encoding scheme and with respect to desired properties like dependability, privacy, confidentiality, etc. On the other hand, it must also take care of the recovery phase, i.e., the data restoration algorithm.

The goal of the cooperative data backup service is to backup critical data despite failures of the data owner and of the nodes storing the critical data for the data owner. The service also needs to be resilient to denial-of-service attacks and other type of non-cooperation attacks such as data retention.

Various replication and data scattering algorithms may be used to implement the cooperative backup service [80]. Replication may be handled by creating full copies of individual data items (we refer to this as *simple replication*) or by more sophisticated *erasure coding* techniques. Choosing between these techniques implies a tradeoff between storage efficiency and data confidentiality

Erasure coding algorithms have been studied extensively in the literature [81-85]. A commonly accepted definition of erasure coding is the following:

- Given a *k*-symbol input datum, an erasure coding algorithm produces $n \geq k$ *fragments*.
- *m* fragments are necessary and sufficient to recover the original datum, where $k \leq m \leq n$. When $m = k$, the erasure code algorithm is said to be *optimal*.

Although not all erasure coding algorithms are optimal (many of them are *near-optimal*), we will assume in the sequel the use of an optimal erasure code where $m = k$. By convention, we denote as (n,k) such an optimal erasure code.

When all *n* fragments are stored on different devices, an optimal erasure code allows $n - k$ failures (or erasures) to be tolerated (beside that of the primary replica). The storage cost (or *stretch factor*) for an optimal erasure code is n/k (the inverse ratio k/n is often called the *rate* of an erasure code). To tolerate a number of erasures *f*, we need $n = k + f$, so the storage cost is $1 + f/k$. Therefore, erasure coding (with $k \geq 2$) is more storage-efficient than simple replication ($k = 1$). For instance, (2,1) and (3,2) erasure codes can both

allow the tolerance of one failure, but the former requires twice as much storage as the original data while the latter only requires 1.5 times as much.

Additionally, when all k fragments are distributed to different devices belonging to different non-colluding users (or under different administrative domains), erasure codes can be regarded as a means for improving data confidentiality: to access the data, an attacker must have control over k contributing devices instead of just one when simple replication is used [86]. This effectively raises the bar for confidentiality attacks and may usefully complement ciphering techniques used at other layers. Similar concerns are addressed by generalized threshold schemes where, in addition to the definition above, less than $p \leq k$ fragments convey no information about the original data, from an information-theoretic viewpoint.

According to an initial setup, the cooperative data backup service implements such an erasure coding scheme. Additionally, when a node has access to the infrastructure, it can connect to a reliable data storage and backup the data fragments it is storing. Both the nodes and the reliable data storage perform some kind of context-aware fragments compression, e.g., discarding old data fragments that won't be useful anymore.

6. Communication Level Services and Protocols

In this section, we first introduce the main challenges related to the communication level. We then identify the fundamental services and building blocks at communication level that are necessary to meet such challenges. We restrict the presentation to services that are particularly investigated in the project and focus the discussion on their relevance for the dependability and resilience goals of HIDENETS in particular with respect to the identified use cases.

6.1 Challenges for the Communication Level

The HIDENETS solutions include functionality both on communication level as well as service-middleware level. As indicated in the simplified node architectures presented in Section 3.3, these levels will be architecturally separated but still closely coupled via well-defined interfaces, which are currently being developed.

Resilience is a key issue for increasing the dependability in communication systems. Network resilience can be defined as the ability of a system/network to adapt to changes like for instance node and link failures and traffic pattern changes. Proper resilience mechanisms will enhance network performance and availability and thus the fault tolerance of the system. In wireless systems defining robust resilience mechanisms is a challenge in itself, and in general such mechanisms may increase the cost of the system quite extensively. A challenge is therefore to find cost efficient mechanisms. As the service requirements and willingness to pay varies a lot, this also means to define a set of mechanisms that matches the individual service needs.

Using multiple channels and/or interfaces in wireless *ad hoc* networks is a means to enhance resilience by increasing the overall network performance and the availability of resources. The combination of this approach with investigation of routing strategies in the *ad hoc* domain will be pursued.

Access to the Internet is vital for many of the use cases of interest for the HIDENETS project. In today's networks, terminals are connected to the Internet using a wired connection or wireless via WLAN hotspots or cellular access. The link between the terminal / *ad hoc* network and the infrastructural part is a crucial factor for the dependability of services in such use cases. Methods are therefore required for the mobile nodes in the network to select the best gateway and to switch seamless to the 'next best' gateway in cases of failure as long as alternate gateways are present.

Another challenge is to ensure that the system detects errors or failures as soon as possible to permit the failover mechanisms to act rapidly and avoid critical delay that could endanger the ongoing communications.

In HIDENETS, a number of services and functional blocks are proposed for the communication level. These are addressed in this section where a discussion is provided about the relevance of each of these services from a dependability point of view and for the challenges described above.

The communication functions as defined here consist of the OSI layers 2-4 (link, network, and transport layers), as well as partly layer 5 (session layer). It is assumed that IP is used at the network layer, while several protocols, including TCP and UDP, may be used at the OSI transport layer. At the underlying link layer, IEEE WLANs in infrastructure and *ad hoc* modes, as well as traditional mobile networks like UMTS and GPRS will be used.

From a dependability point of view we are especially interested in the availability, reliability and resilience of the communication. These aspects are translated into communication level requirements. The communication-level requirements depend on requirements from the applications and middleware. Actual quantification of communication-level requirements will depend on the use case as well since the same application may pose different requirements based on the setting where it is used. In general, the communication functions transport chunks of information from one network node to another. Basically what the middleware does is to abstract some details of the underlying layers for the application running on top.

In what follows, we list some communication-level properties. These can also be quantified to measure the behaviour of the system.

6.2 Communication Level Properties

The communication properties listed here are the ones considered important in the context of HIDENETS.

- *Throughput* is a measure of the amount of data transferred per unit of time. Throughput can be measured either in successfully transmitted packets (or frames) per unit of time or in bytes per unit of time. The throughput may be specified for a single connection or an aggregate. In order to be meaningful, it must be defined at which protocol level throughput is measured and thereby which protocol fields that are considered as data. Throughput is a function of the bandwidth of the underlying medium, overhead at underlying (and possibly present) layers (as viewed from the layer defined for the throughput measurements), the bit error rate, and the protocols and functions for detecting and correcting errors at underlying layers. A related measure is *goodput*, which is the amount of data actually received correctly at the other end per unit of time (i.e., lost and erroneous packets are not counted). Goodput is a function of throughput and protocols and functions for detecting and correcting errors at the current layer.
- The communication *delay* is the time interval between the instant when a packet is sent until it is received at the other end. For a more precise definition see ITU-T recommendation Y.1540[6]. Keeping communication delay at a minimum is very important for some application types, in particular real-time voice and video services. Other applications may tolerate a medium level of communication delay, but are sensitive to variation in the delay (delay jitter). Some interactive applications are very sensitive to exceeding a given threshold. In such cases the goal is not to keep the delay at a minimum, but to always keep it within the threshold.
- Data integrity/packet error ratio refers to the validity/correctness of data received. It can be compromised in a number of ways, mainly by transmission errors caused by the medium or by malfunctions of transmission hardware. The *packet error ratio* is measured as the ratio of number of erroneous IP packets received to the total number of (successful and erroneous) IP packets received. The packet error ratio can be minimized using error detection/correction mechanisms at one or more protocol layers. There are mainly two categories of error detection/correction mechanisms: reactive mechanisms ask for retransmission after a packet error is detected, while proactive mechanisms send redundant information together with the original data. In general, reactive error detection/correction mechanisms increase the delay experienced by the applications, and therefore some applications will not use such mechanisms (e.g., real-time two-way voice and video).
- *Packet loss ratio* is measured as the ratio of number of lost packets to the number of transmitted packets. The packet loss pattern may also be important for some applications. For instance, certain voice and video applications are more vulnerable to loss of consecutive packets than if the packet loss is spread evenly over time.
- *Network congestion* refers to the load in the network. If the network cannot carry all offered traffic, it is congested. Congestion will mostly be related with the amount of generated traffic and is often caused by unbalanced traffic in the network. Normal means to reduce congestion are transport layer feedback mechanisms (e.g., TCP) and traffic engineering mechanisms like rerouting and load balancing.
- *Recovery time* is essential from the availability viewpoint. This network availability is an important parameter in communication networks. It can be defined as the probability that a network can perform its required functions or the proportion of time the network is in the availability state. It is a derived parameter that depends on other primary performance parameters and is mainly related to performance measures like outage intensity and mean time between outages. In our context we talk about failure intensity and recovery time. The purpose of resilience mechanisms is to decrease the recovery time and as such increase the availability. Without proper resilience mechanisms the recovery time would equal the repair time of the actual failure. With resilience mechanisms in place the recovery time will be reduced to the time it takes to detect the failure and then switch over to a backup system. This may be from some ms to some seconds depending on the actual failure and on the resilience mechanism applied.

[6] http:// www.itu.int/itudoc/itu-t/aap/sg13aap/history/y1540/index.html

6.3 From Challenges/Properties to Services

As stated above network resilience is a key issue for increasing the dependability in communication systems. Enabling the use of multiple channels / interfaces / radios is a promising approach to enhance resilience in a wireless network. For this we need the multi-channel / multi-radio routing and management. In the HIDENETS architecture these services are covered by separate services. The multi-channel multi-radio approach should be combined with various routing strategies to increase the robustness of the system by finding alternate communication paths in failure situations and enable fast switchover (low recovery times) for applications demanding this.

The routing service is closely connected with the IP Forwarding and Route Resilience service in the respect that the forwarding tables used by IP Forwarding and Route Resilience must implement the routing updates notified by the Routing Module. The IP Forwarding and Route Resilience module is responsible for forwarding of packets and for prioritising between packets when network is congested. The last is important because some applications are very sensitive to communication delay while others are not.

The ad hoc topology control shall compute and maintain a connected topology, which is very important and necessary in a dynamic ad hoc network. The updated connected topology is a prerequisite for the Routing module to in turn compute the routing topology based on this topology information; i.e., a topology change will necessitate a routing update. In some situations, topology changes may have been prognosticated and new routing tables pre-calculated, enabling more resilient networking in highly dynamic scenarios.

The Broadcast/Multicast/Geocast service has an important role in reducing the amount of traffic at the sender side related to broadcasting and multicasting. By optimising the protocols multiple deliveries of the same content can be minimised and broadcast storms, and in general congestion, can be avoided or at least reduced. Since car-to-car scenarios with connection to the Infrastructure domain play an important role in HIDENETS, we should not forget the mobility aspects. These aspects are taken care of by the Infrastructure Mobility Support. The In-stack monitoring and Error Detection and the Performance Monitoring services are responsible for monitoring and error detection, respectively inside the protocol stack and outside the protocol stack.

The Communication Adaptation Manager, QoS Differentiation Manager, Gateway/Network Selection and Profile Manager cooperate in optimising and managing a set of active gateways to the Infrastructure domain, the communication protocols to be used, the load balancing between the gateways, in addition to assuring that QoS requirements and communication level properties can be met and that in case of congestion low priority connections are pre-empted to protect the higher priority connections.

These services are further discussed below, starting from the multi-channel / multi-radio management at layer two and ending with the profile management at lower layer middleware.

6.3.1 Multi-channel / Multi-radio Management

Although multiple frequency channels are available for WLAN communication, most systems use only one frequency channel. Using multiple channels can increase the performance, but e.g., IEEE 802.11a/b/g are based on a single channel architecture. The draft standard on vehicular communication, IEEE 802.11p, does make use of multiple channels, by defining a mandatory common channel for safety applications. In the US, frequencies have already been assigned for vehicular communication using 802.11p. It is expected that the same will happen in Europe.

When using multiple channels, a multi-channel manager is needed on top of the existing WLAN MAC-layer. The multi-channel/multi-radio manager will be at the interface between the WLAN MAC layer and the network layer. The use of multiple channels and/or radios for communication increases network capacity and dependability.

When using multiple channels, using multiple radios is an option to make even more efficient use of multiple channels and to provide a more dependable system. With multiple radios, frequency channels can be used in parallel and for instance concurrent support of safety and non-safety applications can be better supported. This could be done using multiple 802.11p radios or by using 802.11p and 802.11 a/b/g. Besides using

mobile nodes with multiple radios, WLAN access points might be equipped with multiple radios to increase the capacity of the link to the infrastructure.

6.3.2 Multi-channel / Multi-radio Routing

Routing determines the route through the network for communication and plays an important role in load-balancing of the network. But it can also influence the tolerance of network against node failures by using alternative paths. The use of multiple frequency channels/radios for WLAN communication at link layer can be further exploited by using routing protocols that benefit from using multiple channels/radios. Most routing protocols are designed to use a single channel and use a routing metric like the number of hops. Multi-channel routing uses a routing metric that takes the availability and benefits of multiple channels into account. The benefit can be increased by allowing the routing algorithm to initiate channel switching of an interface.

Multi-channel routing increases the network performance, the capacity can be increased and the dependability can be increased. Using multiple channels can also be beneficial to increase the capacity of multi-hop communication as it improves the end-to-end performance. One of the design issues is whether this should be implemented on layer 2 or layer 3 in a mesh network.

The multi-channel/multi-radio routing component will be part of the network layer and will be connected to the multi-channel/multi-radio manager on top of the WLAN MAC.

6.3.3 *Ad hoc* Topology Control

In the infrastructure domain, nodes establish an association and subsequently exchange routing information with all neighbouring nodes. In *ad hoc* networks, however, the number of nodes within physical reach may be very high. In such cases, a node should associate and exchange routing information with only a subset of its neighbours. Selecting which neighbours to associate with (connect to) is part of *Ad hoc* Topology Control.

More precisely, *Ad hoc* Topology Control can be defined as the problem of computing and maintaining a connected topology among the *ad hoc* network nodes. The topology can be optimised with respect to different parameters like for instance energy consumption, total network capacity, or network stability.

Ad hoc Topology Control is particularly hard in networks with a high degree of node mobility, where associations would potentially only last for a short period of time. Ideally, connections should survive for a certain minimum amount of time, and the topology control therefore establishes connections that have a certain probability of surviving more than a minimum time. It will use the connections with the highest probability of letting the packets pass through based on signal strength, packet loss rate, delay and the state of the connected cars (are they going out of reach of each other soon?). The topology control entity needs to know the position plus the speed and direction of the hosting car plus the cars in the vicinity. So this component needs input from the positioning equipment and the cars sensors. Information about the road is needed to predict the movements of the cars which could come from the positioning equipment.

Topology control should ensure that robust *ad hoc* topologies are formed. Examples could be:

- If possible, the removal of a single link or node should not disconnect a topology. It is necessary to come up with a more sophisticated definition of robustness, but for now on we can simply state that the connectivity number of the topology graph should be as high as possible.
- Another approach is to try to form regular topologies (they will never be regular, but when establishing the connections, we can aim at fulfilling certain structural properties).

The major challenge is to do this without a central overview of the full topology. Addressing could also be a part of the topology control – especially for interfacing with the routing manager. The topology control service contributes to the local overview of possible connections and selection of the best connections.

6.3.4 IP Routing

The term routing refers to selecting paths in a computer network along which to send data. Prior to selecting these paths, three main steps are normally fulfilled: Neighbour discovery, information dissemination and path calculations. In *ad hoc* networks, topology control is also needed to keep a connected topology among the potentially mobile nodes. These processes result in is routing tables. Once the neighbouring routers have been discovered, routing protocol messages are used to exchange information about available links and potentially other relevant parameters. Based on this information the routers calculate paths and next hops to forward packets. This might also include backup paths and next hops.

Many different routing protocols exist. In particular, different routing protocols will be used for fixed/wireless networks and *ad hoc* networks, the latter being designed to handle the increased topology dynamics for instance due to node motion.

The challenge is to improve the routing protocols to increase dependability in wireless environments, both in the *ad hoc* domain and accessing the infrastructure domain. For different purposes it could be beneficial to maintain several routing topologies, e.g., for QoS routing and resilience. Maintaining and calculating backup routes should therefore be an important part of this service. It should support the use of efficient recovery schemes based on e.g. fast re-routing, multi-topology routing, multi-path routing algorithms and multi-homing. Such aspects are pursued by the HIDENETS project.

6.3.5 IP Forwarding and Route Resilience

This module handles forwarding of packets on a hop-by-hop basis. Decisions about scheduling, queuing etc. are included. The module will be responsible for policing, marking and remarking of packets according to QoS classification and traffic contract of the individual connections and the valid QoS policy managed by the QoS and differentiation manager. Policing mechanisms are used to enforce that streams do not exceed their assigned resources. Relevant action on non conforming packets are remarking or dropping of packets. Marking is performed by setting an appropriate field in the packet header. This field is read by the forwarding module enabling it to classify the packet for correct queuing and scheduling.

The forwarding decisions are based on the destination of the packets (i.e. the IP address) and the routing table that has been calculated by the routing module. This module is also responsible for rerouting of packets according to pre-planned backup paths when such are available. The new resilient routing schemes developed for the routing module will therefore also have implications on the design of this module.

6.3.6 Broadcast/Multicast/GeoCast

Broadcast/Multicast/GeoCast is mainly used to reduce the amount of traffic at the sender side and in the network due to multiple deliveries of the same content. In addition to this it increases scalability at the sender side by removing the need for the sender to keep information on each receiver, and it provides a simple and efficient mechanism for reaching all hosts that are close to the sender (either close in topology, or in geography – as is the case for GeoCast).

There are many applications and use cases that require one-to-many or many-to-many communication, and that may benefit from using Broadcast/Multicast/GeoCast. Examples are the use cases infotainment, car accident and assisted transportation described in D1.1 [76]. For the infotainment case, an example would be an IPTV application delivering the same content to many receivers. This may benefit from using IP multicast for more efficient and scalable content delivery. In the car accident scenario, there may be a need to transmit the same information (like voice and video describing the situation at the location of the accident) to several instances. Without multicast, the upstream bandwidth demand may become too high at the sender side. For the assisted transportation case, there may be a need to distribute information regarding local road conditions to all the vehicles in the near vicinity, and this would benefit from GeoCast. Reaching hosts that are close in geography can also be useful for the “car accident” case, where it is important to inform nearby vehicles. A potential future use of multicasting to topologically close nodes is multicast-based name resolution — as described in [30]. This is not directly linked to specific use cases or applications, but shows that multicast

can also be used as a service for other communication protocols below the application layer. These are just examples, and one may foresee many other cases where such mechanisms would be useful.

An additional "Broadcasting Manager" may be present to eliminate unnecessary broadcast messages and contribute to the following:

- To increase efficiency in communication (spectrum, energy)
- To reduce the collision probability as unnecessary broadcasts are eliminated.

6.3.7 Infrastructure Mobility Support – Client Part

The mobility client aims at providing connectivity directly to the infrastructure domain for mobile nodes, or groups of mobile nodes. It ensures that sessions are not broken when a node in infrastructure mode changes IP address as a consequence of a handover from one access point to another.

Candidates for mobility support in the IP network include Mobile Internet Protocol - MIP (L3), Network Mobility - NeMo (L3), Stream Control Transmission Protocol - SCTP7 (L4), and Session Initiation Protocol - SIP (L5). Possibly other mobility schemes may be used, like for instance Layer 2 mobility functions. Over the UMTS and GPRS accesses, the mobility functions of these networks are used.

The mobility support mechanism should take into account requirements at other layers such as naming/addressing scheme and session control.

Keeping a mobile node connected to other entities in the infrastructure, even when mobile, is an important aspect for dependable service provisioning and contributes greatly to the quality of the user's experience. Thus, while moving from one access point to another, the ongoing sessions should be sustained and not broken.

This service is relevant to all use cases and applications that involve the infrastructure domain.

This component is intended for mobile nodes, i.e., the cars. Note that depending on the technology for mobility support other entities in the infrastructure domain might be necessary (e.g., a Home Agent with MIP).

This component needs to adapt to the access technology implemented at layer 2 because some legacy access networks are not compatible with specific mobility solutions (e.g., MIP in UMTS+IMS systems).

6.3.8 In-stack Monitoring and Error Detection

This service represents the monitoring and error detection carried out inside the protocol stack and makes the results visible to other services.

In the protocol stack, monitoring and error detection is performed at all layers. This includes, for example, link failure detection, buffer overflow detection at the IP layer, and packet loss detection at the transport layer. This information should be available for other blocks that need knowledge of system performance. This information is relevant for applications and use cases that require extensive information on the state and performance of the node. For fast and correct failure management this information is very important. Taking advantage of monitoring that is performed in the protocol stack reduces the need to monitor these parameters in separate services, and also provides the data that is used for the in-stack decision making (making it possible anticipate the response from the protocol stack).

6.3.9 Performance Monitoring

This service handles performance monitoring carried out outside of the protocol stack. Performance monitoring is the collection of information on important system parameters that may be used to infer the

[7] http://www.sctp.org

performance of a system. Performance monitoring can be used to infer the performance of a link, a path in the network (like a LSP in MPLS, or the path packet travels between two nodes), a sub-graph (like the access network), overall network performance, or even higher layer information like TCP connection set-up time, information on packet reordering and application/service performance.

Performance monitoring is important for detecting failures, and also in choosing the best failover links, network paths, or higher layer components.

Performance monitoring can be carried out using passive or active measurements. With passive measurements existing traffic is monitored to derive the properties of interest. With active measurements, on the other hand, measurement traffic is injected into the system (network), and the response is measured. Whether active or passive measurements are best suited for a specific type of measurement will depend on the case. In general, active measurements increase the load on the system and only measure the response to artificial traffic. This means that active measurements will, to some degree, give biased results. Passive measurements do not necessarily increase the load on the system, and measures the system response to real traffic. The downsides are that all traffic must be looked through at the measurement point, and measurements from different measurement probes may have to be correlated (like when measuring end-to-end delay).

Some relevant parameters may already be available at a networked device. In a router, for example, there are statistics per interface on the number of packets sent and received - accessible via SNMP.

Performance monitoring can in principle be carried out on any protocol layer, but in this context we are primarily addressing monitoring for layers 1-4. Following is a list of some important parameters for protocol layers 1-4:

- Physical layer: Loss of signal, degradation in signal quality and bit errors.
- Link layer: Capacity, frame loss/retries and frame delay.
- Network layer: Throughput, packet loss, packet delay and jitter.
- Transport layer: Packet reordering and connection set-up time.

At higher layers, the result will depend on the components operating at that layer, as well as all the lower layer components involved.

Avoiding performance degradation and possibly lost connection requires good knowledge of system performance and failures. This should not only be based on information from the protocol stack, but also on other parameters of interest that may not easily be obtained by only looking at node-local information. This service implements monitoring of parameters that is not directly available from the protocol stack. The measurements carried out may also involve several nodes – possibly monitoring performance end-to-end.

This information is relevant for applications and use cases that require extensive information on the state and performance of the node, and possibly also involving several nodes. For fast and correct failure management this information is very important.

6.3.10 Communication Adaptation Manager

This component is responsible for adaptation in communication protocols. Examples of adaptations could be:

- Interface handover
- Channel switching (when multiple logical channels are available)
- Route switching
- Service switching (e.g., in some cases, the application may have to switch down the video service in order not to affect the other services, e.g., voice)
- Parameter adaptations.

This component tries to improve the communication protocols in system level performance and user perceived performance. By monitoring certain parameters, it could determine the best for the current

situation, and adjust the behaviour of various protocols. The monitoring information itself is fetched from the Network Context Repository. It also receives input from the Error Detection module on urgent error situations.

6.3.11 QoS and Differentiation Manager

This module has the overall responsibility of the end-to-end QoS management of flows/sessions/connections originating in the node and admission and rejection of flows/sessions/connections transiting through the node; i.e., the overall QoS policy of the node.

QoS policy regimes for end-to-end QoS management may be based on resource reservation and/or differentiation/prioritising of traffic. With resource reservation, the network seeks to guarantee each traffic stream a certain amount of resources (often bandwidth). In such case, Connection Admission Control (CAC) must be performed whenever a new traffic stream requests access to the network to check whether there are sufficient free resources to support it. Resource reservation and CAC may be applicable only for some traffic types, i.e., normally 'long-lived' connections with hard QoS requirements.

Differentiation, on the other hand, is based on different classes of traffic receiving a differentiation treatment with respect to queue management and scheduling. Each packet is marked according to which service class it belongs to in the originating node.

Differentiation techniques like DiffServ are well deployed in today's commercial IP-networks. Resource reservation, on the other hand, is not so much deployed, but standardisation activities are ongoing to enable hard guarantees for applications with strict requirements (e.g., see [31]). To what extent such an approach is relevant for HIDENETS is yet to be seen, but it needs to be discussed at least in relation to the use cases 'infotainment and work with highly mobile terminals' and 'car accident'. It adds some extra complexity since CAC and some QoS signalling (could be based on ETSI RACS [31]) must be implemented and should therefore only be used for applications requiring it and when resources are scarce. Have in mind that the use of differentiation techniques without resource reservation only implies a relative QoS. Indeed, when the network is congested we cannot give any guarantee, even for the high priority flows/sessions, but these should be the last to experience performance degradation.

Resilience may be viewed as orthogonal to other QoS requirements like delay, packet loss, and throughput. Traffic streams may have strict requirements on loss or delay, but loose requirements to resilience. Thus a voice call, which is often prioritized during normal operation because of its strict delay requirements, may be disconnected after a failover to make room for a web session with high requirements to resilience. Resilience differentiation is the task of providing different traffic streams with different resilience support during failures.

For the individual users this module administrates the user profiles. Such information is used to decide on QoS and resilience differentiation classes to be requested when setting up a connection.

For this purpose of communication using the node as transit the QoS Manager implements connection admission control and conflict resolution, and manages resource reservation locally. This is based on the QoS classes of the individual connections and the congestion level. When the network is congested (or above some threshold), the QoS manager may pick low priority connections to be rejected, rerouted or degraded.

Information about available technologies, gateways and networks is used to find the best working path and possible backup path for the respective communication needs in cooperation with the Gateway/network selection service and the routing service. How the responsibility is divided between these two services is open.

The main targeting problem is to find QoS policies for a node that can support QoS requirements of various applications in an *ad hoc* environment as well as in an environment with communication through infrastructure and multiple access choices available. Part of this is to implement different priorities with regard to access of resources and pre-emption in failure situations. This is necessary to protect resources that have priorities with respect to resilience, i.e., higher dependability requirements, when a failure has been detected.

6.3.12 Gateway/Network Selection

The Gateway/network selection component is involved in selecting between different network technologies and/or base stations that are available to a car/terminal at a given point in time. This component may be quite simple, and only provide information on general guidelines for selecting a network. Then the actual choice of network would be left to the routing module. Alternatively, the Gateway/network selection component may take a more active part in the decision about switching to another network technology/base station. It would then need to gather information from other sources like for instance the performance monitoring component or resource information services in the network, and generally have more intelligence and functionality.

The project is investigating whether the Gateway/network selection component should have a small or major role in selecting between different network technologies and/or base stations.

6.3.13 Profile Management

We define a profile as describing the capabilities, properties and type of the node. A profile might describe power supply, bandwidth, media types, mobility pattern etc. The module manages the configuration, updating and dissemination of the profile. The profile might be configured automatically based on context and location. It is an important input to the process of routing and topology control. The profile information must be updated and correct at any time.

Profiling will help finding the best suitable routes to the Internet and such increase the Internet availability for a node residing within an *ad hoc* network. Also we can differentiate the type of traffic that will get access to the Internet. This is particularly important when the resources are scarce. Capability profiling may improve scalability, because it can ensure that the low-capacity components are not used for data forwarding. Traffic requiring a certain quality of service can be routed on devices that have announced sufficient capabilities. High-capacity and stable nodes can form a backbone. In this way we will be able to exploit the resources on high-capacity devices and save the resources on low-capacity devices.

Attributes denoting device capabilities and user preferences should be described in a standardized fashion and for scalability we should try to define "DiffServ[8]-like" property classes instead of exact values.

The service needs input from other services that manage properties like bandwidth, energy, mobility etc. and the service gives output to topology control, IP routing and IP forwarding and route resilience.

[8] Differentiated Services

7. Fault Analysis

A systematic fault analysis from the communication perspective has been initiated. We explain how faults and failure management fit into the layered communication model by developing a fault hierarchy. This hierarchy describes relations between the faults and resilience mechanisms. A discussion is also included to analyse the impact of error propagation from the communication layers up to the MW services and oracles. It is noteworthy that in the anlysis carried out in this section, we mainly investigate the various types of errors that might ocuur at the various communication layers irrespective ofwhether the causes of these errors are accidental or malicious.

7.1 Fault analysis at the communication level

The analysis of possible faults and errors, their propagation at the communication level can be done following the traditional layered communication model. Error contention means and resilience mechanisms can be implemented at each layer. Additionally, cross-layer optimizations can be used also to enhance resilience.

With a layered model, each layer implements mechanisms that handle faults and errors that occur within the layer. An error that is treated and contained within the layer is not observable from outside, so that layer does not fail. Errors that are not fixed inside the layer could propagate as failures till the layer boundaries. A fault affecting a lower level service when observed at the interface offered to the layer above will be referred to as a failure that the layer above has to treat. In many cases, the failure in a lower layer sooner or later reaches a higher layer in the networking stack where it can be fixed, and hidden for the remaining upper layers.

First, we note that different faults may occur in different layers. It is therefore natural to introduce the term “fault hierarchy” as a framework for fault analyses. The fault hierarchy illustrates in which layer(s) different types of faults may occur. It also illustrates the interaction between faults at different layers. For example, buffer overflow at one layer might translate into a packet loss at the layer above; a bit error at one layer might translate into a lost packet at the layer above; or network contention/congestion at one layer might translate into increased packet delay at a higher layer.

Finally, we note that there is a high degree of resilience and a large number of resilience mechanisms already implemented in today’s layered networking model. Most routing protocols of IP (including the exterior routing protocol of IP, BGP, which binds the Internet together) were designed with robustness in mind. If one router is removed, the routing protocols adapt and — if possible — find alternative routes around the failed router within relatively short time.

Figure 11 gives an outline of a fault hierarchy that is used as a framework in the HIDENETS project. The faults are described by the *yellow* (i.e., lightly coloured or blank) boxes, while the built-in resilience mechanisms are illustrated with *orange* (i.e., grey) boxes. (“OK” refers to the case where the resilience mechanism is able to fix the error, while “NOK”, i.e., Not OK, refers to the opposite). The boxes are placed relative to the different layers, i.e., the physical layer, link layer, network layer, transport layer and session/application layer. The main focus here is on faults caused by failures in the layer below, and these are placed on the boundaries between two layers. The boxes are further explained in [32]. Several typical contextual factors that are relevant for the analysis are also identified (see the ellipses shown at the bottom of the diagram).

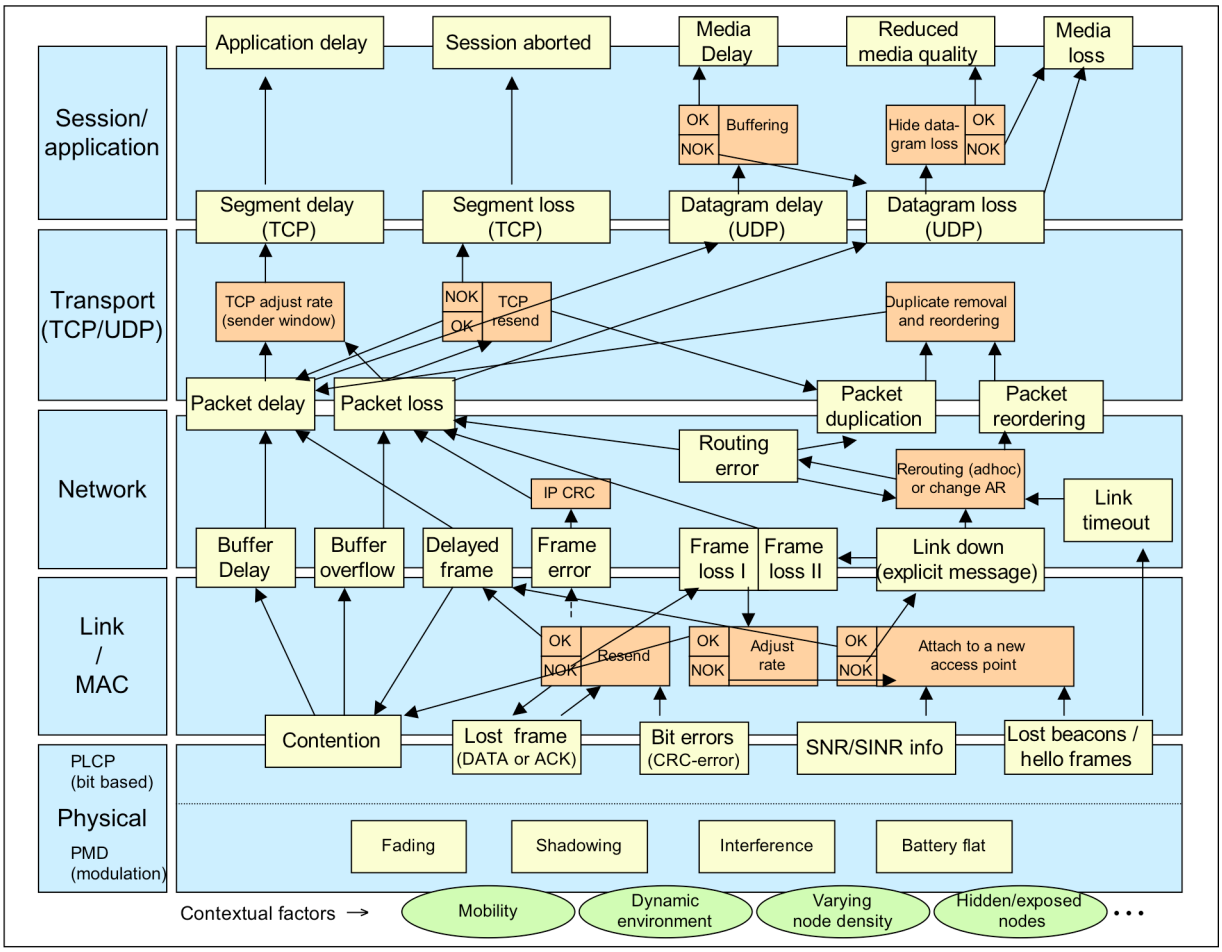


Figure 11: The Fault Hierarchy for fault analysis at communication level

First, we note that it is very difficult to provide a complete picture of the fault hierarchy with every possible fault described in detail. It is also difficult to describe every interaction between faults and between the mechanisms in a layer used to fix faults. As seen in Figure 11, the description is already quite complex, and describing the fault hierarchy in more detail than what is done in the figure might not serve its purpose well.

The fact that a model will not be able to cover all possible faults is easy to realize by mentioning some extreme cases of faults. For example, the hardware might be smashed in a car accident or hit by lightening. Likewise, there might be hackers that purposely try to launch DoS attacks, e.g., by transmitting bogus 802.11 management frames that dissociate all nodes attached to an access point / base station. Finally, nodes might be mis-configured in all sorts of way, leading to nearly all sorts of possible errors. The main point here is that the potential types of faults are quite large, and too large to be included in the framework presented in this section. The first simplification is therefore to focus on some of the most plausible causes of failures.

Another simplification of our model is that it might conceal the fact that faults might have different degrees of severity. For example, the “packet delay” fault box in Figure 11 says nothing about the extent of the delay. The packet delay might be so low that it does not affect the functionality of the higher layers, and the effects of this delay might for example be removed by a streaming buffer at the application layer (Figure 11). On the other hand, the packet might be delayed to such an extent that it is mistakenly perceived as a dropped packet by the higher layers. For example, if the delay is above a certain threshold, the TCP might react by reducing its sender window and consequently the senders mean transmission rate. The same argument goes with packet loss. In some cases only a single packet is lost, which might for example not influence the quality of a voice conversation considerably. In other cases, there might be a large number of consecutive packets that are lost, in which case the consequences are much more serious.

For the further analysis we will restrict our study to the type of failures that we feel are most relevant for the communication level to handle given that the purpose is to increase the dependability of the system. Such failures are link and node failures, and to some extent also general performance degradation (in transfer) like

high packet delay and packet loss. In general the methods introduced may also help increasing the throughput. Faults at the physical layer are out of scope.

For node and link failure we in general need to detect the failure before actions can be taken. Performance monitoring and failure detection are therefore an important part of this.

Mobility may cause nodes to be out of reach. This may for our purpose be seen as link/node failure, but if foreseen by the mobility pattern, actions may be taken in advance. In other cases performance degradation, e.g. packet loss / contention, may trigger similar actions as if the link failed. It is important to define threshold values for such actions to take place.

In a network with alternative routes the problem may be handled by protection paths or rerouting of traffic, depending on requirements of the particular application and the service level agreement of the individual users. Different methods will be addressed.

Figure 11 illustrates the fact that some of the faults are forwarded to the layer above without being fixed there, e.g., lost frame at the link layer might result in a packet loss error at the networking layer. Other examples are also found in the figure. For example, contention (/congestion) at the link layer, might translate into buffer delay or buffer overflow at the lower part of the networking layer (e.g., in the device driver or in the socket). These are perceived by the transport layer as packet delay and packet loss, respectively, and as datagram delay and datagram loss by the application or session above if UDP is being used. Finally, a lost datagram might for example result in a loss of VoIP signal (referred to as “media loss” in Figure 11).

Furthermore, the figure also shows that if a TCP segment is delayed, it might delay the execution of the application above, while a TCP segment loss might eventually result in a TCP reset in which the TCP session is aborted. Finally, loss of a multicast hello packet (in *ad hoc* mode) or of a beacon frame (in infrastructure mode), might result in link timeout at the networking layer, in which the networking layer assumes that the link is down.

The grey box labelled “Rerouting or change of AR” addresses the cases where rerouting or change to a new access point is necessary due to a link failure. Also in the case that the node is in *ad hoc* mode, a “Link down” notification might lead to changes in the routing protocol. This change (or “reroute”) is also carried out in the grey “Rerouting or change AR” box. Alternatively, the reroute might also be triggered by the loss of a number of hello frames. In order to capture this type of fault, a link timeout is implemented in the routing protocol at the network layer, as illustrated in the figure.

If the node is in *ad hoc* mode and is a part of an *ad hoc* network, the functionality carried out in the “Rerouting or change of AR” box might be considerably more complex than the functionality carried out if the node is in infrastructure mode. Thus, for a node that is in *ad hoc* mode the “Rerouting or change of AR” box could probably easily be divided into a large number of additional blank fault-boxes and additional grey resilience-boxes that try to fix these additional errors within the context of the routing protocol. Thus, a separate fault hierarchy could probably be constructed for the routing protocol separately. However, this is out of the scope of this document.

7.2 Implication of Communication Fault Hierarchy on the MW Oracles

In the previous subsection, we carried out an analysis of faults and errors propagation and containment at the communication level. However, this analysis focused on the “normal” or payload communication services, which are used by middleware services and applications located in the payload part of the system, be it at the operating system level or in user space (see Figure 5). We now address the fault model and error propagation and containment from the point of view of the oracles included in the resilience kernel, according to the hybrid system perspective.

It should be noted that this discussion is only relevant when considering the existence of MW oracles with a distributed nature (see Section 4.3.1). In this case, the adoption of a hybrid system perspective must also be extended to the communication subsystem. A distributed MW oracle, like, for instance, the timely timing failure detection MW oracle, typically requires the timely execution of a distributed protocol, which must be done through a communication system with better (synchrony) properties than the payload communication system. For other MW oracles, which provide strictly local services or which do not need the execution of

distributed protocols with timeliness requirements, there is no need for a separate communication system and so the following discussion is not relevant.

On the other hand, the need to ensure communication channels with better properties when distributed oracles are to be implemented turns out to be an easier task than it would be if those better properties had to be ensured for general purpose communication channels. This is so because, by construction, these better communication channels are only used for specific purposes, for transmitting typically small and a priori known pieces of data, which are thus comparatively easier to schedule in a predictable way.

Quite clearly, it is not always possible to provide strict guarantees for the timeliness or trustworthiness of the communication channels. It is a matter of coverage of assumptions. We argue that by using the proposed hybrid architecture it may be easier to secure some better properties for the channels serving the oracles with higher coverage than it would be possible to achieve for general purpose channels. We also note that this is the case, independently of the techniques or mechanisms that are used to achieve improvements of the communication channel properties. Some specific techniques might also be considered that exploit the fact that communication can be better controlled (by knowing how the oracles exchange information) to achieve the improved properties.

The diagram of the fault hierarchy for the communication system serving the MW oracles (see Figure 12) is based on the fault diagram represented in Figure 11. In fact, by design this communication system should be simpler than a general purpose communication system, as required to ensure increased resilience and synchrony, for instance. Nevertheless, we use the diagram presented in the previous section in order to better illustrate the differences in the fault analysis and what could be done to secure the required fault model for the communication between MW oracles.

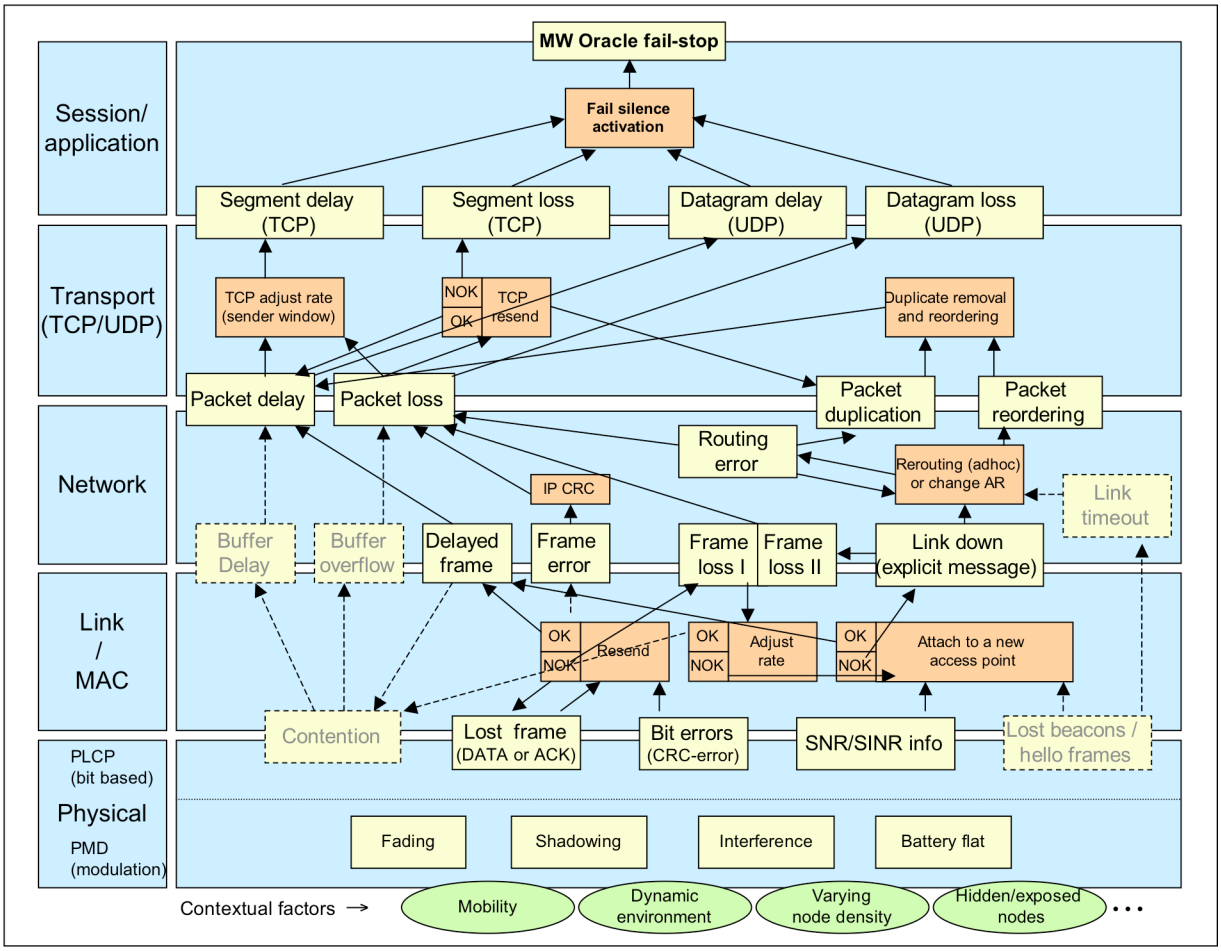


Figure 12: Fault analysis for the communication between MW oracles.

In general, when compared to the fault model assumed in the payload part of the system, the fault model assumed for the construction of MW oracles must be more strict. This stronger model is necessary in order to be able to provide improved services and secure better properties to be presented by the MW oracles. This is true not only in general terms, but in particular concerning fault assumptions for the communication.

For the sake of illustration, the diagram for general-purpose communication has therefore been modified in two different ways, assuming the following assumptions are enforced by the design:

- First, some faults have been considered as less probable, and therefore partly removed from the assumed ones, which can be supported assuming that fault prevention techniques can be used.
- Second, the behaviour of the communication subsystem has been transformed into a synchronous system with crash behaviour, by transforming every timing or omission fault into a crash failure, which we assume to be possible by using adequate fail-stop mechanisms.

To be more specific, for this example, the faults whose likelihood of occurrence can be virtually discarded (with a realistic assumption coverage) are:

- contention, and
- lost beacons / hello frames.

The corresponding boxes are identified by means of dashed lines, as well as the related links and directly linked downstream boxes.

For what concerns contention, this could be accomplished by the use of *private channels*. Due to this, the following situations are not expected to occur: buffer delay and buffer overflow. A simple technique to avoid the buffer overflow is simply to define buffers with a size that makes it impossible to fill them completely. This of course requires the knowledge or the definition of upper bounds for some transmission parameters, which might be done more easily in the scope of the MW oracles (for instance with the implementation of admission control techniques).

Concerning lost beacons/hello frames, other techniques will have to be used. One possibility is increasing the redundancy of these beacons, with a higher number of transmissions, as high as possible to not increase the probability of collisions. Another solution might come from the use of alternative techniques and standards, such as the 802.11p standard, which defines the possibility of using a more restricted control channel that could be assigned to the MW oracles and, in that sense, would probably allow reducing the possibility of collisions.

With respect to the transformation of the timing and omission faults into crash fault, the affected services in the diagram are those on the top layer and include:

- Segment/Datagram delay, and
- Segment/Datagram loss.

These faults provoke a deadline miss so the MW oracle must ensure that this is not perceived at the application level. Otherwise, the assumed properties for the oracle would not be satisfied. Therefore the MW oracle transforms these failures into crash failures, and the node where the MW oracle resides goes into a fail-silence state. This transformation is highlighted in Figure 13, in which it is possible to observe the referred transformation of timing and omission faults into crash faults. Note that we distinguish between detected and non-detected timing/omission faults, because it might not be possible to implement a detector with the perfect properties, one that would perfectly detect and distinguish all the kinds of faults. Any implementation will necessarily have to be based on some assumptions, which might fail.

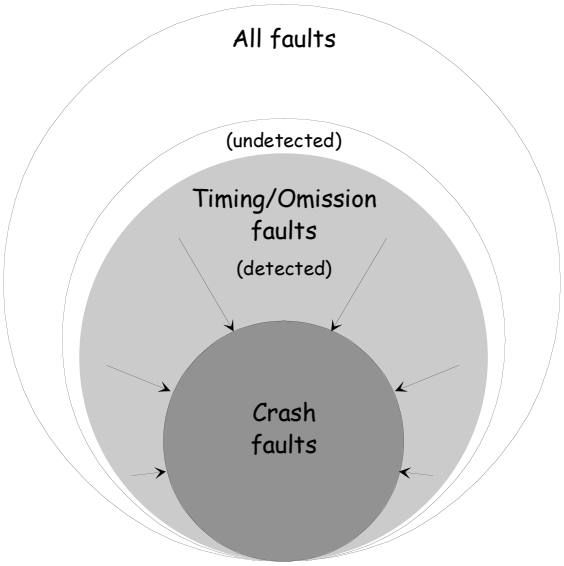


Figure 13: Fault classes and their treatment for the communication in distributed MW oracles.

8. Quantitative Evaluation

The HIDENETS activities related to quantitative evaluation are aimed at developing methodologies and techniques that can be used to quantify and analyse dependability and performance characteristics of HIDENETS design solutions considering system, middleware and communication level properties. In particular, these methodologies will be used to analyse the high-dependability solutions developed in the course of the project. Following a holistic approach, several evaluation methods can be used for quantitative assessment of dependability and QoS indicators, including analytical stochastic modelling, simulation and experimental measurements.

In this section, we focus on the following fundamental topics:

- Identification and description of the most challenging HIDENETS characteristics with respect to the quantitative evaluation activity (Section 8.1). The "complexity" of the HIDENETS system is detailed in terms of more concrete characteristics, like heterogeneity, dynamicity, variety of threats and mobility aspects.
- Identification and discussion of the challenges concerning the different evaluation approaches (Section 8.2). Such hard problems need to be addressed inside each evaluation technique, but also by exploiting the interactions among different evaluation approaches.
- Discussion about the opportunity to adopt a holistic approach that exploits the interactions among different evaluation techniques (Section 8.3). The HIDENETS holistic approach can be used to provide the proper response to the specificities and difficulties to quantitative evaluation of the HIDENETS environment. The rationale is to exploit the collaboration among different evaluation techniques in order to obtain a more accurate evaluation of the final QoS measures. So far we did not mention and do not intend to provide a physical integration of the techniques, but we just highlight the importance of exploiting the logical interdependencies among different evaluation methods.
- References to evaluation methods developed and used in HIDENETS, capable to cope with the complexity problems previously outlined (Section 8.4).

In the following, according to D4.1.1 [51], we outline each of these topics.

8.1 Challenging HIDENETS Characteristics

The assessment of the dependability-related attributes for HIDENETS systems is a very challenging topic due to its characteristics, which are common to many other contemporary application area having a high interconnectivity between different infrastructures with seamless interactions. In the following, we list the main system characteristics that can affect the quantitative evaluation activity:

- **Heterogeneity:** the networking scenario includes wireless *ad hoc* networks, wireless infrastructure-based networks, and also wired networks. The characteristics of these network domains are quite different. For example, the wireless *ad hoc* domain shows high dynamicity, caused by changing topologies and changing link properties (varying propagation conditions), while the fixed network has only low dynamicity (mainly due to network traffic fluctuations and congestion). The wireless *ad hoc* domain in particular poses challenges to dependability solutions, since an increase of traffic by redundant packets may lead to increased self-interference, medium access blocking (including hidden terminal problems), and congestion. Furthermore, heterogeneity is also caused by different device types, which may differ in capability (processing power, available battery energy), available (wireless) communication interfaces, as well as available middleware functions. The heterogeneity could force the modeller to consider different modelling and solution techniques, each one specifically tailored to capture the behaviour of a part of the overall system. In this context several challenging issues arise, like the definition of a proper mapping between subsystems and modelling techniques, as well as the identification of the possible interactions between the different techniques, both at model-level and at solution-level.

- **Use of OTS:** Off-the-shelf components (OTS) are inherently unreliable, therefore end-to-end system-level resilience solutions addressing both accidental and malicious faults must be developed. The characterization of OTS failure modes and the analysis of how these failure modes might affect the dependability of the services interacting with the OTS is a critical step. Usually, OTS exhibit little information on their development process, on their architecture and on their actual failure behaviour. Evaluation techniques based on controlled experiments and robustness testing are very much suitable to reveal and understand the faulty behaviour of these components and then design appropriate mechanisms to tolerate these undesired effects.
- **Large number of components and scenarios:** The set of interacting components involved in a single use-case can be very large, and their number immediately increases if the scale of the system increases as well. The number of components to be considered in the quantitative evaluation process will depend on the level of detail needed to evaluate the quantitative measures under study. Besides the number of components, the complexity of the evaluation can also result from the existence of a large number of failure modes and recovery and maintenance scenarios to be taken into account.
- **Dynamicity:** The considered systems are dynamic in terms of topology, connectivity, and channel conditions. Mobility contributes to most of the dynamics. In *ad hoc* networks, this mobility results in highly dynamic network topologies making static routing not applicable. Dynamic routing itself is a challenging task, even without requirements on dependability. Due to the dynamics of the system, inconsistency problems with respect to different local views of the topology may occur, which can cause route breakage or routing loops in the *ad hoc* domain. Moreover, high mobility is expected to make the modelling of the radio propagation channel harder. Finally, mobility at high speed leads to very short time periods of connectivity to the same access point or other cellular-type infrastructure networks. Hence, frequent handovers can be the result that also imply that fast approaches for establishing link-layer connectivity are required. Providing highly dependable network solutions for such a dynamic scenario is so challenging that the resulting systems tend to be complex and to incorporate a lot of adaptive and fault-tolerant solutions. Since mobility contributes in major parts to the dynamicity, the modelling of mobility becomes extremely important.
- **Strong interdependencies between different system parts:** Interdependencies can result from functional or structural interactions between system components related to the system architecture, or from fault tolerance and maintenance strategies, leading to stochastic dependencies that need to be captured by the models and evaluation techniques used to assess dependability. Communication and network performance also introduce additional challenges to the modelling and quantitative evaluation as the three major influencing factors, network traffic models, dynamic topology models, and link-level connectivity models are heavily inter-dependent in *ad hoc* networks. Furthermore, adaptive protocols, such as the Transmission Control Protocol (TCP) and the communication layer resilience solutions introduce a feedback from the resulting network performance and fault-model to the traffic model and the routing. Such types of systems frequently require *fixpoint analysis* (if stable) or they require *transient analysis* (see [33, 34] for examples) due to potential instabilities.
- **Variety of threats:** Historically, accidental faults have been the main source of failures considered for computer and communication systems. Such faults may result from physical phenomena, design misconceptions or human mistakes. In the last decade, malicious faults have also received an increasing attention from the developers and the users of computer-based systems because of the exponential increase of reported vulnerabilities and malicious threats, and the openness of systems and their interconnection to the Internet. Mobile *ad hoc* systems in general, and HIDENETS systems in particular, will have to cope with such a variety of threats and address both accidental and malicious faults (attacks and intrusions). From the evaluation point of view, quantitative evaluation techniques have been mainly used to evaluate the impact of accidental faults on systems dependability. On the other hand, the evaluation of security has been mainly based on qualitative evaluation criteria. Only recently have attempts been made to quantify measures associated with system security, but no general system-level methodology currently exists that can quantify the security provided by a particular system-level security approach [37].
- **Correlated events and need for transient analysis:** Several applications, e.g., the car accident use-case in HIDENETS, include communication and middleware services that are triggered locally (in time and

space) by a specific event**.** Such scenario leads to highly correlated network traffic models (independence assumptions between sources cannot be applied), and in addition creates the need for transient analysis, since the services and communication are triggered by non-repetitive events (here the accident).

8.2 Challenges Related to Each Evaluation Technique

The system characteristics previously described lead to several challenges that need to be addressed in performing a quantitative evaluation. Some of these challenges are discussed in the following considering three main evaluation techniques: 1) analytical models, 2) simulation and 3) experimental evaluation.

8.2.1 Challenges in Analytical Models

In this section we discuss the main challenges concerning the analytical modelling activity, namely: 1) modelling complexity, 2) state-space explosion, 3) stiffness, and parameters estimation.

- **Modelling complexity:** The overall description of critical complex systems can be a very tedious and difficult task. The modelling complexity is generally related to the level of detail considered for describing the main phenomena and behaviours captured by the models and to the quantitative measures that need to be evaluated. The complexity could also be exacerbated when multiple interactions and interdependencies exist among the model components. This problem is also related to the system scalability. A HIDENETS use case can involve several applications, different network domains and can affect a large number of users; therefore, system scalability must be addressed considering efficient ways to build models representing very complex scenarios.
- **State-space explosion**: The state-space methods construct a structure (the state-space) that consists of all states that a system can reach, and all transitions that the system can make between those states. The problem is that the size of a state space of a system tends to grow exponentially with the number of its processes and variables. In particular, the largeness of the state-space depends on the number of local states a process has, on the number of values a variable may store and the extent to which the states of components are determined by the states of other components.
  The state-space explosion problem could arise in the HIDENETS context due to its large number of interacting components. To master this problem, a modelling methodology is needed so that only the relevant aspects can be detailed thus allowing numerical results to be effectively computable. In addition, simplifying hypotheses are very often necessary to keep the model manageable. Of course, the choice of such hypotheses is critical, to avoid obtaining a system model so far from the real behaviour that evaluation results become useless in practice.
- **Stiffness:** A problem is stiff if the numerical solution has its step size limited by the stability of the numerical technique. Therefore, a symptom of the potential presence of stiffness is the existence of components that change much faster than others, although it is important to state that the presence of this symptom is not necessarily an indication of stiffness.
  This problem could arise in the HIDENETS context for example in presence of components having very different "physical" characteristics. Actually, the complexity of the system includes both "high-level" events that occur with a very low rate (e.g., a hardware fault in a mobile network device) and "low-level" events that occur with a very high rate (e.g., the data packet transmission over the GPRS network).
- **Parameters' estimation:** Another problem is the determination of the values to assign to the parameters required by the models. Actually these values can be difficult to obtain (usually by way of experimental tests), and they cannot be provided during the preliminary design phases of the system. 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.
  This problem could arise in the HIDENETS context considering the complexity of the model that should describe the system behaviour, thus including a large set of parameters that need to be properly instantiated. Sensitivity analysis is generally needed to evaluate how the variations of the system and

modelling parameters affect the evaluated quantitative measures and to identify those parameters which have the most impact on these measures and for which particular care is needed to obtain accurate estimations. Indeed, it is well known that a small variation of the coverage factors characterizing the efficiency of fault tolerance mechanisms might have a significant impact on the evaluated quantitative dependability measures.

8.2.2 Challenges in Simulations

Simulations are used in the context of HIDENETS for multiple purposes:

1. To obtain quantitative performance results for scenarios which are not suitable for analytical approaches (e.g., due to rich functionality, high complexity).
2. To validate analytical models and experimental measurements and to verify how simplifying assumptions in analytic/experimental models impact the analysis results.
3. To obtain preliminary insights into system behaviour in order to identify performance/dependability relevant system parts for the analytical modelling.

In the following, we discuss three critical issues that need to be carefully analysed to fulfil the objectives listed above: 1) development of simulation models with adequate system abstractions, 2) scalability of the simulation scenarios, and 3) output analysis and rare event problems.

- **Development of simulation models with adequate system abstractions:** Although simulation models can be rich in features, runtime constraints typically require to determine the adequate level of detail of the simulation model and abstract the remaining parts of the system. For instance, modulation and coding as well as detailed symbol transmission over wireless channels are typically abstracted by adequate packet error probabilities in network-level simulations. Since the HIDENETS network architecture and service set are rather large and complex, adequate choices on the granularity of the simulation models need to be made. Basically, this also requires supporting models (analytical or decoupled simulations) in order to correctly represent the abstracted parts, e.g., packet error models as input to the simulation, or use of dynamically changing connectivity graphs instead of geographic node mobility as input.
  As simulation models become more and more complex, the execution of such complex simulation models presents a resource-expensive task. This is the case if the simulation includes a large number of nodes. Furthermore, the introduction of new traffic models with long-range dependent properties even increases the number of samples that have to be evaluated for statistically significant results. Techniques for (semi-) automated parallel simulation can alleviate the resource problem.
- **Scalability of the simulation scenarios.** Scalability is related to adequate system abstractions as described above in the sense that the complexity of particular components puts restrictions on attainable simulation scenarios and also requirements to the scalability of the simulation tool and implementation. In HIDENETS, there is strong cross-influence between different system parts (Section 8.1), and hence simplifications and focus on one system part cannot always be a solution for providing scalable simulation scenarios. The solutions investigated in HIDENETS should also be applicable to very complex scenarios where a wide variety of terminals, technologies, communication needs, traffic types, priorities etc, might influence each other's quality and availability. Designing scalable simulation scenarios that still cover these aspects is an important challenge for the simulation studies in HIDENETS.
- **Output analysis and rare event problems.** The rare events problem occurs when there are events that occur at very different time scales, so the computational time needed to obtain a statistically significant solution becomes unacceptable. In many situations, the properties of interest depend critically on the occurrence of a rare event, that is, on observing a subset of system states which appear with a very low probability. In this case the standard simulation techniques become inefficient, since the low probability of the interesting states makes it very improbable to observe them in a random sample of the evolution of the system. As a consequence, the statistical significance in the estimation of the target measures becomes very poor.
  For any simulation analysis, simulation results have to be inspected carefully and standard methods to

evaluate statistical significance of the simulation output have to be applied. It will depend on the assumptions, more specifically on fault models, whether actual rare event problems occur in the simulation or not. Moreover, simulation experiments for systems with strong (long-range) correlation properties suffer specific problems, e.g., steady-state behaviour may not be observable or the simulation results can show large variance. Such systems may pose technical challenges on the conduct of the simulation experiments, e.g., on the quality of the used random number generator.

8.2.3 Challenges in Experimental Evaluations

The main challenges concerning the experimental evaluation activity that we have identified concern: cost, intrusiveness, semantic gap and reproducibility.

- Concerning *cost*, it is worth noting that experimental evaluation is usually a quite expensive activity since it requires having access to real systems or prototypes that need to be evaluated. Therefore, it is clear that the identification of the interesting situations in which to "test" the proof-of-concept laboratory set up is a very important activity, since the experimental evaluation approach cannot be easily tuned during the course of the project.
- Experimental evaluation activity can be seen as performed by a set of monitoring probes that are "plugged" into the target system (a real system or a prototype). In the case of controlled experiments aimed at active measurements, e.g., fault injection experiments or some cases of performance monitoring, active probes may be used to interfere with the target system. Accordingly, *intrusiveness* relates to both monitoring and active probes (e.g., faultload or workload "injection"). It is clear that such probes should be non invasive, in the sense that they could not affect the behaviour of the target system. Rather, the target system should evolve as in absence of the probing devices.
- Frequently there is a *semantic gap* between the data collected during experimental evaluation and the relevant measures interesting for the user or evaluator. Due to the physical constraints of the available interfaces for non-intrusive observation, monitoring systems often collect low-level data (e.g., interactions at the physical communication layer, memory accesses of a processor), that shall be 'translated' to higher level quality of service characteristics (e.g., expected service delay) or dependability measures. The model(s) of the system effectively support this translation.
- Conducting field measurement or controlled experiments with real wireless interfaces may make it difficult to ensure the *reproducibility* of the results of the experiments. Even if the geographic mobility of the nodes is very carefully described and followed during the course of the experiment, the results of the experiments may not be the same since the propagation conditions are strongly influenced by changing environment conditions, e.g., interference, moving objects etc.. Hence, even if geographic mobility is reproduced, the resulting ad hoc network topologies may be different unless the propagation and interference conditions are also reproducible, e.g., via shielded labs etc. The latter may be impractical for large-scale experiments; hence reproducibility for communication in dynamically changing topologies can in some scenarios more easily be achieved by emulation of link-layer connectivity. It is also important to note that, for evaluation purpose, reproducibility requirements relate to the ability to obtain statistically equivalent results as explicitly recognized for example in the experiments carried out for dependability benchmarking [38, 39].

8.3 The HIDENETS Methodological Approach

Analytic, simulation and experimental approaches show different characteristics, which determine the suitability of the method for the analysis of a specific system aspect. The most appropriate method for quantitative assessment depends upon the complexity of the system, its development stage, the specific aspects to be studied, the attributes to be evaluated, the accuracy required, and the resources available for the study. Analytical and simulative approaches are generally cheap for manufacturers and have proven to be useful and versatile in all the phases of the system life cycle. They are typically based on a parametric model of the analyzed system and on a set of assumptions concerning the behaviour of the system and/or of the system environment. Analytical approaches are highly efficient, but the accuracy of the obtained results is

strongly dependent upon the accuracy of the values assigned to the model parameters and on how realistic the system model assumptions are. The simulative approach is one of the most commonly used approaches for quantitative evaluation in practice, especially for highly complex systems, for which analytical solutions are generally precluded; however, it tends to be generally more expensive in terms of computational time, especially in case of rare event problem. The simulative approach is superior to analytical modelling in capturing relevant phenomena through more realistic representations (e.g., to overcome the exponential distribution for event occurrences, which is usually implied by the analytical solution). Experimental measurement is an attractive option for quantitative assessment of an existing system or prototype. This method allows for monitoring the real execution of a system to obtain highly accurate measurements of the metrics of interest. However, it may turn out to be quite expensive, e.g., when the interest is in very rare events, and the obtained results are often difficult to generalize. In this case, appropriate techniques based on active measurements and controlled experiments (e.g., fault injection) can be adopted.

The largeness and complexity of dependability-critical systems, together with the necessity of continuous verification and validation activities during all the design and development stages in order to promptly identify deviations from the requirements and critical bottleneck points, call for a composite V&V (verification and validation) framework, where the synergies and complementarities among several evaluation methods can be fruitfully exploited. This is the rationale of the *holistic approach* proposed for the analysis of HIDENETS systems: several (maybe complementary) evaluation techniques cooperate to reach a common objective, that is the end-to-end quantitative assessment of dependability and QoS indicators, both as system's characteristics and as perceived QoS of the services offered to the users.

The application of the holistic approach allows defining a "common strategy" using different evaluation techniques, applied to the different components and subsystems, thus exploiting their potential interactions. In the evaluation of systems like HIDENETS, a single technique is not capable of tackling the whole problem. The idea underlying the holistic approach follows a "divide and conquer" philosophy (see Figure 14): the original problem is decomposed into simpler sub-problems that can be solved using appropriate evaluation techniques. Then the solution of the original problem is obtained from the partial solutions of the sub-problems, exploiting their interactions.

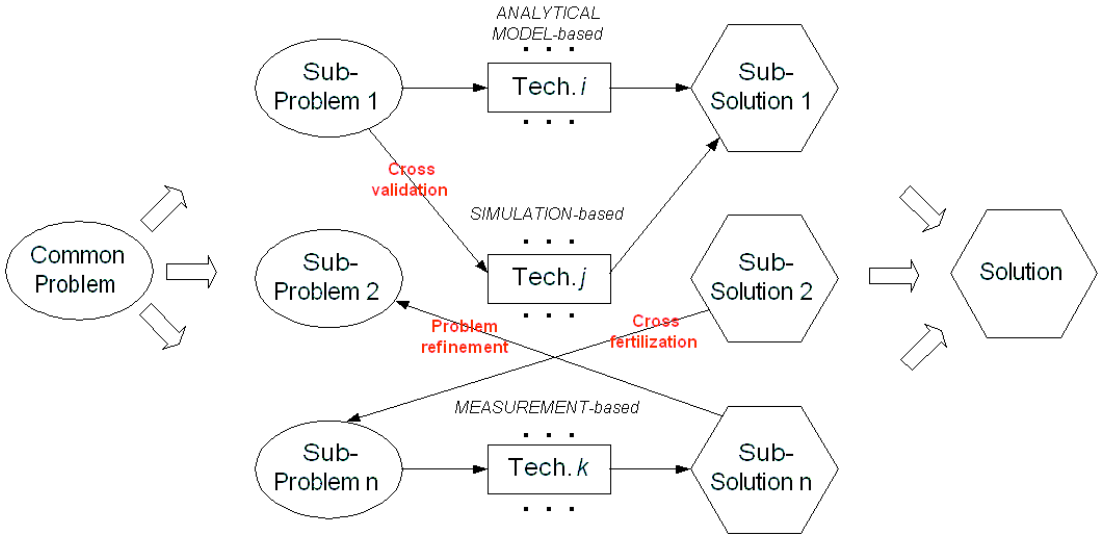


Figure 14: Example of possible interactions among the approaches

Examples of possible interactions are (see Figure 14):

- Comparison of results for a certain indicator obtained through the application of two alternative methods allows *cross-validation* of both.
- Feeding a system model with parameter values derived through experimental measurement is a central example of *cross-fertilization* among different methods.
- The analysis of results gives some additional knowledge that leads to a *problem refinement* (e.g., the architecture of a component changes since it is recognized to be a system bottleneck).

It is clear that the system decomposition of Figure 14 is not unique, as we can identify different system decompositions corresponding to different levels of abstraction. The higher is the level of detail required to capture the system behaviour, the higher is the complexity of the system to be modelled and solved. Therefore the choice of a particular system decomposition is of primary importance, and it is always a trade-off between correctness of representation of the real system behaviour (with respect to the measures of interest) and capability to solve the corresponding models.

In the following subsection we depict a type of system decomposition, the abstraction-based system decomposition, which statically focuses on the various levels of abstractions that can be used to represent a system. Another type of conceptual-level system decomposition focusing on the dynamicity of the system behaviour will be introduced in Section 8.4.1.1 (the "interaction-based" system decomposition). These two types of decomposition approaches could be applied in isolation or in cooperation, depending on the HIDENETS scenario to be analyzed, as it will be sketched in Section 8.4.1. In parallel to the top-down decomposition approach, a complementary bottom-up modelling activity is performed concerning the architecture and communication system aspects (see Section 8.3.2).

8.3.1 Abstraction-based System Decomposition

The overall system can be analyzed at different levels of abstraction: Each level captures a specific aspect of the overall system behaviour and it "communicates" with the other levels through some well-specified interfaces. Such interfaces mainly define the input and the output that, respectively, they need and provide to the other abstraction levels. In particular, we could identify the following abstraction levels:

- **Communication level**. It captures the communication aspects of the system that can affect the application level. It addresses the link layer (considering several types of networks like WLANs, UMTS and GPRS), the network layer (IP-based) and the transport layer (considering several types of protocols like TCP and UDP).
  - The main expected inputs are the traffic and mobility patterns and a set of assumptions introduced to hide some too low-level system details.
  - The expected outputs are low-level measures like: message delay, probability of lost message, probability of that a message is incorrectly emitted or is omitted. Such measures could be mean values or complete distributions, and could be used at the application-level.
- **Architecture level**. This is the part of the system capturing the behaviour of the main hardware and software components (resources) that can affect the application-level measures. It describes how the application components and services of the application level are implemented on these resources. This layer also includes the middleware that abstracts some details of the underlying layers for the application running on top.
  - The expected inputs are some low level parameters concerning hardware, software or basic services, such as failure rate, error latency, repair rate, error propagation probability.
  - The expected outputs are some medium-level dependability-related attributes, like availability and reliability of some services used at the application-level.
- **Application level**. This level describes the system behaviour from the application point of view. The applications differ for their technical properties, for their mechanisms, for their interfaces, and they can impose different communication and middleware level requirements. This level identifies for each function and application component the set of services offered by the architecture for its implementation. Each function and application component may depend on several services and the services may depend on each other.
  - The expected inputs are the outputs produced by the architecture and communication levels.
  - The expected outputs are high-level system-oriented measures, like availability, reliability, safety, performance, QoS and so on. Some of these measures could be provided as input to the user level.

- **User level**. This level describes the users' profiles, that is how the users interact with the application and how their requests are mapped to the different components of the architecture. Each user scenario can be characterized by the set of functions and application components invoked and the frequency of activation of each of them. A user level is needed to account for different classes of users having different behaviours and different requirements. Mobility, available applications and application utilization are just some examples of users' characteristics that can differentiate a user's class from another.
  - The expected inputs are the outputs produced by the application level.
  - The expected outputs are high-level QoS attributes related to the user's perspective.

8.3.2 Complementary Bottom-Up Modelling

In parallel to the top-down decomposition approach as described above, a selected number of model implementations are developed from the very beginning in a bottom-up manner. The expected benefits from conducting such a bottom-up modelling approach in parallel to the top-down decomposition are the following:

- The quantitative results from sub-models for dedicated services (both on communication level and middleware level) in a stand-alone manner provide useful feedback for the development of the corresponding service.
- The parallel development of top-down modelling approach and bottom-up sub-model implementation will lead to concrete experiences about inter-working problems of the different model implementations in an end-to-end model and will thereby increase the chances for a successful, implementable, holistic model.

The bottom-up sub-model implementations belong to the architecture level as well as to communication level. They utilize analytical approaches, simulation models, or selected experimental approaches.

8.4 Individual Approaches Composing the HIDENETS Framework

This section describes a few methodologies and techniques that can be used for the evaluation of the individual parts of the HIDENETS systems, and for solving specific sub-problems in which the end-to-end evaluation of HIDENETS can be decomposed. In particular, the low level details of the system (at the architecture and communication layers) can be captured by experimental evaluation techniques and simulation approaches, while more high-level end-to-end scenarios can be assessed by model-based approaches, both analytical and simulative.

8.4.1 Analytical Methodologies

The complexity of the HIDENETS system (considering the presence of a large number of components, the dynamicity of the network and the mobility of the users) raises special problems during the analytical modelling and evaluation activity. State-based models suffer from the problem of *state-space explosion*, i.e., the large number of states may impede the solution of the models; therefore largeness avoidance or largeness tolerance techniques have to be applied.

In the HIDENETS evaluation framework system decomposition techniques are applied along two "dimensions":

- The system lifetime is decomposed into a sequence of *phases*, i.e., time periods which are characterised by fixed dependency relations among entities (these entities come from the functional decomposition of the system). The resulting phased-interaction system is introduced in the forthcoming Section 8.4.1.1.
- Considering a given phase of the system, a *hierarchical construction* and solution of dependability evaluation models is carried out that distinguishes user, application, architecture and communication levels (see Section 8.3.1). The approach is described in Section 8.4.1.2.

The construction of dependability models at the different levels of abstraction can benefit from the (semi)formal description of the entities and relations belonging to that abstraction level. In Section 8.4.1.3, we propose a tool-supported approach to construct dependability models on the basis of enhanced UML diagrams.

8.4.1.1 A decomposition approach to evaluate high-level performability measures of HIDENETS-like systems

Decomposition approaches seem to have great potential in managing system complexity. Here we propose a decomposition approach based on the interactions among system subcomponents [41]. Elaborating on [49], we first analyze a system from a functional (or logical, conceptual) point of view (functional decomposition). The overall system is decomposed in a set of interacting sub-systems, that we call "entities", each one corresponding to a critical system function with respect to the validation objectives. Therefore, the behaviour of an entity corresponds to the critical system function it performs. Similarly like the inputs of a function may depend on the output produced by another function, the behaviour of an entity may depend on the behaviour of another entity. We say that there exists a "dependency relation" (or "dependency connection") from entity X to entity Y during a phase $k=[t0;t1]$ (interval of time), that we denote with $X \rightarrow_k Y$, when the behaviour of Y depends on the behaviour of X during the period starting from time t0 and ending at time t1.

As shown in [45], the system lifetime can be seen as a *sequence of phases* in which each phase is characterized by some properties. In particular, we define a phase as the period during which the dependency relations (interactions) holding between the entities remain fixed, while they vary between two successive phases.

Therefore, the lifetime of the system may be seen as a sequence of phases in which two consecutive phases have at least one different dependency relation between entities (temporal decomposition).

Thus the application of the functional and temporal decomposition to a system generates another system, that we call "phased-interacting system". Its behaviour is equivalent to that of the original one but it is built considering some critical subsystems (each one performing a critical function) and the interactions between them.

The modelling approach sketched above can be decomposed into two steps.

1. In the first step we build the whole model representing the behaviour of the generated phased-interacting system. Although built following a modular approach, the complexity of the whole model is still huge, as the modularity does not necessarily correspond to a decomposition of the solution process and, then, to a mitigation of the model complexity.
2. Therefore, in the second step, we properly modify the structure of the whole model, thus enabling the application of a decomposed solution process that consists in solving a set of simpler submodels in isolation, possibly sharing some intermediate results. The solution process relies on the identification of:
    a. the sequence in which the separate submodels have to be solved, in accordance with the existing dependency relations;
    b. the intermediate results that a separate submodel has to produce and/or use in its execution.

It is noteworthy that the abstraction-based system decomposition described in Section 8.3.1 can be seen as a particular instance of the interaction-based decomposition in which the entities are the different levels of system abstractions, and their dependency relations, defined through the interfaces, remain fixed during the entire system's lifetime (one phase only).

This type of functional/temporal system decomposition appears to have great potential in managing HIDENETS complexity, especially considering some specific HIDENETS challenging characteristics like the presence of a large number of components and the high system dynamicity (see Section 8.1). This approach is currently investigated in the course of the project considering some appropriate applications and use cases (car accident, black box application, platooning). An example of application of the interaction-based decomposition approach in the context of evolving GPRS networks is detailed in [41] and sketched in [51].

8.4.1.2 The multi-level modelling approach tailored for HIDENETS

Multi-level dependability modelling approaches (based on the hierarchical description of the target systems into various abstraction levels) are well suited to support the analytical evaluation of largely distributed systems and infrastructures, with multiple interconnected layers of software and hardware components. The hierarchical description of the HIDENETS target application from a functional and structural point of view was presented in Section 8.3.1. Here we concentrate on the hierarchical construction and solution of dependability evaluation models to assess the impact of component failures and repairs on the quality of service delivered to the users.

The dependability modelling and evaluation step is directly related to the hierarchical abstraction-based system description as presented in Figure 15. A *submodel* is associated with each abstraction level to evaluate dependability measures characterizing the quality of service delivered by the different entities of the corresponding level taking into account the outputs evaluated from lower level submodels. The outputs of a given level are used in the next immediately upper level to compute the dependability measures associated to this level.

Various techniques can be used to model each level: fault trees, reliability block diagrams, Markov chains, stochastic Petri nets, etc. The selection of the most appropriate technique depends primarily on the kind of dependencies between the elements of the considered level and on the quantitative measures to be evaluated. In particular, the modelling approach described in Section 8.4.1.1 could be applied for building the models associated with each level.

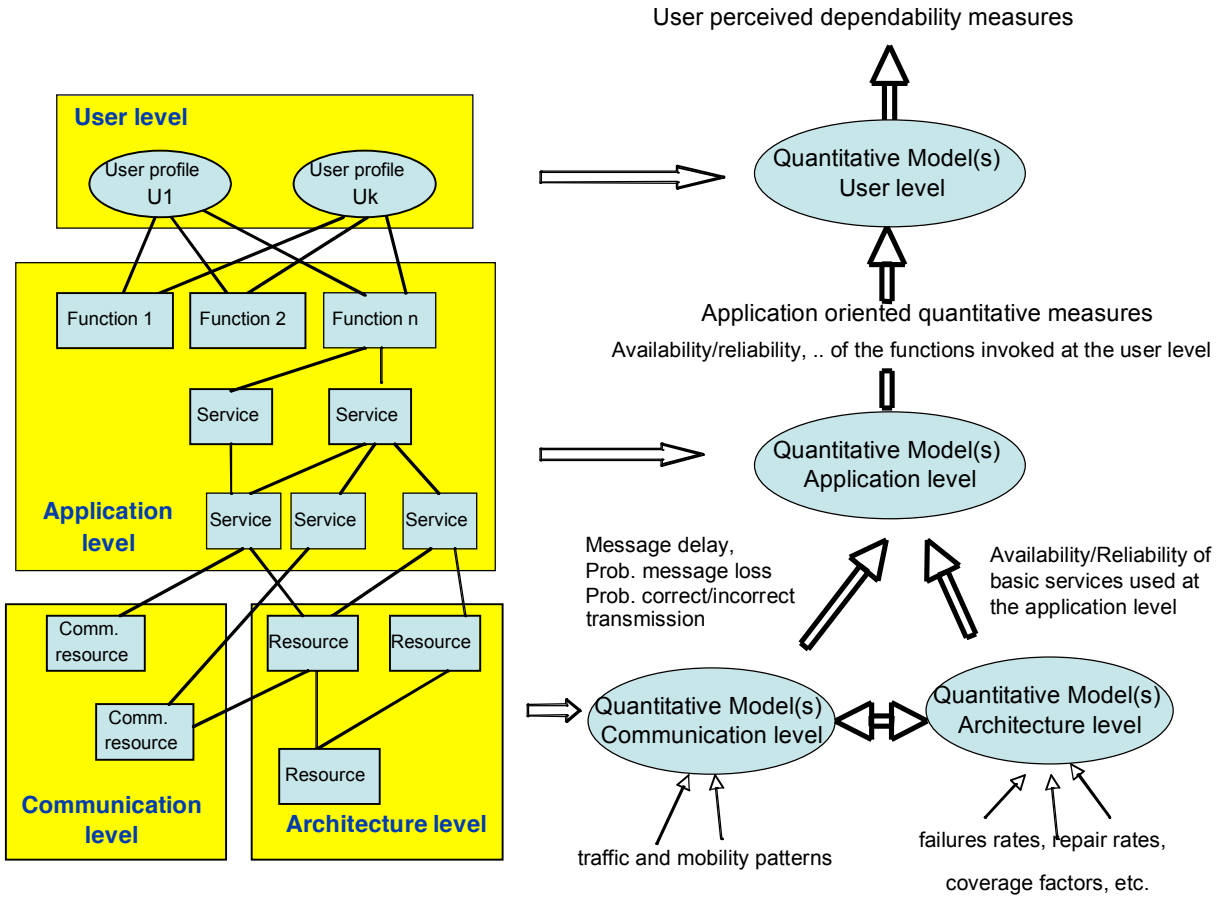


Figure 15: Multilevel and hierarchical dependability modelling framework

8.4.1.3 Dependability modelling using UML

If the system is described in a (semi)formal language then the dependability model can be constructed automatically. As a semi-formal language, UML (Unified Modelling Language) diagrams like class, object, component and deployment diagrams can be used to describe the system, while dynamic behavior can be

described by activity and state diagram.. The elaboration of proper UML conventions (profile) and a modelling workflow suited to the HIDENETS requirements is addressed in Section 10.

Basic UML does not support the description of the dependability attributes of components (resources, services and functions), and the specification of the dependability measures to be achieved. The HIDENETS UML profile adopts the General Resource Model [46] that provides a standardized approach supporting model-based dependability evaluation. Stereotypes are used to distinguish component and association types and typically tagged values are used to assign parameters.

The dependability model at a given abstraction level is a mathematically precise description of the failure and repair processes and mechanisms. Generalized Stochastic Petri nets (GSPN) and Stochastic Reward Nets (SRN) are widely used and well-supported formalisms to construct the dependability model. The UML based description of the system architecture allows a *modular composition* of this dependability model: Component and association types are associated with modules (subnets in GSPN terms) that represent their fault occurrence, error propagation and repair mechanisms.

In HIDENETS applications, the UML based dependability modelling can be applied according to the hierarchical abstraction based system decomposition approach presented in Section 8.3.1. UML conventions defined for architecture level design will identify the resource types, and these will be mapped to analysis subnets, including subnets for architecture level redundancy structures. The subnets of the hardware and software resources belonging to a service can be composed automatically to form a dependability model (submodel in Figure 15, Section 8.4.1.2). Similarly, UML conventions defined for application level models will identify the different types of services (e.g., the service model of the Application Interface Specification standardized by the SA Forum can be applied in case of infrastructural services). The services and their relations (e.g., “uses the service of” another service, or “is composed of” another services) are then mapped to analysis subnets. The latter non-trivial relations can cover service-level fault tolerance solutions (e.g., alternative service or degraded service). The input for the construction of subnets is the UML model of the application software structure that identifies the services used by the function, and the HIDENETS resilient architecture that identifies the services and their relations.

To allow the quantitative evaluation of dynamic HIDENETS scenarios, UML based dependability modelling will also support temporal decomposition. Phases of execution are identified on the basis of diagrams that describe the dynamic behaviour of the system. The dependability model is generated as a multiple phased system (MPS) model where the system net represents the system architecture (see above) and the phase net represents the phases of the scenario.

8.4.2 Simulation Methodologies

By simulation, we quite often intend to evaluate systems/mechanisms which are too complex or difficult to be analyzed using an analytical approach. A simulation evaluation normally consists of the following phases:

1. We should first have a clear view on the target to be analyzed,
    a. The relevance of system components for the evaluation, i.e., the components in the system that may impact on the simulation results.
    b. Performance metrics: definitions of the quantitative measures that will be evaluated by the simulations.
    c. All potential input to the simulation model that may impact the simulation results.
2. Given the above information, we can then list all relevant components. As it is not always easy or necessary to implement all the components involved, we quite often use the abstraction of some components, especially when the targeting system is too complex in structure and components (as it is the case in the HIDENETS system). However, finding the right abstraction level is a tricky task as it may hide important characteristic of the systems.
3. Implementation of the simulator is quite often the most resource consuming task. Depending on the complexity of the system, different approaches can be adopted, however, extensibility, modularity, processing efficiency, usability are the factors to be considered in the implementation. Finally, a simulator should be tested before the real simulation evaluation starts.

4. With the developed simulator, we still need to develop the simulation model before the simulation evaluation. The simulation model may include the parameterization of various components in the system, and the input to the simulator if available.

5. Now, we should have the raw simulation results/traces available for analysis. Careful investigation and interpretation of the results is among the most important tasks in simulation evaluation.

In HIDENETS, we take advantage of simulation methodologies to analyse: 1) network behaviour 2) behaviour of middleware components. Because of the mobility of the network nodes in the operating environment of the proposed HIDENETS applications, simulation is a feasible way to analyse the different parts of the environment.

Main input parameters for such simulation models will include:

- Specification of the geographic mobility model: for instance, geometry of the geographical space, initial node coordinates, movement patterns (speed and direction preferences of the individual mobile units), specification of (stochastic) processes for newly entering and exiting nodes,
- Rules to map geographic relations to network topology graphs: e.g. criteria for link existence which can be based on fixed transmission range or on detailed modelling of the channel, PHY and link-layer characteristics.
- Specification of the fault-model
- Specification of the application behaviour (application-layer traffic model) and potential cross-traffic within the considered networking domain.

Network simulations are supposed to model and analyze the communication level of the abstraction-based system decomposition (see Section 8.3.1). There are two main purposes for conducting network simulations. One is to evaluate individual mechanisms and the other is to give advice on how to enhance the reliability of the communication. The focus of HIDENETS with respect to network simulation is on mechanisms related to layer 2 and layer 3 in the OSI model [52]. A first step is to develop a routing simulator that simulates different rerouting techniques and how they affect the reachability and path lengths, given different mobility patterns as described in [51] Nodes and links can at first be modelled with homogeneous properties. Furthermore, support for heterogeneous properties of nodes and links should be included. An additional extension to the performance evaluation is to use the network simulator[9] NS-2 with different traffic patterns and mobility patterns as described in [51].
The network simulation on the communication level can also provide useful output to the higher levels of the abstraction-based system decomposition (see Section 8.3.1). The application and user level can utilize output regarding network performance indicators like packet drop rate, delay, jitter, throughput and path lengths. In addition, outputs including resilience indicators like recovery time and repair coverage are useful too. An important challenge to the architecture level is the scalability of the solutions, i.e., how large are the networks the HIDENETS solutions can support. All these outputs provide the other abstraction levels with an abstracted view of what happens at the communication level.

Middleware level simulations could be considered to analyse the architecture level of the abstraction-based system decomposition. The purpose of simulating the behaviour of the middleware level services is to help the analysis of the developed solutions taking into account different mobility scenarios and fault tolerance assumptions. In complex scenarios with large amounts of dynamicity of the underlying layers in the architecture usage of analytic models becomes infeasible. Hence simulation of the middleware level services is needed. An example of a HIDENETS middleware service to be analysed with respect to e.g, dependability, could be the replication manager which will be the part of the middleware that supports service replication. Results from the communication level simulations can be used as inputs to expose the service to realistic and challenging situations.

8.4.3 Experimental Evaluation Methodologies

In this sub-section, we introduce two different experimental evaluation methodologies:

[9] The network simulator –ns-2, http://nsnam.isi.edu/nsnam/ns

- instrumentation of the target system to quantitatively evaluate functional components of the HIDENETS middleware level;
- use of topology emulation as support to the experimental evaluation activity.

For what concerns the first item, Figure 16 depicts a typical experimental evaluation session in HIDENETS context, in which both infrastructure servers, laptop and PDAs/Smartphones are involved. The usage of PDAs/smartphones is relevant because these devices well represent the characteristics of a great part of mobile HIDENETS nodes (e.g., terminals in the cars), especially for a first period of usage of the HIDENETS results.

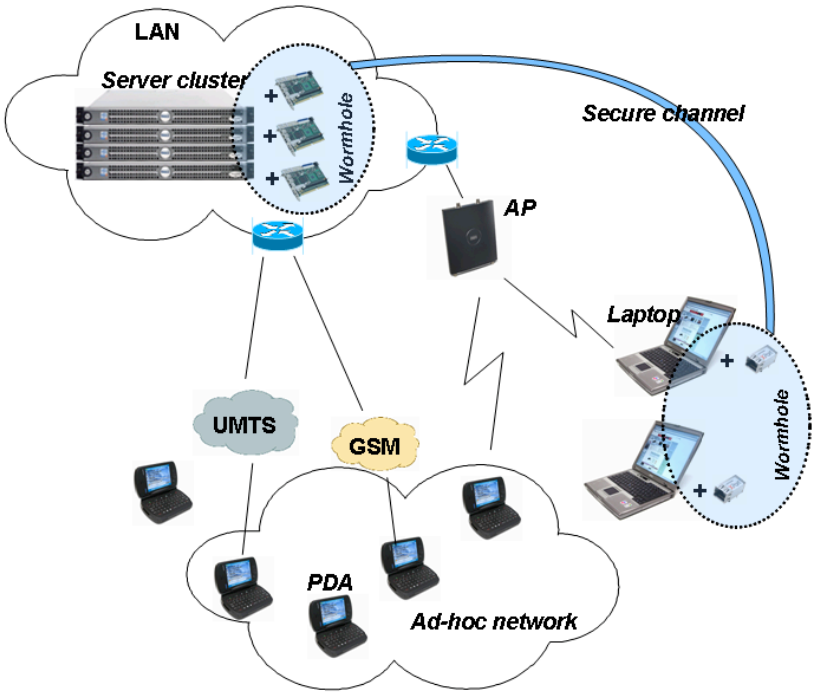


Figure 16: Devices involved in a experimental evaluation of a prototype of HIDENETS system

In this context, as reported in [51], we had first identified Neko, and more precisely its NekoStat extension[10] as a potential framework to support such experiments. Neko/NekoStat is a framework and tool recently developed for evaluation of Java based distributed systems. However, the further analyses we have carried out have shown that Neko/NekoStat was not directly usable for experimental evaluation of prototypes in the HIDENETS context: in fact the framework is based on J2SE (Java 2 Standard Edition), and the capabilities of actual JVM for small devices are currently too limited to allow a direct translation of the framework.

Using the experience acquired with the definition of the Neko/NekoStat framework, we are working on the definition of the supporting framework necessary for experimental evaluation of HIDENETS prototypes. In particular, this framework will be implemented taking into account *intrusiveness*, minimizing as much as possible the functions that have to be executed for system monitoring, in order to influence as little as possible the behaviour of the system. *Probes* and *data collection facilities* must be kept simple, both to obtain low intrusiveness and to make them easily portable in a large set of systems (e.g. systems with scarcity of resources, as PDAs/smartphones). The *data analysis components*, the part of the supports used to exploit the quantitative and qualitative information from the data collected during the execution of the prototype, ought to be executed at the termination of the execution of the system, or in any case *off-line*, in order to maintain low intrusiveness. These supports can be used to instrument and to evaluate both functional blocks of HIDENETS middleware and HIDENETS prototype applications; in the first case we will be able to obtain estimation of *QoS* (Quality of Service) of middleware components; in the second case we will be able

[10] A complete example of the usage of NekoStat for quantitative evaluation of distributed mechanisms can be found in [53].

to obtain estimation of (end-to-end) application metrics. The details of these supports will be specified in D4.1.2.

Concerning the second item, *topology emulation* is proposed to help ensuring reproducibility of the experiments. Indeed, as pointed out in Section 8.2.3, reproducibility is a tricky challenge when conducting experiments with wireless network equipments. By emulating the wireless media, reproducible experiments can actually be achieved. An emulator of the wireless media should plug into the experimental network and not require any changes/integration effort in the *ad hoc* nodes. In the simple case, the only required effort should be to plug in an Ethernet cable. In a more complex setting involving simulated positioning equipment, a GPS equipment driver interface will be implemented so that the *ad hoc* node should have the GPS software running. The software will retrieve positioning data from the network emulator. The network emulator is running a scenario where the *ad hoc* nodes in the testbed are moving around virtually. Because the emulator controls the network topology this entity will know the virtual positions of the nodes. In this advanced setting the integration will be using a standard GPS driver API and letting a Daemon run on the *ad hoc* node that retrieves the positioning information.

The details of the topology emulator are specified in D6.1 [73]. However, experimental evaluation approaches as developed in WP4 can be partially based on the existence of such a tool.

9. The Testing Framework

Software testing consists of executing a program with some valued inputs and then verifying whether the outputs conform to the expected behaviour. In this section, we address the challenges and methodologies for the verification of HIDENETS-like applications and middleware services using testing. As will be explained in Section 9.1, the characteristics of mobile applications and services provide new challenges for the testing technologies. Still, the development of appropriate methods has been seldom explored so far, and remains an open issue. The HIDENETS contribution to testing is focused on the highest layers of mobile systems. The objective is to develop solutions for the testing of applications and middleware services in mobile settings, with consideration for both functional and robustness requirements. Section 9.2 discusses the first direction for our investigation, as well as the expected results.

9.1 Challenging Issues in Testing Mobile Computing Systems

The problems raised by mobile computing systems are discussed hereafter, by considering both fundamental issues of testing and technological ones. The three fundamental points to tackle in mobile applications testing are:

i) *the determination of testing levels*,

ii) *the test selection problem*,

iii) *the so-called oracle problem*, i.e., how to assess the correctness of the test outputs.

Technological issues concern the platform required to control and observe test experiments.

9.1.1 Determination of the Testing Level

The determination of a relevant testing level requires the identification of the (sub-)system that will be the target of the testing activity, and of its interface to its surrounding environment. Typically, the testing levels are determined in the framework of an integration strategy that progressively aggregates the system components until a system level test is performed. For mobile applications and services in ubiquitous communication scenarios, a difficulty is that the notions of system boundaries, and of system components, are not as clear as for traditional applications. Moreover the composition of the system may continuously vary during testing. Generally speaking, the determination of the testing levels should depend on the target application and on the target properties to be verified. To the best of our knowledge, currently, there exist no practical guidelines to address this problem.

9.1.2 Selection of the Tests

There exist two traditional approaches in software testing for the test selection techniques: *white-box* selection approaches, and *black-box* ones.

White-box approaches are based on structural coverage criteria (e.g., branch coverage). They are applied for small pieces of software, typically at a unit testing level. An underlying assumption is that the behaviour of the target piece of software is governed by its structure. But in mobile applications, this assumption may not be sufficient. Software running in mobile devices depends not only on the application logic, but also on conditions of contextual parameters that are not explicit in the applicative code. Such a situation is exemplified by [54].

Black-box approaches need a model of the application functions. Currently, there is no standard specification language for mobile computing systems. Some authors have proposed using UML or Mobicharts [55], [56], [57] and [58], but it remains to be studied to what extent such formalisms may cover key aspects of mobile computing. Further UML extensions are mentioned in Section 10.3. Contributions on model-based testing

mainly came from the protocol testing community where the modelling language SDL (Specification and Design Language) is widely used. Since SDL is not targeted to mobile systems, some authors have proposed to introduce additional components as artefacts in the model [59][60], e.g., to represent the mobility context (which router is currently seen by a node) and to redirect the messages to the right router. Test cases can then be generated from the complete model, but only for a predetermined system configuration.

Hence, finding effective test selection techniques for mobile applications is still an open problem.

9.1.3 The Testing Oracle Problem

An oracle procedure should be investigated to compare actual test outputs with the expected ones. The best solution is to have an automated testing oracle based on a formal specification. However, as mentioned above, the specification of mobile applications is itself a problem. Moreover, in mobile settings, it may be the case that part of the inputs is uncontrollable and unobservable. For example, if a device is tested in a real environment, neighbouring devices can join and leave a network in an unpredictable manner. The resulting inputs for the target device cannot be determined, and the expected outputs cannot be decided. The solution in this situation would be to develop *partial* oracles that perform (weak) plausibility checks. Examples of such oracles for mobile applications are provided in [61] and [62]. Observability problems can also be alleviated if a simulated (rather than real) environment is used.

9.1.4 The Test Platform

The choice of the test platform must be carried out with regards to the facilities it offers to observe and control the input and output events, received or transmitted from each component. This is typically done by means of Points of Control and Observation (PCO). A PCO can be global or local. Compared to traditional systems, mobile computing systems need a higher complexity of PCOs in both cases, in order to adapt to the high dynamicity due to the mobility of entities and rich contextual parameters.

The test platform should be as realistic as possible with respect to the operational environment. However, this is not easy to achieve. In practice, it is difficult and expensive to test some mobile applications that require checking the combined logic of the involved hardware and software components. For example, in automotive domain, realistic platforms involve prototypes that are implemented into real cars (e.g., see [63]). The cost of such platforms, as well as controllability and observability constraints, imply that part of the testing activities may preferably be performed using emulation/simulation facilities.

Generally speaking, there is a trade-off to be found between the need for realism, and the need to support fine-grain control and observation to implement a given test strategy.

9.2 Preliminary Directions for the Testing Framework

HIDENETS focuses to the development of a test strategy to support the black-box testing of mobile applications and middleware services in a *simulated* environment. A discussion on the implementation of the test platform and on the test scenarios to be considered is provided hereafter. Our investigation will be supported by case studies.

9.2.1 Implementation of the Test Platform

For the implementation of a test platform for mobile-based applications, we recommend the use of three categories of components: *Application execution support*, *Network simulator*, and *Context controller*. Concrete examples of platforms built according to this generic architecture (see Figure 17) are provided in [64] and [65].

The application execution support is needed to emulate the executive support for the applicative code. A requirement is to provide an interface to the context controller and network simulator, so that the application can interact with them as if it would interact with a real environment. Since applications and

services connect over wireless links, the simulated environment must include a model of the underlying network. The network simulator is responsible for simulating the full functionality of a real wireless network. Network simulators like ns-2, GlomoSim or SWANS [66] can be used.

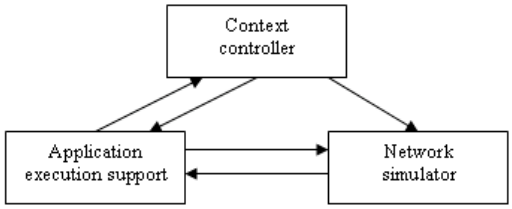


Figure 17: High-level view of the platform

The context controller is needed to simulate context information. Applications exploit context information to take different actions adaptively. Context is also needed by the network simulator to set up input parameters for imitating the real network characteristics in order to manage connectivity between nodes. Indeed, the delivery of the messages has to be done in agreement with the network topology. There have been several toolkits for simulating contexts in recent years. Some of them are built on 3D game engines and serve to simulate a first-person view of the physical environment [67]. Other examples include generic location event simulators, like [68], or traffic simulators e.g., see [87].

This simulated environment needs PCOs to control and observe the test experiments. Depending on the target application and test strategy developed, PCOs can be integrated in each component or can be realised as a global component added to the architecture.

9.2.2 Specification and Implementation of Test Scenarios

In model-based black-box testing, test cases are derived automatically from a model capturing the expected behaviour of the application. The ability to perform such testing crucially depends on the availability of adequate formalisms for the design and specification of mobile computing systems. An open issue is how to capture system-level behaviour and spatial topology in a mobile setting. As the issue is not yet mature (see e.g., [90]) a first step within HIDENETS will be to provide a language for expressing candidate test scenarios. This should be a pragmatic approach. We do not require the availability of a complete behaviour model from which the test scenarios could be automatically derived, using some coverage criteria. But at least, the user should be able to describe scenarios that are *deemed* important to be tested, and to implement them on the kind of platforms described above. Obviously, there will be a need to investigate how the underlying language could be integrated into the modelling framework described in Section 10.

In order to exemplify the concepts that should be offered by the scenario language, let us take the example of a case study we have performed [88], a Group Membership Protocol (GMP) for mobile nodes. The GMP offers a partitionable service in the ad hoc domain: groups merge and split according to location information. Decision is based on the notion of *safe distance*. The nodes in the same group must be “close enough” to prevent motion-induced disconnection for some time. The safe distance is thus lower than the communication range.

Test scenarios must consider the system behaviour in terms of message emission and reception. For example, a merge operation involves communication between leaders and members of the merging groups. Generally speaking, language scenarios for distributed systems must offer concepts to define partial orders of send/receive events. Some events are ordered, while some events may interleave in a non-deterministic way.

In addition to this, the specificity of mobile computing systems is the need to consider the spatial relationship of nodes as first class entities. For example, the GMP behaviour is governed by two spatial relations:

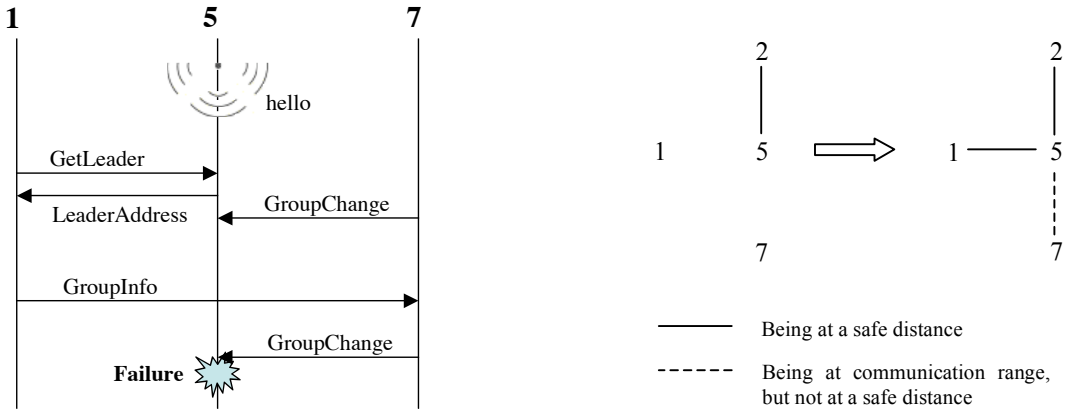
- being at a safe distance (which determines decision to split or merge groups),
- being at communication range (which determines the single-hop connectivity).

Scenarios must define the target evolution of spatial configuration to be tested. For instance, there could be several scenarios to test the split operation, depending on whether a group is to be partitioned into two or more subgroups.

The language should then offer concepts to express *both* the partial order of messages and the spatial topology of nodes. It turns out that existing languages do not cover this requirement:

- Usual scenario languages focus on the partial order of messages. Typical examples are Message Sequence Charts (MSCs) and their derivatives.
- Graph-like descriptions are convenient to represent the topology of mobile and fixed nodes. Movement can then be abstracted by successive graph transformations.

Figure 18 shows the informal representation of a GMP scenario according to these two views. It corresponds to a scenario for concurrent split and merge operations, and was indeed observed to yield a GMP failure (a GMP requirement is violated). The scenario is triggered by the hello message (containing location information) broadcasted by Node 5 after a change in spatial configuration. This hello message is received by all neighbouring nodes in the new configuration, including Nodes 1 and 7. Then, Node 1 starts a merge operation, while Node 7 starts a split. As can be seen, both views are necessary to define the scenario, because the interactions between nodes (Figure 18-a) depend on their topology (Figure 18-b). The provided language will make the dependency explicit.



(a) MSC-like view (b) Graph-like view

Figure 18: Example GMP scenario

In order to run a scenario on a test platform, an acute problem concerns the production of concrete data that instantiate the desired evolution of topology. For example, in the previous scenario, movement is abstracted by graph transformations, but ultimately the tested implementation needs to be fed with GPS input data. The data may have to obey a complex mobility model (think of cars moving in a geographical area, with a given network of roads) so that manual production is not realistic. This is why a context simulator is needed.

Our proposal is to have a preliminary production phase, based on runs of the context simulator taken in isolation:

- A run may involve a large number of nodes moving according to the implemented mobility models. At each simulation step, relevant contextual data for each node are recorded.
- The concrete simulation trace is then abstracted by a series of graphs representing the evolution of the system topology.
- It is then searched whether subgraphs can match the desired evolution pattern (defined in the scenario).
- The list of matches provides alternative baseline configurations for the implementation of the scenario.

Technically, the search for matches can be implemented by means of an algorithm for graph homomorphism building. A tool will be developed within HIDENETS.

10. The Design Methodology and Modelling Framework

Because of the challenging properties of its environment and its previously mentioned characteristics, both the HIDENETS platform itself and the possible distributed applications running on it are very complex systems. All modern techniques for designing, evaluating and testing of complex systems rely on some kind of modelling. Models are used *i) to record the specification and design of a system as unambiguous as possible, ii) to allow different kinds of analysis of the design and iii) to support the documentation of the development.* Models of the HIDENETS platform can be used to support the evaluation and testing of its building blocks, and they serve as platform model for the application development.

In this section, we outline the design methodology investigated in the context of HIDENETS to support the development, engineering and analysis of HIDENETS-like applications and services using in particular metamodels and UML profiles and design patterns.

10.1 The Design and Modelling Challenges

Modern design methodologies rely on a model driven architecture in which applications are primarily specified and also designed by their formal (or semi-formal) models. In an iterative process of specifying and designing the target system — in our case an application that will run on the HIDENETS platform — first platform independent then platform specific models are gradually refined. Each step suggests the verification and validation against the initial requirements that can be extremely challenging in this case.

This kind of application design methodologies requires a detailed model of the underlying platform to document its specific internal architecture and properties. In this way a well-drawn detailed model of the platform connects platform development and application design both of them including their own verification and validation steps. That is, how the modelling framework connects the different parts of the HIDENETS project.

Big and complex problems are much easier to be solved, if separated into smaller units, which are small enough to have a clear view of the solution for them. This principle has driven the last decade of software development, and as a result object-oriented, later component-based system development has been elaborated. In parallel with this process, modelling languages have been created that help abstract away from the global problem and provide different views of it, thus, facilitating the breaking down into smaller units.

In many cases, modelling languages are not trivial to be used. The corresponding methodologies of a given modelling / application domain are collected into modelling frameworks. These frameworks provide all the required entities and functionality for the developer during modelling and application development.

These entities and functionalities become a part of a modelling language that is specialized for the given application domain (or for the given platform). During this process new specialized model elements are introduced by specifying existing general elements of the standard modelling languages. The most widely used general purpose object-oriented modelling language, UML (Unified Modeling Language) introduces the concept of *stereotypes* for referring to these specialized model elements when the developer instantiates them in a domain (or platform) specific model. A collection of all of the stereotypes for a given domain is called a profile. While the UML is a general purpose modelling language, a UML profile provides a language to model systems for a specific domain. While defining a specific modelling language the abstraction level of modelling is raised. Some of the newly introduced model elements called *design patterns*, represent different *best practice* design solutions for the given domain. To allow application design for the HIDENETS platform with reasonable efforts we have defined a HIDENETS UML profile that is documenting design patterns, as well.

However a modelling framework provides formal definitions of all of the required entities, it is not directly applicable in the application development process, because its highly formal language is usually inconvenient for the average developer staff. That is why a domain specific editor has to be offered that

provides tools for applying the specific entities (defined by the metamodel and collected in the profile but for the sake of applicability hidden from the application developer) in the application design.

But how was the modelling framework of HIDENETS developed? First, let us define the various elements of the modelling framework (see Figure 19). The *metamodel* is the base of the modelling system. It defines the notions used in HIDENETS in a semi-formal way. Using the notions associated to such a metamodel, the *HIDENETS UML profile* could be created, which contains syntactic and semantic extensions to the basic UML language to support the description of HIDENETS specific entities. The metamodel and the UML profile then serve as the input for the creation of the *design patterns library* that contains best practice solutions for various problems.

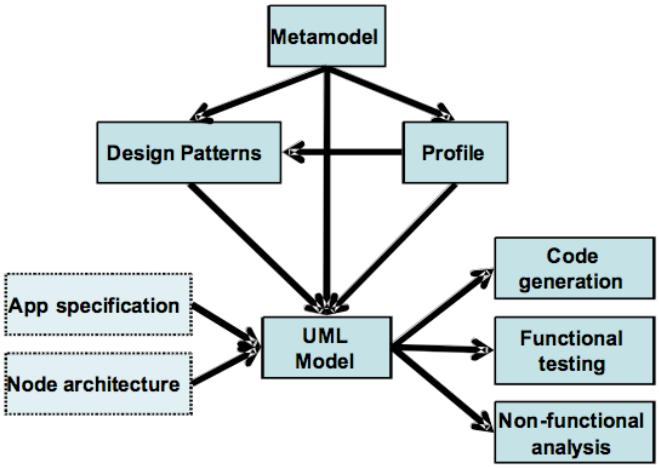


Figure 19: Relations of modelling system elements

These fundamental elements are required for the creation of *application and node architecture configuration models* that can be used for various purposes, e.g., code generation, functional and non-functional testing.

The elements mentioned above are discussed in details in what follows.

10.2 The Metamodel

A metamodel is the formalism that describes the entities and their relations in a given domain. The HIDENETS metamodel is built around three main groups of notions: the basic concepts, the application interfaces and the dependencies of services.

10.2.1 Concepts

The development of the metamodel started with the extraction of basic concepts from the deliverables that describe the HIDENETS architecture. These concepts were the main use case scenarios and their applications. Based on the use cases simple classes were created that represent the various services and other entities.

Since the information in the deliverables was too general and not precisely structured the partners were asked to specify their services and the belonging operations in more details in the form of questionnaires. In the first questionnaire the services and operations were detailed while the second one provided a top-down view of the platform i.e., which services and operations are accessible for applications. This information was later used to refine the corresponding elements in the metamodel.

It was an important consideration during the development of the metamodel that wherever it was possible standards based solutions were elaborated. Different standardized profiles for UML were used (e.g., *Fault Tolerance and Quality of Service UML Profile*, *UML Profile for Schedulability, Performance and Time, and the Reusable Asset Specification).* The advantage of using standardized profiles is that the metamodel

becomes self-contained, i.e., no additional information is needed for engineers who are familiar with those profiles to understand the semantics of the metamodel.

10.2.2 Service interfaces

As some HIDENETS nodes may run on distinct hardware platforms the implementations of the services may vary accordingly. However, the portability of applications has to be maintained. Thus, interfaces are defined that need to be provided by the different service implementations.

A specific interface has been developed for each HIDENETS service that defines all the operations that are offered for the applications. For example, the interface and the implementer class are depicted in Figure 20.

Figure 20: Authentication service interface and implementer class

Although these interfaces allow portability of applications, the writing of programs is still hard since the interfaces are specific to the HIDENETS platform. The *Application Interface Specification* (AIS) of the Service Availability Forum (SAF) provides a set of standardized application programming interfaces (APIs) for accessing services of a highly available middleware. As an alternative to the pure HIDENETS service interfaces the application access a set of higher level services (e.g., messaging, timer, naming) through the AIS interfaces as well. These selected AIS services use the existing HIDENETS services for carrying out the specific operations rather than re-implementing those functionalities.

10.2.3 Service dependencies

In the HIDENETS architecture the services are not independent from each other; one service may use the functionalities of other services. The metamodel contains these dependencies as well. This information is essential when installing new services on a node (what other services are needed for the given one) or during debugging. As an example, Figure 21 shows the dependencies of the Diagnostic Management Service and the services it depends on.

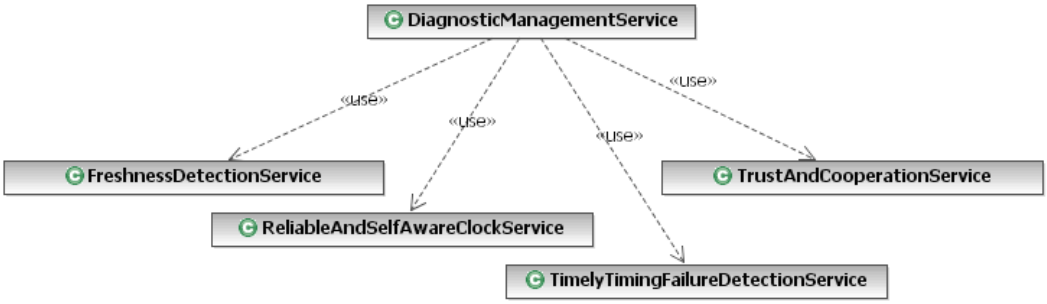


Figure 21: Dependencies of the Diagnostic Management Service

10.3 UML Profile

10.3.1 Rationale for creating a UML Profile

The metamodel defines all the abstract and concrete concepts for the given domain. However, from an application developer's perspective only those elements are relevant that can appear directly in the application's design. For example, the metamodel could contain the complete, detailed description of the different means for communication, but the developer only wants to specify that a given component needs a reliable channel for one of its functionality. Furthermore, there are several ways to extend the model with the concepts of the metamodel, though it should be done in a *standardized way*, otherwise tool support and reusability cannot be guaranteed. For these reasons, UML includes a standard way to specialize the model elements and adapt them to a specific domain: creating *UML profiles*.

A *UML profile* is a collection of elements called *stereotypes* that realize the concepts in the metamodel. A stereotype can be attached to a model element changing its semantics and notation. A stereotype shows that the given model element plays a specific role in the domain of the application. A stereotype could have parameters, these are called *tagged values*. Giving specific data for tagged values can further refine the purpose of the model element.

The following example highlights how the extension of a model with stereotypes helps the unambiguous design of applications for a specific domain.

Figure 22: Model of the example application

In Figure 22, the designer models that the application will consist of two components. However, there is no standard way in UML to model that these components will operate in a mobile environment and besides the functional requirements they have to comply to non-functional requirements also, e.g., they should pay attention to dependability and resilience properties. These constraints could be specified in external documents or UML comments; however, these approaches have the following drawbacks:

- The way of specifying the properties are *ad hoc* and non-formal, they are not integral part of the model. The information they carry cannot be used by tools in later phases.
- The developer has to explicitly define the way these properties are described; this means also that the description is hard to reuse in later application developments.

However, if first a metamodel and from it a profile would have been created, the designer could use stereotypes to represents non-functional properties (Figure 23).

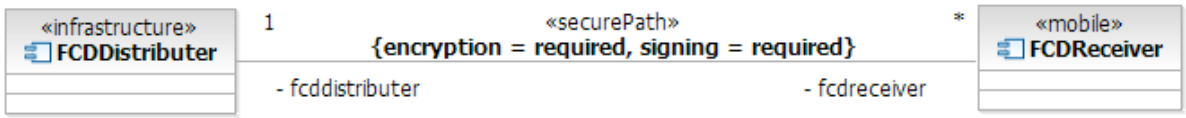


Figure 23: Example model with stereotypes

By using the stereotypes in the profile, the designer could simply tag its components to show e.g., i) which components are mobile, ii) what specific needs arise for a given communication channel.

This solution has the following advantages:

- The challenging work to define the domain-specific concepts has to be elaborated only once by the profile creators.
- The concepts are represented in a well-defined, reusable way. Applying these stereotypes to any model elements is an easy task, which is supported by most of the modern CASE tools.
- The information added with the stereotypes could be later automatically used, e.g., i) different code skeleton can be generated for a component running on a mobile node, ii) analysis techniques could check whether the specified properties are satisfied in the implementation.

More generally, defining and applying a UML profile can support the development process in many ways.

10.3.2 Workflow for defining a profile

When creating the UML profile for HIDENETS, the following approach was applied:

1. Create the metamodel as described in Section 10.2.
2. Study the domain and potential applications to identify important and frequently used concepts that are relevant when designing an application.
3. Create stereotypes representing the identified concepts.
4. Define parameters of the stereotypes, which will constitute the tagged values.
5. Optionally, specify custom notation for the stereotypes.

The second step is a key feature of the process, the other tasks are quite straightforward. The challenge in identifying the candidates is that it requires a deep knowledge of the application domain. In HIDENETS, collecting the middleware requirements of the previously selected applications helped in selecting the relevant stereotypes.

10.4 Design Patterns Library

Design patterns are previously elaborated best practice solutions for given problems. The design patterns library contains design patterns that can directly be used during modelling.

The design patterns serve three goals:

- facilitate the reuse of best practice solutions,
- reduce the number of faults in the model by automatic generation of parts of the model,
- speed up the development of applications.

The HIDENETS design pattern library contains solutions for different problem domains e.g., application related, communication related and measurement related problems.

11. Outlook

This document has provided an updated view on the HIDENETS Reference Model at the mid term of the Project. It has been put together by WP1, in close cooperation with HIDENETS WPs (WP2 to WP6). It is meant to present, in a consistent frame, the essential aspects relevant to the Reference Model that are being addressed by these WPs.

The Reference Model contains the overall approach to end-to-end resilience solutions and their assessment. As such, it presents a framework, which due to its abstraction level is not only restricted to the HIDENETS car-to-car and car-to-infrastructure applications and use-cases. The detailed solutions dedicated to the HIDENETS scenario will be further developed in the second half of the HIDENETS project until end of 2008. Adaptations and changes of the reference model may occur during this Phase and, if applicable, will be documented in the Deliverable D1.3 at project end.

The main activities during the second half of the HIDENETS project within the framework of the reference model will entail:

- refining the HIDENETS specific fault models for the purposes of the detailed technical dependability solutions in WPs 2 and 3;
- applying the HIDENETS quantitative modelling framework (WP4) and the HIDENETS testing approach (WP5) in order to assess the specific HIDENETS solutions;
- applying the UML-based metamodelling on the HIDENETS middleware and communication solutions to create a software development framework for applications in HIDENETS scenarios (WP5);
- implementing specific parts of the dependability solutions in the proof-of concept prototype set-ups (WP6) and assessing the experimental implementations using the methodologies from WP4 and WP5.

The detailed results will be presented in upcoming WP-specific deliverables, while a final view on the overall achievements of HIDENETS will be presented at project end in Deliverable D1.3.