

Project no.: IST-FP6-STREP-26979

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

Project Acronym: HIDENETS

Deliverable no.: D5.1

Title of the deliverable: UML profile and design patterns library (preliminary version)

Contractual Date of Delivery to the CEC:	31 th March 2007	
Actual Date of Delivery to the CEC:	30 th March 2007	
Organisation name of lead contractor for this deliverable:	BME	
Author(s):	András Kövi, András Pataricza, Bálint Rákosi, Gergely Pintér, Zoltán Micskei	
Work package contributing to the deliverable:	WP5	
Nature:	R	
Version:	1.0	
Total number of pages:	120	
Start date of project:	1 st 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 introduces a metamodel for HIDENETS related software development and the corresponding UML profile. The metamodel (i) presents the relation of HIDENETS services, data types etc. in an unambiguous semi-formal notation; (ii) indicates the relation of HIDENETS artifacts to previously published standards and (iii) provides a solid conceptual foundation for modelling applications running on the HIDENETS architecture.

Considerable features of the approach are the integration of previously published standards (UML 2.0, Reusable Asset Specifications, SysML, UML Profile for Schedulability, Performance and

Time, profiles for mobile systems, AUTOSAR, the Fault Tolerant UML Profile and Service Availability Forum specifications) and the application of most recent achievements from the software modelling and model transformation communities.

Keyword list: metamodel, UML profile.

Table of Contents

EXECUTIVE SUMMARY	5
ABBREVIATIONS.....	6
REFERENCES	8
1 INTRODUCTION	12
1.1 DESIGN CONTEXT	12
1.2 HIDENETS ARCHITECTURE AND PROFILE REQUIREMENTS.....	13
1.3 METHODOLOGY.....	14
1.3.1 Languages.....	14
1.3.2 Metamodelling.....	15
1.3.3 Model Driven Architecture (MDA).....	16
1.3.4 Open systems	19
1.3.5 Patterns.....	20
1.4 THE HIDENETS MODELLING APPROACH	22
2 STANDARDS AND SPECIFICATIONS	23
2.1 GENERAL MODELLING.....	23
2.1.1 Reusable Asset Specification	23
2.1.2 UML 2.0 Testing Profile	24
2.2 EMBEDDED STANDARDS.....	26
2.2.1 SysML	26
2.2.2 UML profile for schedulability, performance, and time	28
2.2.3 Profiles for mobile systems	29
2.2.4 AUTOSAR.....	31
2.3 DEPENDABILITY STANDARDS.....	33
2.3.1 Fault Tolerant UML profile.....	34
2.3.2 Service Availability Forum - Application Interface Specification Metamodel.....	36
2.4 EVALUATION	40
2.4.1 Reusable Asset Specification	40
2.4.2 UML 2.0 Testing profile	41
2.4.3 SysML	41
2.4.4 UML profile for Schedulability, Performance and Time	42
2.4.5 Profiles for mobile systems	42
2.4.6 Specifications of AUTOSAR.....	42
2.4.7 Fault tolerant UML profile.....	43
2.4.8 Specifications of the Service Availability Forum.....	43
3 THE HIDENETS METAMODEL	44
3.1 INTRODUCTION.....	44
3.2 NOTATION	49
3.3 OVERVIEW ON THE MODEL STRUCTURE.....	50
3.4 FUNDAMENTAL DATA TYPES INTRODUCED BY HIDENETS	52
3.4.1 Overview	52
3.4.2 Abstract syntax.....	52
3.4.3 Class descriptions.....	55
3.5 BASE METACLASSES FOR HIDENETS ARTIFACTS.....	57
3.5.1 Overview	57
3.5.2 Abstract syntax.....	57
3.5.3 Class descriptions.....	57

3.6	FUNDAMENTAL ORACLES DEFINED BY WP2.....	58
3.6.1	Overview.....	58
3.6.2	Abstract syntax.....	58
3.6.3	Class descriptions.....	59
3.7	HIGH-LEVEL SERVICES DEFINED IN WP2.....	63
3.7.1	Overview.....	63
3.7.2	Abstract syntax.....	63
3.7.3	Class descriptions.....	65
3.7.4	Integration of Application Interface Specification services (façade abstraction).....	68
3.7.5	Missing services.....	70
3.8	SERVICES DEFINED BY WP3.....	71
3.8.1	Overview.....	71
3.8.2	Abstract syntax.....	71
3.8.3	Class descriptions.....	73
3.9	APPLICATION INTERFACES.....	77
3.9.1	Overview.....	77
3.9.2	Interfaces of WP2 services and oracles.....	77
3.9.3	Interfaces of WP3 services.....	79
3.9.4	Interfaces for AIS façade services.....	79
3.9.5	Interfaces for client applications of the Service Availability Forum façade services.....	80
4	APPLICATION DESIGN SUPPORT.....	81
4.1	THE UML PROFILE FOR MODELLING HIDENETS APPLICATIONS.....	81
4.1.1	Representing HIDENETS-specific artifacts in UML models.....	81
4.1.2	Utility concepts introduced for supporting application development.....	92
4.2	DEPENDABILITY DESIGN PATTERNS.....	98
4.2.1	Application patterns.....	98
4.2.2	Safe measurement.....	99
4.2.3	Messenger.....	102
4.3	INTEGRATION OF TOOLS.....	103
5	CONCLUSIONS.....	105
6	APPENDIX.....	106
6.1	MODELLING CONCEPTS BORROWED FROM THE SPT PROFILE.....	106
6.1.1	Concepts of the General Resource Modelling sub-profile.....	106
6.1.2	Concepts of the Time sub-profile.....	115
6.2	MODELLING CONCEPTS BORROWED FROM THE QOS & FT PROFILE.....	117
6.2.1	Abstract syntax.....	117
6.2.2	Class descriptions.....	118

Executive summary

The current proposal summarizes the metamodel and UML profile related to the HIDENETS architecture and services. The metamodel defined here is based on the information obtained from deliverables documenting efforts carried out previously in the framework of the project (mainly work packages WP2 and WP3). The goals of the HIDENETS metamodel are (i) to present the relation of HIDENETS services, related data types etc. in an unambiguous semi-formal notation; (ii) to indicate the relation of HIDENETS artifacts to previously published standards and primarily (iii) to provide the solid conceptual foundations for modelling applications running on the HIDENETS architecture.

A primary objective in designing the metamodel was the clarification of the relation between the HIDENETS specific model-driven design and implementation approach and the standards in the field. The main policy was to assure compliance to the major standards as far as possible in order to support the future reuse of development technologies in the mainstream of embedded applications. In this context, SysML was taken into account as one of the evolving standards. An important domain-specific standard package related to the target pilot field of application (automotive) is AUTOSAR [AUTOSAR] where the fitting of the HIDENETS and AUTOSAR concepts has already been done but future compliance checks are needed as this package undergoes a recent rapid evaluation.

On the other hand, as HIDENETS aims at the assurance of dependable services over an unreliable and rapidly changing communication environment, the reuse of existing standards was of a topmost priority. This way, compliance to the standards already elaborated by the Service Availability Forum, the standardization body composed of market leading IT and telecommunication enterprises, was followed whenever possible down to the single service level.

The main promise of this approach is a high-level of portability of the applications between the mobile and fixed network based infrastructure domains.

In this context, our intention is to communicate with the standardization bodies on the practical usefulness and feasibility of a widespread use of this proposal.

Further work on the latest meta-model and UML profiles will include the following main activities:

- Further harmonization with the standards with a special emphasis on AUTOSAR.
 - Interaction with and progressing of standards in the modelling and high-availability area.
 - Continuous maintenance of this initial proposal in concordance with the evolution of the HIDENETS reference model (WP1), node architecture (WP2), and the communication infrastructure (WP3).
 - The recent meta-model can be extended by WP4 in order to support verification and validation activities and serves as a basis for the test-related activities in WP5.
 - Finally, the meta-model and UML profile will be used in WP5 to create the pilot domain-specific technology supporting the technology demonstrators in WP6.
-

Abbreviations

AIS	Application Interface Specification
AMF	Availability Management Framework
API	Application Programming Interface
AUTOSAR	AUTomotive Open System Architecture
CKPT	Checkpoint Service
CLM	Cluster Management
COTS	Commercially Off-The-Shelf
CWM	Common Warehouse Metamodel
EMF	Eclipse Modeling Framework
EVT	Event Service
FT	Fault Tolerance
GMF	Graphical Modeling Framework
GMT	Generative Modeling Technologies
GRM	General Resource Modeling
HA	Highly Available
IMM	Information Model Management Service
LCK	Locking Service
LOG	Log Service
MDA	Model Driven Architecture
MOF	Meta Object Facility
MSG	Message Service
NAM	Naming Service
NTF	Notification Service
OCL	Object Constraint Language
OMG	Object Management Group
PIM	Platform Independent Model
PSM	Platform Specific Model
QoS	Quality of Service
RAS	Reusable Asset Specification
SA	Service Availability
SAF	Service Availability Forum

SPT	UML Profile for Schedulability, Performance and Time
SUT	System Under Test
TMR	Triple Modular Redundancy
UML	Unified Modeling Language
XMI	XML Metadata Interchange

References

- [AGILE] AGILE. Architectures for mobility, IST-2001-32747, URL: <http://www.pst.ifi.lmu.de/projekte/agile/>, 2005
- [AIS] Service Availability Forum: Application Interface Specification, URL: http://www.saforum.org/specification/AIS_Information/
- [Aalst03] W.M.P van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(3), pages 5-51, July 2003
- [Acharya03] Satyajit Acharya, R.K. Shyamasundar, MOBICHARTS: A Notation to Specify Mobile Computing Applications, in Proc. of 36th Hawaii International Conference on System Sciences (HICSS-36 2003), January 6-9, 2003, Big Island, HI, USA. IEEE Computer Society, 2003, ISBN 0-7695-1874-5
- [AUTOSAR] AUTomotive Open System Architecture, URL: <http://www.autosar.org>
- [Ayed05] D. Ayed and Y. Berbers, UML Profile for the Design of a Platform-Independent Context-Aware Application, MODDM '06, Melbourne, Australia, 2006.
- [Balogh06] András Balogh and Dániel Varró: Advanced Model Transformation Language Constructs in the VIATRA2 Framework. In Proc. ACM Symposium on Applied Computing - Model Transformation Track (SAC 2006), 2006
- [Baumeister03] H. Baumeister, N. Koch, P. Kosiuczenko, P. Stevens, and M. Wirsing. UML for Global Computing. In C. Priami, editor, Proc. IST/FET Int. Wsh. Global Computing (GC'03), Rev. Sel. Papers, volume 2874 of Lect. Notes Comp. Sci., pages 1-24. Springer-Verlag, 2003
- [Belloni04] E. Belloni and C. Marcos, MAM-UML: An UML Profile for the Modelling of Mobile-Agent Applications, in Proceedings of the XXIV International Conference of the Chilean Computer Science Society (SCCC'04), 2004
- [Budinsky03] Frank Budinsky, David Steinberg, Ed Merks, Raymond Ellersick and Timothy J. Grose: Frank Budinsky, David Steinberg, Ed Merks, Raymond Ellersick, Timothy J. Grose, 2003, Addison Wesley Professional, ISBN: 0-13-142542-0
- [Budinsky05] Budinsky, F., The Eclipse Modelling Framework: Moving Into Model-Driven Development, 2005, URL: <http://www.ddj.com/184406198?pgno=1>
- [Börger03] Egon Börger and Robert Start. Abstract State Machines: A Method for High-Level System Design and Analysis. Number ISBN 3-540-00702-4. Springer, Heidelberg, 2003
- [D1.1] Markus Radimirsch, Erling V. Matthiesen, Gábor Huszerl, Manfred Reitenspieß, Mohamed Kaâniche, Inge Einar Svinnset, António Casimiro, Lorenzo Falai: Use case scenarios and preliminary reference model, HIDENETS D1.1 Deliverable
- [D2.1] António Casimiro (editor), Andrea Bondavalli, Andrea Ceccarelli, Alessandro Daidone, Lorenzo Falai, Peter Frejek, Felicita Di Giandomenico, Gábor Huszerl, Marc-Olivier Killijian, András Kövi, Erling V. Matthiesen, Odorico Mendizabal, Henrique Moniz, Thibault Renier, Matthieu Roy: Resilient Architecture, HIDENETS D2.1 Deliverable

- [D3.1] Inge-Einar Svinnset (editor), Marius Clemetsen, Paal Engelstad, Lorenzo Falai, Audun Fosselie Hansen, Tone Ingvaldsen, Tom Lippmann, Yaoda Liu, Erling V. Matthiesen, Jens Myrup Pedersen, Thibault Renier: Report on Resilient Topologies and Routing – preliminary version, HIDENETS D3.1.1 Deliverable
- [Dai05] Zhen Ru Dai, "UML 2.0 Testing Profile", in Broy, M.; Jonsson, B.; Katoen, J.-P.; Leucker, M.; Pretschner, A. (Eds.), Model-Based Testing of Reactive Systems, Springer Verlag, ISBN: 978-3-540-26278-7, 2005.
- [Doban01] Orsolya Doban, Andras Pataricza, "Cost Estimation Driven Software Development Process," *euromicro*, p. 0208, 27th Euromicro Conference 2001: A Net Odyssey (euromicro'01), 2001.
- [Douglass02] Bruce Powel Douglass: Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems, 2002, (ISBN: 0-201-69956-7) Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA
- [Douglass99] Bruce Powel Douglass: Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns, Addison-Wesley Object Technology Series, 1999, (ISBN: 0-201-49837-5) Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA
- [Eclipse] Eclipse Framework, URL: <http://eclipse.org>
- [Eclipse-EMFT] Eclipse Modelling Framework Technology, An Eclipse subproject, URL: <http://eclipse.org/emft>
- [Eclipse-EMF] Eclipse Modelling Framework, An Eclipse subproject, URL: <http://eclipse.org/emf>
- [Eclipse-GMF] Graphical Modelling Framework, An Eclipse subproject, URL: <http://eclipse.org/gmf>
- [Ecore-API] Eclipse, Ecore API, URL: <http://download.eclipse.org/modeling/emf/emf/javadoc/2.3.0/org/eclipse/emf/ecore/package-summary.html>
- [Ehrig99] H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg (eds.): Handbook on Graph Grammars and Computing by Graph Transformation, vol. 2: Applications, Languages and Tools. World Scientific, 1999
- [Fowler97] Fowler, M. Analysis Patterns: Reusable Object Models. Addison-Wesley, Reading, MA., 1997
- [Gherbi06] Abdelouahed Gherbi and Ferhat Khendek: UML Profiles for Real-Time Systems and Their Applications, in Journal of Object Technology, vol. 5, no. 4, May-June 2006, pages 149-169
- [GoF95] E. Gamma, R. Helm, R. Johnson, J. Vlissides: Design Patterns - Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995
- [Grasshopper] Grasshopper - A Universal Agent Platform Based on OMG MASIF and FIPA Standards, URL: <http://cordis.europa.eu/infowin/acts/analysys/products/thematic/agents/ch4/ch4.htm>
-

-
- [Grassi04] V. Grassi, R. Mirandola, and A. Sabetta, A UML Profile to Model Mobile Systems, in The Unified Modelling Language, LNCS 3273, Springer, 2004
- [JDT] Java Developer Tools, An Eclipse subproject, <http://www.eclipse.org/jdt/>
- [JUnit] JUnit, A Simple Framework to Write Repeatable Tests, URL: <http://junit.sourceforge.net/>
- [Litani05] Elena Litani, Ed Merks and Dave Steinberg: Discover the Eclipse Modelling Framework (EMF) and its Dynamic Capabilities, 2005, URL: <http://www.devx.com/Java/Article/29093/0/page/1>
- [MAF] Object Management Group, Mobile Agent Facility (MAF), version 1.0, 2000, URL: http://www.omg.org/technology/documents/formal/mobile_agent_facility.htm
- [MASIF] Object Management Group: Common Facilities RFP3, Request for Proposal OMG TC Document 95-11-3, Nov. 1995, <http://www.omg.org/>; MASIF specification is available through, URL: <http://ftp.omg.org/pub/docs/orbos/97-10-05.pdf>
- [MDA] Object Management Group: Model Driven Architecture Guide v.1.0.1, 2003, URL: <http://www.omg.org/cgi-bin/doc?omg/03-06-01>
- [MOF] Object Management Group: Meta Object Facility (MOF), URL: <http://www.omg.org/mof/>
- [Muscutariu01] F. Muscutariu and M.P. Gervais, On the Modelling of Mobile Agent-Based Systems, In Mobile Agents for Telecommunication Applications: Third International Workshop, MATA 2001, Montreal, Canada, August 14-16, Springer, 2001
- [OCL] Object Management Group: UML 2.0 OCL Specification, 2003, URL: <http://www.omg.org/docs/ptc/03-10-14.pdf>
- [Pataricza06] A. Pataricza, B. Polgár, Sz. Gyapay, A. Balogh, Gy. Csértán, “Formal checking of metamodels and models,” In Proc. of the DECOS/ERCIM Workshop, ad joint to SAFECOMP 2006, Gdansk, Sept. 26, 2006
- [Purao03] Sandeep Purao, Veda C. Storey, Taedong Han. Improving Analysis Pattern Reuse in Conceptual Design: Augmenting Automated Processes with Supervised Learning, Information Systems Research, Vol. 14, No. 3, September 2003, pp. 269-290
- [QoSFT] Object Management Group: UML Profile for Modelling QoS and FT Characteristics and Mechanisms, v1.0, OMG Specification, 2006
- [RAS] Object Management Group: Reusable Asset Specification (RAS), 2005, URL: <http://www.omg.org/technology/documents/formal/ras.htm>
- [SAF] Service Availability Forum, URL: <http://www.saforum.org/home>
- [SPT] Object Management Group: UML Profile for Schedulability, Performance and Time, version 1.1, 2005
-

-
- [Schmidt00] Douglas Schmidt, Michael Stal, Hans Rohnert and Frank Buschmann. Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects, Volume 2, ISBN: 0471606952, John Wiley & Sons, 2000
- [Schmidt96] Douglas C. Schmidt, Ralph E. Johnson, Mohamed Fayad. Software Patterns, Communications of the ACM, Special Issue on Patterns and Pattern Languages, Vol. 39, No. 10, October 1996
- [TTCN3] Testing and Test Control Notation (TTCN-3), URL: <http://www.ttcn-3.org/home.htm>
- [UML] Object Management Group: Unified Modelling Language, 2006, URL: <http://www.uml.org/>
- [U2TP] Object Management Group: UML Testing Profile v1.0 (U2TP), 2005, URL: http://www.omg.org/technology/documents/formal/test_profile.htm
- [UML2.0] UML 2.0 Superstructure Specification. August 2003, ptc/03-08-02. Unified Modelling Language Infrastructure Specification, September 2003, ptc/03-09-15. <http://www.omg.org/>
- [VIATRA2] VIATRA2 Framework, An Eclipse GMT Subproject, URL: <http://www.eclipse.org/gmt/>
- [Valiente05] Mari Cruz Valiente, Gonzalo Genova, Jesus Carretero: UML 2.0 Notation for Modelling Real-Time Task Scheduling, in Journal of Object Technology, vol. 5, no 4, May-June 2005, pp. 91-105
- [Varró03] Dániel Varró, András Pataricza: VPM: A Visual, Precise and Multilevel Metamodelling Framework for Describing Mathematical Domains and UML, Journal of Software and Systems Modelling vol. 2 pp. 187-210, 2003
-

1 Introduction

This section presents an overview on the HIDENETS design context, enumerates the requirements against the metamodel to be developed (e.g., should support modelling application structure both at the application and the node level, should provide a solid conceptual basis for automatic code generation, etc.), outlines the modelling methodologies and paradigms integrated into our approach (modelling languages, metamodelling, model driven architecture, open modelling environments and design patterns) and highlights the key parts of our contribution (i.e., the metamodel itself, the UML profile based on the metamodel and the design pattern library).

1.1 Design context

Modern design methodologies fit under the model-driven architecture initiative in which applications are primarily designed and specified by their (semi-) formal model. After each major step in the design workflow these models are verified and validated against the initial requirements those generating a series of gradually refined models. The final implementation model can serve as the basis of automated code generation (the generated code may range from code frameworks defining the calling sequences together with their parameterization to a complete automation generating a final run-time code).

Benchmark-based experiences indicate [Doban01] that, depending on the support on the terms of formal verification and validation, this approach promises an increase in productivity of one order of magnitude completed by an improved quality by one and half orders of magnitude in the terms of residual faults in the target code.

The additional benefits of a model-based approach originate in the uniform specification at all the level of the design workflow and a high-level of reusability of the embedded intellectual property:

- A meta-model has the advantage over the traditional verbal formulation that it delivers a compact and unambiguous specification of the modelling notations and their interrelation. The compliance of a particular special meta-model like that of the HIDENETS approach to the standards to which it has to comply (for instance, different fundamental UML dialects, SA Forum standards etc.) can be even formally proven by a novel, ontology-based approach developed in the framework of the DECOS¹ IP at BUTE [Pataricza06].
- In the context of embedded systems the modelling paradigm has to be compliant with different standards in order to support the reuse of existing development support technologies like model editors, V&V tools etc. An important element of a tool-chain is the availability of open source solutions for the most common development tasks in a reusable and customizable form. For instance, Eclipse [Eclipse] the target platform selected to serve as the fundamental framework for the HIDENETS software development technology, already provides such fundamental services as customizable editors or a version control. The use of such a widespread technology has the additional benefit that the designer can rely on a well-known user interface which is basically independent in its philosophy from the actual application domain.

¹ DECOS – Dependable Embedded Components and Systems, IST 511764, EU 6th FWP

- General purpose languages, like the basic UML, are tedious in the case if some repetitive elements (like HIDENETS middleware calls) appear frequently as the user is forced to repeat the elementary model component creation operations despite the fact that they are template-like. Domain-specific editors support the reduction of this routine workload by introducing stereotyped elements and higher level constructs into the graphical editor. Modern support services, like GMF, the Graphical Modelling Framework in Eclipse, are able to generate so-called domain-specific editors from the meta-model representing the notations complemented by the specification of the appearance of the visual elements.
- Automated transformations can be used to generate out of an application model different formal analysis codes (e.g. Petri Nets, queuing networks, etc.) in addition to the run-time code. Moreover, automated analysis may cover the generation of such models which differ from the original one for the purpose of a “what-if” analysis. A typical example for such an analysis is dependability evaluation of the overall architecture in which automated transformations are able to generate faulty mutations of the application model based upon the technology related knowledge on fault mechanisms. These derived mutations can be made subject of a formal analysis proving the fulfilment of the dependability related requirements.

1.2 HIDENETS architecture and profile requirements

In this deliverable we introduce a framework for the HIDENETS platform. The main objective of the framework is to facilitate the development of HIDENETS based applications. The framework follows the principles of the Model-driven Development on the basis of UML technology. The two basic elements of the framework, that are described in this report, are the metamodel of the HIDENETS platform and the UML profile that extends the semantics of Basic UML.

Levels of requirements. There are certain requirements that the framework has to satisfy to provide all the functionalities that are needed for application development. These requirements can be conceptualized on two levels of the HIDENETS platform: the application level and the node level.

Application level. Application level requirements describe the functionalities that are needed for the development of application structures. These are the following:

- **Model application structure and behaviour aspects.**

Model-driven development is based on models which are abstractions of the application from different perspectives. These perspectives can be e.g. the application structure, behaviour of components and middleware service access. Furthermore these models should support the programmatic checking of various application properties like provided quality of service.

- **Provide automatic code generation facilities.**

Many faults are introduced during programming by humans, thus the automatic generation of application code is an essential part of the introduced framework.

Node level. Another group of requirements corresponds to the node level of the HIDENETS platform. Here we have the following requisites:

- **Ability to describe the service components and deployment configurations.**

It is substantial to be capable of describing the service configuration of a node in the system. The service configuration contains the components that provide the services and their interfaces.

- **Model service-service dependencies.**

Service-service dependencies show which service requires which other services during operation; such information is essential for correct deployment of services and applications on HIDENETS nodes, and for analysis of non-functional properties.

1.3 Methodology

In this section we discuss the methodologies that are used during the creation of the HIDENETS metamodel. First we introduce the concepts of modelling languages and UML in detail. Then we deal with the methodologies of the Metamodelling and Model Driven Architecture. Next we give a short introduction on open systems. Last we discuss the usage of patterns.

1.3.1 Languages

As nowadays' systems are getting more-and-more complex new ways had to be found to keep the control over the creation and maintenance of them. As a result, different modelling languages have been created that provide abstract, domain specific views and thus simplify the overview of these tasks. One of these languages is the Unified Modelling Language.

The Unified Modelling Language (UML) [UML] is a standard of the Object Management Group (OMG). UML is a visual language for specifying, constructing and documenting the artifacts of systems. It is a general-purpose modelling language that can be used with all major object and component methods, and that can be applied to all application domains and implementation platforms. During the last few years UML has emerged as the software industry's dominant modelling language. It is widely accepted among system designers, analysts and programmers. The UML specification is defined using a metamodelling approach that adapts formal specification techniques. While this approach lacks some of the rigor of a formal specification method, it offers the advantages of being more intuitive and pragmatic for most implementers and practitioners.

UML was designed as a general modelling language. However, instead of defining all the modelling concepts of the domains where UML could be used, the specification contains only some core elements and a standardized extension mechanism is given. These extensions include the *Constraint*, *Stereotype*, and *TaggedValue* constructs. A constraint is an expression that restricts the structure or the behaviour of an element, usually written in the Object Constraint Language (OCL) [OCL]. A stereotype defines how an existing class may be extended, and enables the use of platform or domain specific terminology or notation in place of or in addition to the ones used for the extended class. A stereotype can be attached to a modelling element to further classify it and add additional properties to it. E.g. in software modelling a class can be labelled with the Web Form stereotype (marked with writing <<Web Form>> before the name of the class), and a tool can use this information to visualize the element with a specific graphics. A tagged value is a property defined for a stereotype. E.g. in the previous example the Web Form stereotype can have the title and the font size parameters.

1.3.2 Metamodelling

Metamodelling is the precise definition of a modelling language. A metamodel consists of the concepts and rules that can be used in a model for a specific problem. More precisely, for the definition of a modelling language the followings shall be given i) the abstract syntax defining the concepts of the given domain and their relations, ii) the concrete syntax defining the textual or graphical notations of the concepts, iii) well-formedness rules defining further constraints for the concepts, and iv) the formal semantics defining the dynamic behaviour of the models.

One of the first metamodelling frameworks is OMG's standard Meta-Object Facility (MOF) [MOF]. The MOF specification defines an abstract language and a framework for specifying, constructing, and managing technology neutral metamodels. A metamodel is in effect an abstract language for some kind of metadata. Examples include the metamodels for UML, CWM, SysML and the MOF itself.

The specification of MOF includes the following aspects:

- a formal definition of the MOF meta-metamodel; that is, the abstract language for specifying MOF metamodels,
- an XMI format for MOF metamodel interchange.

The XML Metadata Interchange (XMI) specification defines technology mappings from MOF metamodels to XML DTDs (Document Type Definition) and XML documents. These mappings can be used to define an interchange format for metadata conforming to a given MOF metamodel (see more in the section about XML).

UML and MOF are normally viewed in the context of a conceptual layered metadata architecture. Although the metamodels for MOF and UML are designed to be architecturally aligned, sharing a common subset of core object modelling constructs, this does not bind the modeller to stick to UML as the modelling language. Just on the contrary, the whole metamodelling mechanism is useful to provide a common modelling framework where model instantiation can occur using different modelling languages.

The classical framework for metamodelling is based on an architecture with four metalayers. These layers are conventionally described as follows:

1. the information layer with the data that should be described;
2. the model layer with an abstract representation of the data in the information layer;
3. the metamodel layer with the descriptions that define the structure and semantics of metadata;
4. the meta-metamodel layer with the description of the structure and semantics of meta-metadata.

Figure 1 depicts this classical four layer framework illustrated with a HIDENETS, platooning use case related example: the *metamodel layer* contains a metaclass (PhysicalNode) taken from the metamodel discussed below; the *model layer* presents a fragment of a software model building on the HIDENETS metamodel (introducing classes FirstVehicle and FollowingVehicle as possible members of a platoon) while the *information layer* presents an actual platoon (in the form of an object diagram) where the platoon consists of a first vehicle and two following vehicles.

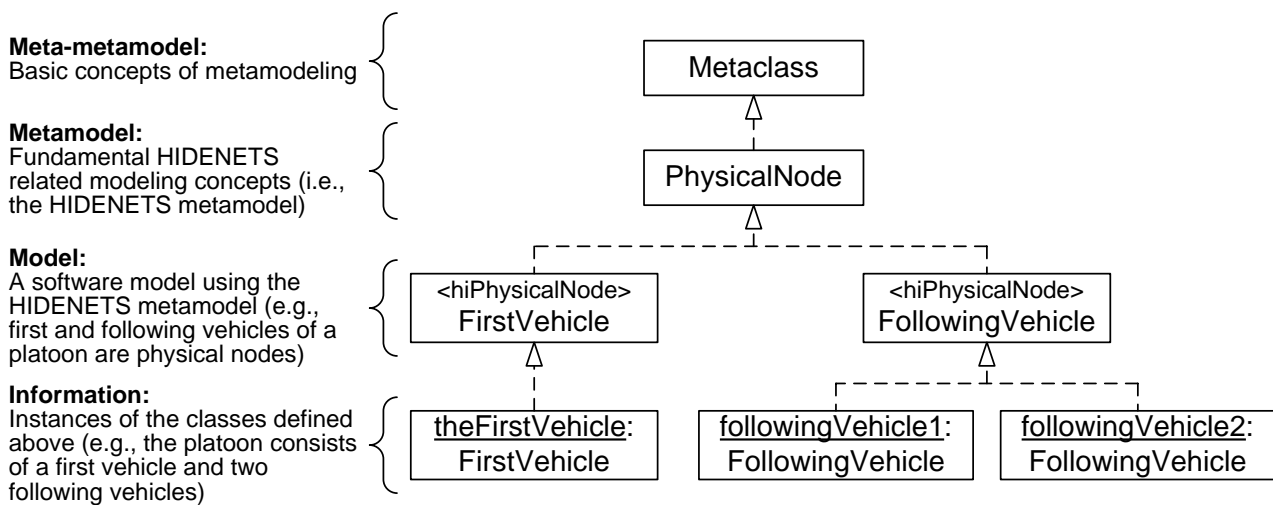


Figure 1: Illustration of the MOF 4-layer framework

As the first adopted technologies specified using a metamodeling approach, the UML, MOF, and XMI provide the foundation for OMG's Model Driven Architecture (MDA). Future metamodel standards should reuse MOF and UML's core semantics and emulate their systematic approach to architecture alignment.

1.3.3 Model Driven Architecture (MDA)

The Model Driven Architecture (MDA) [MDA] is a specification of the Object Management Group. MDA starts with the well-known and long established idea of separating the specification of the operation of a system from the details of the way that system uses the capabilities of its platform. MDA provides a powerful and elegant conceptual framework for the consistent integration of semi-formal visual modelling, model analysis based on formal verification and validation techniques and code generation (Figure 2).

MDA provides an approach for and enables tools to be provided for:

- specifying a system independently of the platform that supports it,
- specifying platforms,
- choosing a particular platform for the system, and
- transforming the system specification into one for a particular platform.

The three primary goals of MDA are:

- portability,
- interoperability,
- reusability.

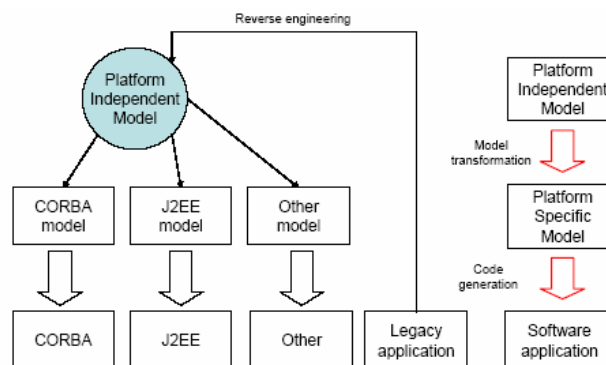


Figure 2: Model Driven Architecture approach

The main concepts of MDA are (excerpt from the specification):

- *System*: A system may include anything that exists or is planned to exist: a program, a single computer system, some combination of parts of different systems, a federation of systems, each under separate control, people, an enterprise, and a federation of enterprises.
- *Model*: A model of a system is a description or specification of that system and its environment for some certain purpose. A model is often presented as a combination of drawings and text. The text may be in a modelling language or in a natural language.
- *Application*: An application is a functionality being developed. A system is described in terms of one or more applications supported by one or more platforms.
- *Platform*: A platform is a set of systems and technologies that provide a coherent set of functionality through interfaces and specified usage patterns, which any application supported by that platform can use without concern for the details of how the functionality provided by the platform is implemented.
- *Platform independent model (PIM)*: A platform independent model is a view of a system from the platform independent viewpoint. A PIM exhibits a specified degree of platform independence so as to be suitable for use with a number of different platforms of similar type.
- *Platform specific model (PSM)*: A platform specific model is a view of a system from the platform specific viewpoint. A PSM combines the specifications in the PIM with the details that specify how that system uses a particular type of platform.
- *Platform model*: A platform model provides a set of technical concepts, representing the different kinds of parts that make up a platform and the services provided by that platform. It also provides, for use in a platform specific model, concepts representing the different kinds of elements to be used in specifying the use of the platform by an application. A platform model also specifies requirements on the connection and use of the parts of the platform, and the connections of an application to the platform.
- *Model transformation*: Model transformation is the process of converting one model to another model of the same system.
- *Implementation*: An implementation is a specification, which provides all the information needed to construct a system and to put it into operation.

MDA is said to be model-driven because it provides a means for using models to direct the course of understanding, design, construction, deployment, operation, maintenance and modification. It suggests using different models at the different levels of abstraction and at different phases of development during system development. The process of development will consist of the composition and refinement of models. Transformations serve as means of progressive refinement of models from abstract, platform independent, requirement centric towards concrete, platform specific, implementation centric.

Our HIDENETS modelling approach fits to the MDA initiative as discussed below:

- The *platform independent model* of an application is a UML model where the restrictions enforced or services provided by the platform are still not taken into consideration i.e., this is a model where the underlying platform has not yet been chosen thus no commitments were made for the actual implementation of various services.
- The *platform specific mapping* of the platform independent model takes into consideration the specialties of the target platform for the most suitable implementation of some services, e.g., with respect to the HIDENETS platform the modeller may decide to use the reliable communication middleware provided by HIDENETS for the implementation of some application specific messaging function. Platform specific UML models are stereotyped according to the corresponding profile.
- The *implementation* of a platform specific model obviously uses the implementation of various platforms specific services, e.g., in case of the example above the messaging function will call HIDENETS service implementations as COTS components.
- Having clearly separated the three key steps of the development the power of MDA can be exploited by creating *automatic tools* that are capable of *transforming* platform specific models to their implementation (e.g., generating source code or at least code frames, configuration files, deployment descriptors etc.).

1.3.3.1 Model transformation

MDA highly relies on effective tool support, which is still lacking today. Most importantly tools and engineering practices are needed not only for model engineering (*'modelware'*), but for transformation engineering (*'transware'*). Below we give a short overview on the VIATRA2 graph transformation system that aims at providing the necessary technological foundations for automatic model transformations to be used in MDA-based development.

The main objective of VIATRA2 (VIual Automated model TRAnsformations [VIATRA2]) is to provide general-purpose support for the entire life-cycle of engineering model transformations including the specification, design, execution, validation and maintenance of transformations within and between various modelling languages and domains.

VIATRA 2.0 is a generic model integration and transformation framework integrated into the Eclipse platform, by the means of plugins, offering a large variety of development and design tools for the user.

The basic architecture of VIATRA2 uses a central model space that can contain an arbitrary number of models and meta-models simultaneously. A generic graphical user interface is attached to this repository to support the browsing and manipulation of models. The framework has an extendable set of model import and export modules that serve the integration of various modelling tools. The transformation engine also works on the model store and uses a powerful command language

(called VIATRA Textual Command Language VTCL) [Balogh06] that is composed of graph transformation rules and abstract state machines (ASM) [Börger03].

VIATRA 2, a general model transformation system, has been developed in order to support the dynamic, multilevel metamodelling features of the visually precise and multilevel metamodelling framework VPM [Varró03], and generic/meta transformations. The main intended usage of the framework is dependability evaluation and optimization of business process workflow models and UML models.

A source user model (which is a structured textual representation such as an XMI description of a UML model exported from a CASE tool) is imported into the VPM model space. Transformation specifications can be constructed by combining graph transformation [Ehrig99] and abstract state machine [Varró03] rules. These rules can be created within the framework or in a UML tool using a special profile.

The rules are then executed on the source VPM model by the generic (higher-order) VIATRA rule interpreter in order to yield the target (VPM) model. Finally, the target model can be serialized into an appropriate textual representation specific to back-end tools.

As mentioned above, model transformations may be used for automatically transforming HIDENETS platform-specific software models to source code, configuration files, deployment descriptors etc. The application of such automatic tools promises a significant increase in the productivity, reducing development costs and enhancement of software quality.

1.3.4 Open systems

Interoperability is a recurring problem in the history of computer systems development. Open systems have been developed with publicly known interfaces to overcome this issue. Well known examples are e.g. POSIX, CORBA or Web services.

A wide range of tools has been developed to support the standards based software engineering process. In the recent years the integration of tools is getting more and more attention. Eclipse is one of the most widely used frameworks that support such integration. Since Eclipse is a primary candidate for providing the basis for application modelling and model transformations (possibly used for implementing the MDA initiative in the context of HIDENETS as outlined above), below we give an overview of the Eclipse related technologies.

1.3.4.1 Tool integration in the Eclipse framework

The *Eclipse Project* [Eclipse] is an open source software development project dedicated to providing a robust, fully-featured, commercial quality, industry platform for the development of highly integrated tools. It was developed by IBM from 1999, and a few months after the first version was shipped, IBM donated the source code to the Eclipse Foundation. The Eclipse project consists of many subprojects, the most important being the Eclipse Platform, that defines the set of frameworks and common services that collectively make up the "integration-ware" required to support a comprehensive tool integration platform. These services and frameworks represent the common facilities required by most tool builders, including a standard workbench user interface and project model for managing resources, portable native widget and user interface libraries, automatic resource delta management for incremental compilers and builders, language-independent debug infrastructure, and infrastructure for distributed multi-user versioned resource management.

Eclipse Plugin architecture. The Eclipse Platform has an easily extendable modular architecture, where all functionality is achieved by plugins, running over a low-level core called Platform Runtime. This runtime core is only responsible for loading and connecting the available plugins, every other functionality, such as the editors, views, project management, is handled by plugins. The plugins bundled with Eclipse Platform include general user interface components, a common help system for all Eclipse components, project management and team work support.

Eclipse Modelling Framework. Eclipse Modelling Framework (EMF) [Eclipse-EMF] is a Java framework and code generation facility for building tools and other applications based on a structured model. EMF provides a meta-model (Ecore) [Ecore-API] for describing structured models. Using these structured models EMF provides tools and runtime support to produce a set of Java classes representing the model, a set of adapter classes that enable viewing and a basic editor.

Graphical Modelling Framework. The Eclipse developers have realised that there is a need for rapid language engineering. Thus, an intermediate framework was drafted, whereby a diagram definition would be linked to a domain model as input to the generation of a visual editor. This is the Graphical Modelling Framework (GMF) [Eclipse-GMF] project, which aims to provide the fundamental infrastructure and components for developing visual design and modelling surfaces in Eclipse. The typical workflow to create a domain-specific editor and a code generator is the following: (1) create an "EMF model" (which is actually the domain metamodel); (2a) create diagram metamodels; (2b) create a mapping model between the EMF model and the diagram model; (3) refine the domain metamodel on a graphical interface, add OCL constraints; (4) design the visual representations for diagram elements using a graphical interface; (5) generate the domain-specific visual editor; (6) use the generated Eclipse plug-in for modelling; (6) manually implement a code generator; (7) generate application code.

1.3.5 Patterns

A pattern is a recurring solution to a standard problem. When related patterns are woven together they form a "language" that provides a process for the orderly resolution of software development problems. Pattern languages are not formal languages, but rather a collection of interrelated patterns, though they do provide a vocabulary for talking about a particular problem. Both patterns and pattern languages help developers communicate architectural knowledge, help people learn a new design paradigm or architectural style, and help new developers ignore traps and pitfalls that have traditionally been learned only by costly experience [Schmidt96].

Patterns are usually given in a structured format. For example [Schmidt00] uses the following sections. *Example* section introduces the *Context*, *Problem*, and *Solution* sections, which summarize a pattern's essence. The *Solution* section foreshadows the *Structure* and *Dynamics* section, which then present more detailed information about how a pattern works, preparing for the *Implementation* section. The *Example Resolved*, *Variants*, *Known Uses*, *Consequences* and *See Also* sections complete each pattern description. Many other formats have been used in other publications, thus OMG created the Reusable Asset Specification to unify the definition of patterns.

Design patterns. Design patterns describe solutions to typical object oriented problems. Design patterns are usually organized in the following categories: creational, structural, and behavioural [GoF95]. A typical design pattern is the abstract factory, which provides an interface for creating families of related or dependent objects without specifying their concrete classes.

Architectural patterns. Architectural patterns describe higher-level solutions when designing complex systems. For example, in case of a concurrent and networked middleware proven patterns exist for event handling, synchronization or service access.

A well-known architectural pattern is the *Reactor* (also known as Dispatcher) [Schmidt00] pattern, which is used when a system receives multiple concurrent request simultaneously, but processes them serially. The reactor handles the demultiplexing and dispatching of the service requests. The structure of the pattern is illustrated on Figure 3.

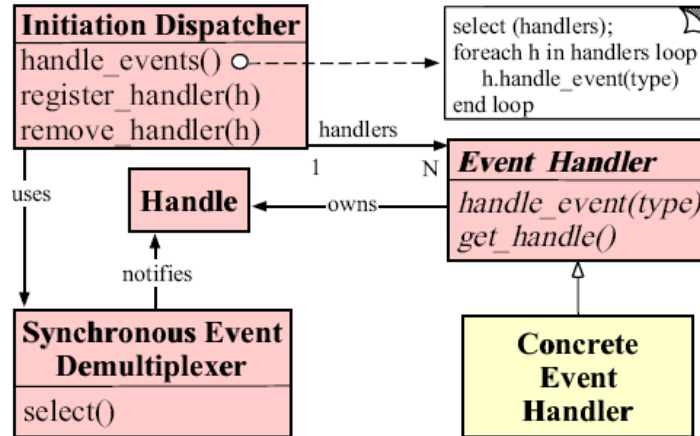
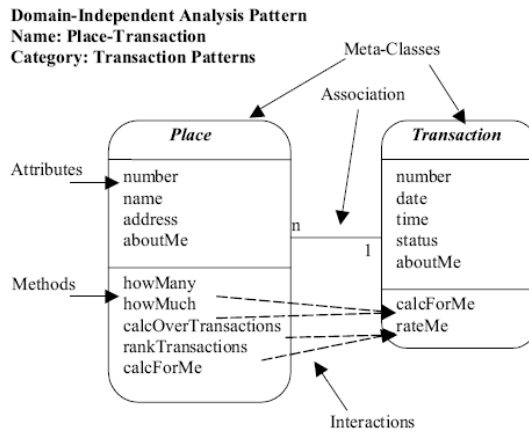


Figure 3: The Reactor pattern [Schmidt00]

Analysis patterns. An analysis pattern is a group of related, generic objects with stereotypical attributes, behaviours, and expected interactions defined in a domain-neutral manner [Fowler97]. The patterns define more abstract concepts and could be used in the conceptual design phase.



Typical Object Interactions:
 howMany → calcForMe calcOverTransactions → calcForMe
 howMuch → calcForMe rankTransactions → rateMe

Examples:

- Place: airport, assembly line, bank, clinic, depot, garage, geographic entity, hangar, hospital, manufacturing site, plant, region, sales outlet, service center, shelf, station, store, warehouse, zone.
- Transaction: agreement, assignment, authorization, contract, delivery, deposit, incident, inquiry, order, payment, problem, report, purchase, refund, registration, rental, reservation, sale, shift, shipment, subscription, time charge, title, withdrawal.

Combinations:

- Participant-Transaction; Specific Item-Transaction; Transaction-Transaction Line Item; Transaction-Subsequent Transaction.

Figure 4: An example for an analysis pattern [Purao03]

In Figure 4 an analysis pattern called Place-Transition is described with its structure and the domains it can be applied.

Workflow patterns. Workflow patterns refer specifically to recurrent problems and proven solutions related to the development of workflow applications in particular, and more broadly, process-oriented applications. In [Aalst03] patterns are categorized as control-flow, resource, data, and exception handling patterns. The patterns range from simple constructs, e.g. multi-choice, also to complex routing primitives, like Cancelling Partial Join for Multiple Instances.

1.4 The HIDENETS modelling approach

In this chapter we present the modelling approach that was used for the elaboration of the HIDENETS modelling framework. The work consists of the following parts:

- **Metamodel.** The metamodel is a semi-formal description of the HIDENETS platform. The metamodel defines the following:
 - **Concepts** used in the application and the resilient middleware. Whenever it was possible a representation defined in a standard was used for each concept.
 - **Interfaces** defining the interaction points between the applications and the HIDENETS middleware. The interfaces are based on the SA Forum interfaces, only extended if a new functionality was introduced by a HIDENETS service.
 - **Service dependencies** describing the interaction between the HIDENETS services.
- **Profile.** The UML profile is the representation of the metamodel from an application developer viewpoint. With the elements defined in the profile the developer can use the functionality of the platform in a straightforward, declarative way in its application.
- **Design pattern library.** The more complex scenarios, e.g. when the functionality of multiple services is used, that are common in many applications could be defined in the form of reusable design patterns. The design pattern library is a collection of such HIDENETS related patterns.

Please note that the metamodel described here is in an initial phase of the development procedure and further iterations steps will refine it based on the progress of the other work packages.

2 Standards and specifications

This chapter presents a short overview on previously published standards concerning (i) some general modelling aspects (reusing components and testing), (ii) embedded software development (various UML profiles proposed in the literature), (iii) dependability-related specifications (UML fault tolerance profile, Service Availability Forum specifications) and (iv) automotive industry's domain-specific modelling initiatives (AUTOSAR).

The structure of the next section is as follows. After the overview of all the relevant standards and specification each of them is evaluated in Section 2.4 according to the following criteria: compliance to HIDENETS objectives, industrial relevance, extensibility, extension requirements, tool support, and openness.

2.1 General modelling

In this section we give an overview of the standards that are used for the creation of the metamodel or provide methodology and notational conventions that are to be used later in the project.

2.1.1 Reusable Asset Specification

To support the reuse of components and solutions for various problems, OMG has defined the Reusable Asset Specification (RAS) that introduces a rigorous hierarchy for describing software assets. RAS focuses on the classification, content, and usage of software assets.

An asset is a solution for a software development problem. The problem may be related to the evolution of the system's artifacts or be directly related to the domain problem that the system is being developed for. If an asset has been developed with reuse in mind then we call it a reusable asset.

RAS is described in two major categories, Core RAS and RAS Profiles. Core RAS represents the fundamental elements of asset specification, while RAS Profiles contain extensions to those fundamental elements. A RAS Profile can extend the Core RAS or another RAS Profile (see Figure 5). A RAS Profile must not loosen the constraints defined on the elements of its ancestor Profile but imposing stricter rules is allowed.

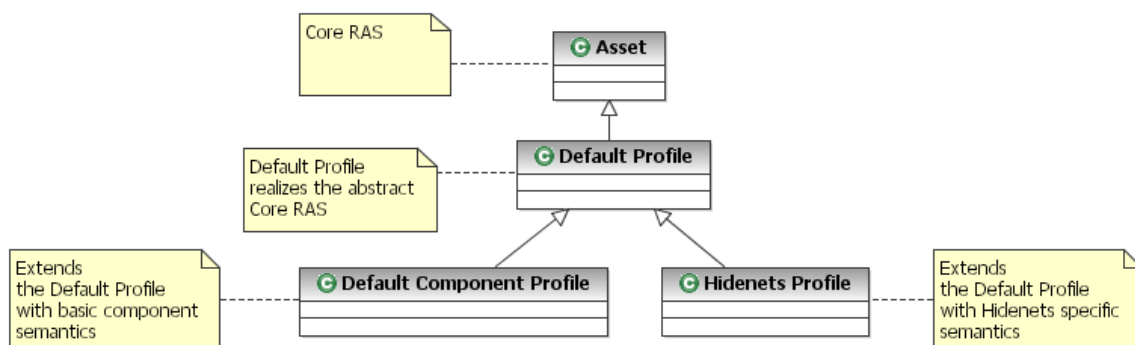


Figure 5: Relation of the Core RAS and Profiles

Assets are defined by building models that use the elements defined in a selected profile. These models are stored in XML files that conform to the XML schema defined by the Core RAS metamodel and contain the extensions of the applied profile. In Figure 6 the Core RAS metamodel is depicted.

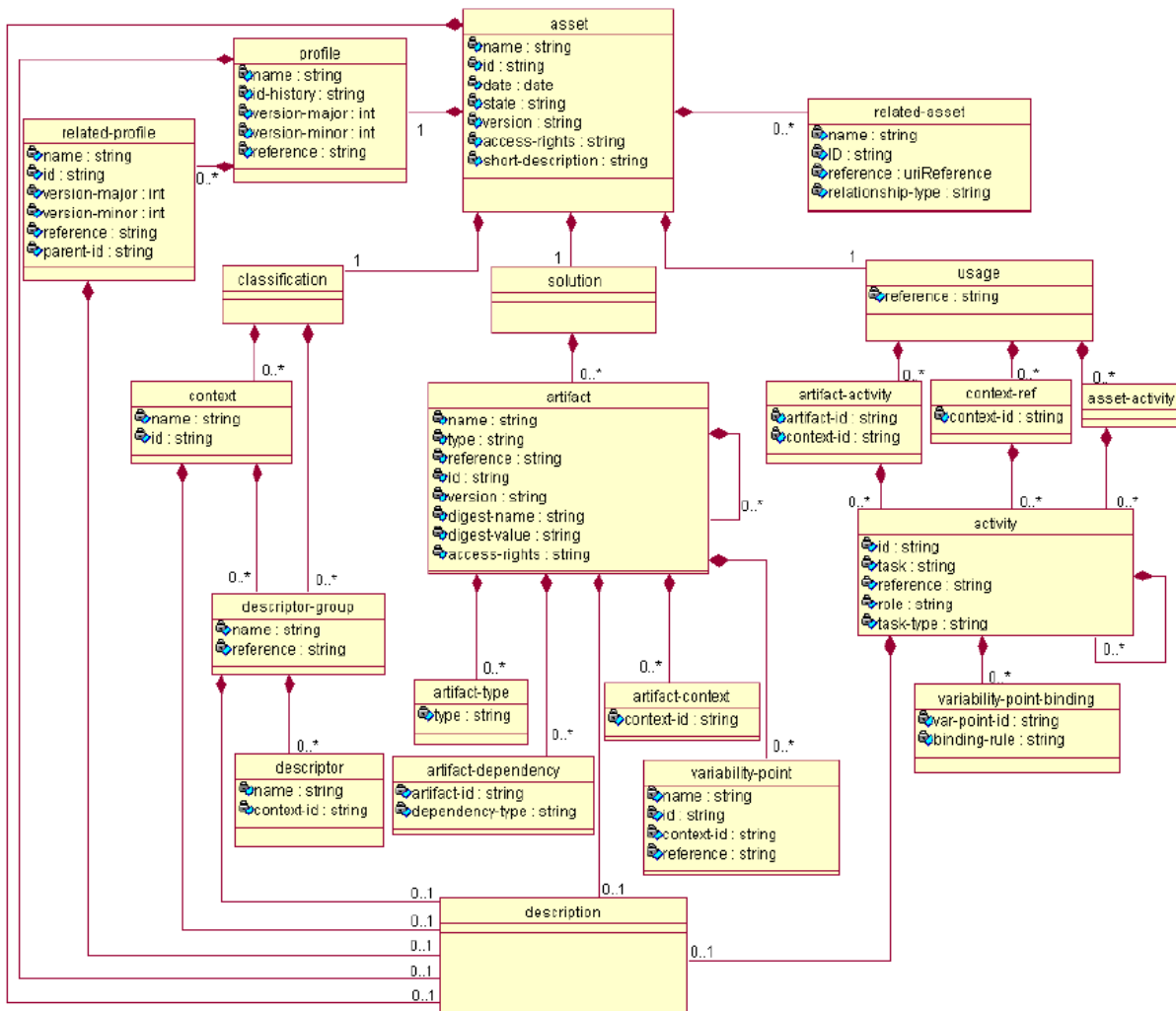


Figure 6: Core RAS metamodel

The root element is the *asset* class. The *asset* is composed of a *profile*, the *classification* of the solution, the *solution* itself and the information related to the *usage* of the asset. The *profile* is a reference to the RAS profile that defines the elements and semantics that the given asset uses, e.g. the Default profile or the Default Component Profile. The asset can be classified from one or more perspectives and this is what the *classification* element is used for. The *solution* contains all the artifacts that are needed for solving a given problem. The *usage* describes the activities to be performed for applying or using the asset. And there may be assets that are somehow related to the described asset. Such assets are identified by the *related-asset* element.

More details of the model can be found in [RAS].

2.1.2 UML 2.0 Testing Profile

The UML 2.0 Testing Profile (U2TP) [U2TP] is an official OMG standard UML extension defining testing related concepts. By defining the common testing notions and providing profile elements for them it can help to specify test scenarios and test cases using the existing design models.

The concepts of the profile are grouped into four groups:

- Test Architecture,

- Test Behaviour,
- Test Data,
- Time.

Test Architecture. The *Test Architecture* group contains concepts to describe the test environment and the role and relationships of each element within. Typical examples include well-known testing concepts like *System Under Test* (SUT), *test component* and *test context*. These concepts and the corresponding stereotypes in the U2TP profile could be used to describe the static structure of the test system. Figure 7 illustrates a component diagram, which describes the testing roles of the components in the given test.

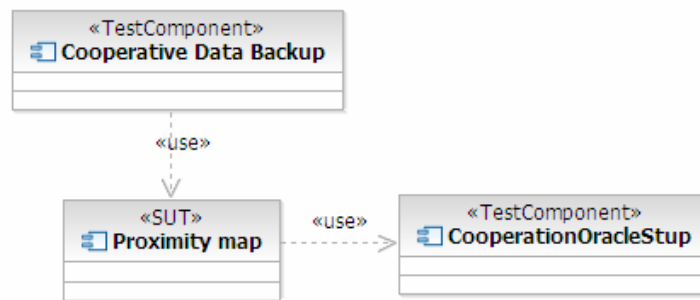


Figure 7: Specifying test environment with UML Testing Profile

Test Behaviour: The *Test Behaviour* group contains elements for describing the test case specifications. The stereotypes included in the profile help express what are the passed and failed branches of the tests or compulsory and optional messages in a communication scenario. By assigning *verdicts* (e.g. pass, fail or error) to *test cases* the outcome of a test can be specified. Figure 8 depicts a case, where the `<<ValidationAction>>` stereotype is used to specify the outcome of the test.

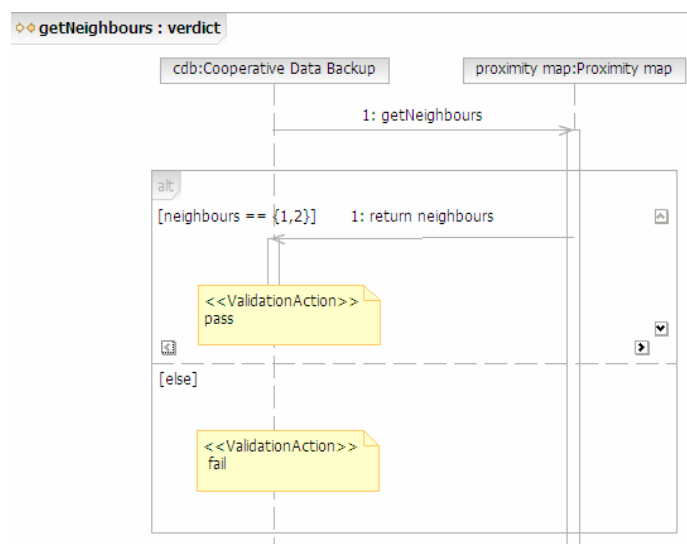


Figure 8: Test case specification using U2TP

Test Data: The Test Data concepts are used to specify the test data used in the test cases. With the help of *Wildcards* and *data pools*, a large number of test data can be given in a compact form.

Time: The Timer concepts contain *timers* that can be used to define timeouts in the test cases. A special element is available for grouping time-synchronized parts of a system, called *timezone*.

Once the test cases have been defined using the elements of the U2TP profile, the test could be executed using the predefined mappings. The concepts in the standard are mapped to two well-known test execution systems, namely JUnit [JUnit] and TTCN-3 [TTCN3]. It should be noted, that the mapping is not complete in some cases, because U2TP defined a more general functionality than the ones supported by these two platforms.

For a more detailed description of the U2TP profile and a case study showing the usage of the different elements see [Dai05].

2.2 Embedded standards

In this section we describe the standards that are available in the field of embedded systems and are subject to use in the HIDENETS project.

2.2.1 SysML

The System Modelling Language (SysML) is a general-purpose modelling language for systems engineering applications. SysML aims at supporting the specification, analysis, design, verification and validation of a broad range of large and complex systems that include hardware and software components.

SysML is being defined by the “SysML Partners” which is an informal partnership of organizations, including large and well-known companies like IBM, Motorola, I-Logix, Genteware, Telelogic, from the field of aerospace and defence: Boeing, Nasa/Jet Propulsion Laboratory, BAE Systems, Northrop Grumman, THALES etc. (the whole list of SysML partners is enumerated on the <http://www.sysml.org> website).

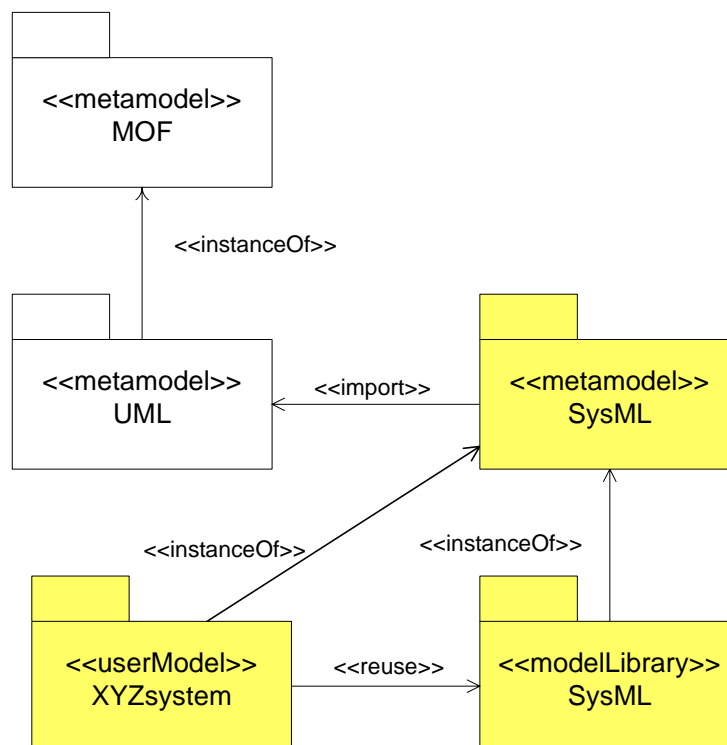


Figure 9: SysML extension of UML

The original aim of the SysML initiative was to customize the Unified Modelling Language (UML) for systems engineering applications. This collaboration resulted in the submission of SysML in response to the OMG's *UML for Systems Engineering Request for Proposal (RFP)* issued in 2003.

The latest available version of SysML was submitted to the OMG in May 2006. It is available for download from the above mentioned website.

SysML is architecturally aligned with the evolving ISO AP-233 data interchange standard for systems engineering in order to support interoperability between systems engineering tools and other tools that interface with them.

Since our aim in the HIDDENETS project is to propose a PIM metamodel, which is able to describe complex, embedded, real-time systems from high-level, system wide view, it was obvious to survey the SysML metamodel from the point of view of embedded system design.

In the following, we give a brief overview of the SysML metamodel, placing emphasis mainly on the embedded system specific aspects of this novel proposal.

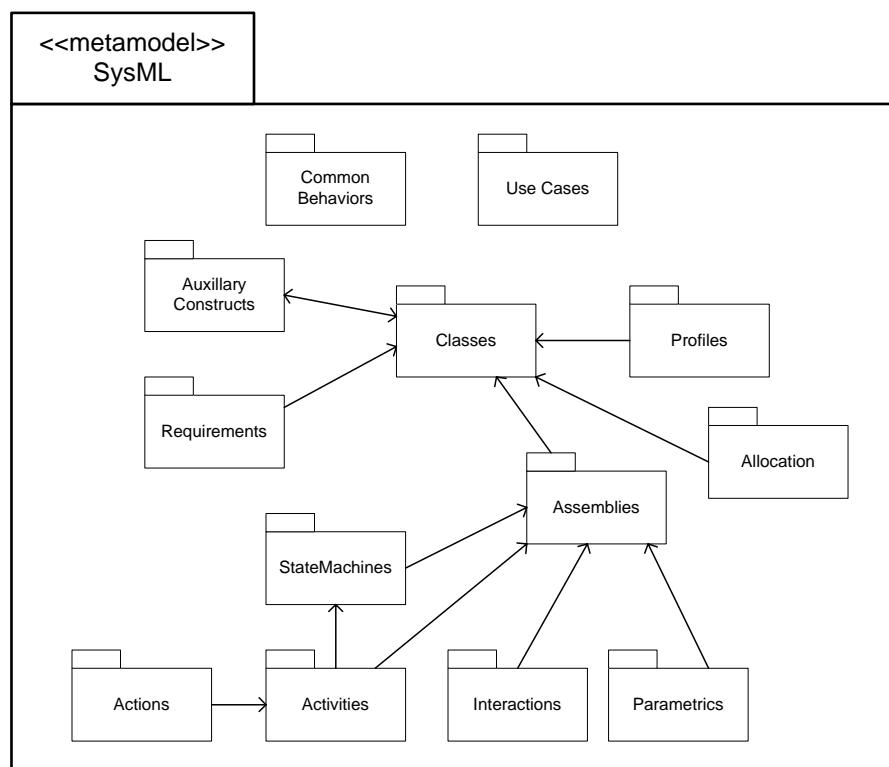


Figure 10: SysML package structure

The SysML metamodel. The proposed SysML specification is defined as an extension of the UML 2.0 Superstructure Specification (see Figure 9), this way reuses the UML syntax and semantics roles.

As shown in Figure 9 SysML reuses a subset of UML (<<IMPORT>> relationship) and creates extensions to support the specific requirements needed to model complex engineering systems. Several extension mechanisms are used including stereotypes, metaclasses and model libraries. The SysML stereotypes and metaclasses are instantiated in the <<USERMODEL>> while classes are grouped into subclasses in the <<MODELLIBRARY>>.

The SysML package structure is shown in more details on the Figure 10. Among these packages there are some imported from the UML without any modification, like State Machines, Interactions and Use Cases.

The major structural extension in SysML is the Block, which extends UML *Structured Class*; it is a general-purpose hierarchical structuring mechanism that abstracts away much of the software-specific detail implicit in UML structured classes. Blocks can represent any level of the system hierarchy, including the top-level system, a subsystem, or logical or physical component of a system or environment. A SysML Block describes a system as a collection of parts, and connections between them that enable communication and other forms of interrelationships.

Parametric Equations Diagrams help the designer cope with complex mathematical and logical expressions as well as constraints. The parametric model can be expressed in a specific language, such as MathML or a programming language.

New SysML packages are Requirements, Parametrics and Allocation. The Assemblies package uses structured classes from COMPOSITESTRUCTURES (UML2.0) and contains some minor extensions.

The Requirements package supports describing textual requirements of a system and their relations to the specification, analysis models, design models etc. A requirement can be assigned to a model element that is intended to realize or satisfy the requirement by the *<trace>* and *<assignment>* UML relationships and this can be further grouped for various purposes using a dependency group. The requirements model can be shown in graphical, tree structure, or tabular format.

The Requirements package definitely should be considered when describing the requirements of applications in the context of the HIDENETS project.

2.2.2 UML profile for schedulability, performance, and time

The “UML profile for schedulability, performance, and time” (SPT) [SPT] profile was intended to provide a single unifying framework to support the existing analysis methods and still provide support for different specializations. SPT supports schedulability analysis based on schedulability theory and performance analysis based on queuing theory. By introducing quantitative information into a UML model, the model can be analysed to predict crucial time-related characteristics well in advance of committing to costly implementation decisions.

The SPT profile has a modularized structure so future extensions can be easily integrated. It is partitioned into a number of sub-profiles as illustrated in Figure 11. The *General Resource Model (GRM)* is at the core of the SPT profile. It contains three sub-profiles *RTresourceModelling*, *RTconcurrencyModelling* and *RTtimeModelling* for dealing with concepts like resource modelling, concurrency and time-modelling. The various types of analysis-specific models are defined in the Analysis Models package of the overall profile. This package contains three specific sub-profiles also. The *PAProfile* incorporates a set of general concepts used in performance analysis. The *SAProfile* package specializes the concepts of the concurrency modelling and time modelling sub-profiles enabling schedulability analysis on the model while the *RSAProfile* package provides schedulability analysis support for the real-time CORBA middleware standard. Finally the *RealTimeCORBAModel* model library captures the essentials of the real-time CORBA technology. In the future different vendors can add their own pre-built models for supporting specific technologies.

SPT offers also some primitives for modelling resources (Resource, ResourceUsage, ResourceManager, etc), scheduling (Schedule, Scheduling Policy, Schedulable Resource, etc). However these concepts are abstract, and in order to be used in real projects, they need to be specialized into a real-time framework.

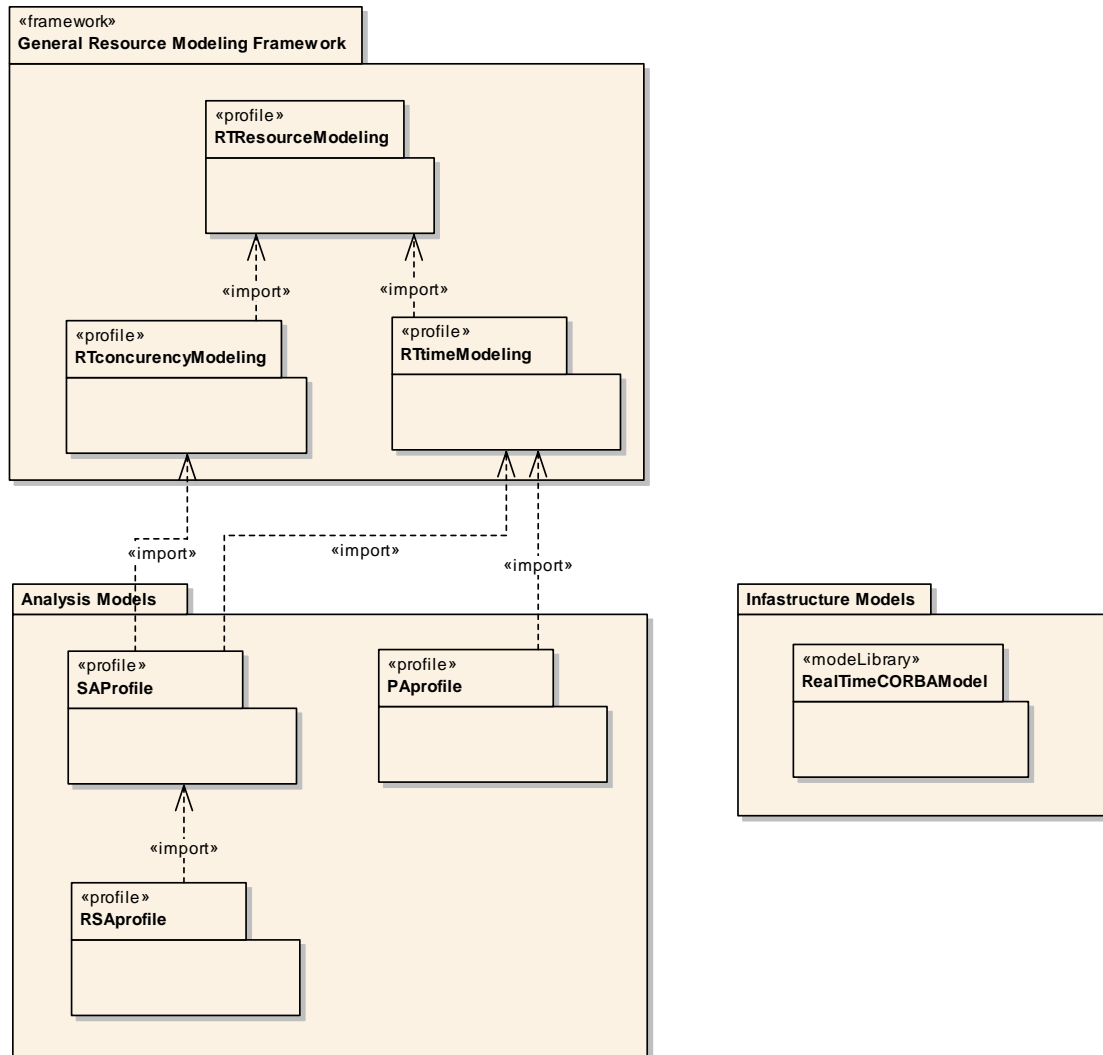


Figure 11: The sub-profiles of SPT profile

2.2.3 Profiles for mobile systems

According to our research, currently there is no standard for modelling mobile systems yet, but in the recent years several approaches emerged. The following is a brief summary of the recommended methods.

A number of publications focus on mobile agents from the broad area of mobile systems. An agent is a software component that executes specific tasks on behalf of someone with some autonomy [Belloni04]. A mobile agent can move between systems to accomplish its goal with transferring its code to another computing environment.

Mobile Agent Modelling with UML (MAM-UML) is a UML profile recommended by Belloni and Marcos in [Belloni04]. The stereotypes and tagged values of the profile are organized into views that describe the different aspects of the mobile agent. The appendix of the paper lists all defined elements; here only a subset of them is highlighted.

- The Organizational view consists of Class or Object diagrams where the mobility related entities are marked, e.g. with the following stereotypes: *mobile-agent*, *resource*, *place* and *home*.
- The Life-cycle view uses Interaction diagrams or Statecharts to describe the dynamic behaviour of agents like how they clone their code and states during movement.
- The Interaction view illustrates how agents and mobile-agent systems communicate, e.g. the *synchronicity* or the *locality* of the interaction.
- Finally, the mobility view is a tagged Deployment or Component diagram to express the details of code transfers (*protocol*, *direction*, etc.).

Another approach is OMG's **MASIF**, and later Mobile Agent Facility (MAF) [MAF], was a specification for agent environments. It defined common platform elements (*Region*, *AgentSystem*, *CoreAgency*, *Place* and *Agent*) for the interoperability of platforms supporting the execution of mobile agents. The MASIF-DESIGN UML profile was proposed in [Muscutariu01] to support the description of MASIF compliant platforms. Key concepts in MASIF are distribution transparencies, e.g. access or location transparencies. These are supported in the profile with tagged values. However, we are not aware of any recent activity about MASIF or MAF, and the well-cited MASIF compliant platform (see [grasshopper]) does not seem to be developed any further.

Grassi et al. proposed an UML profile to support physical mobility of the computing nodes and the logical mobility of software elements [Grassi04]. The following stereotypes were included in their profile to describe the structural aspects: *Place*, *NodeLocation*, *MobileElement*. The behaviour of mobility is expressed on mobility manager statecharts with the *MoveActivity* stereotype. The paper includes examples to show how the profile can be applied to describe basic mobile code paradigms (e.g. Code on Demand, Mobile Agent).

In [Baumeister03] a UML extension called Mobile UML was proposed to model mobile systems in global computing. The extensions consist of (i) a UML profile to express mobility concepts (*location*, *mobile*, *mobile location*) and (ii) new diagram types. According to the authors the problem with UML Sequence diagram when modelling mobile scenarios is that movement of an entity can be expressed only indirectly by adding a new object box. Thus, to overcome the complexity of this approach, a new diagram type Sequence Diagram for Mobility (SDM) was recommended. In SDM UML diagrams are extended (i) with stereotypes like *move*, *copy* and *become* to express movement of objects into new locations and (ii) with new drawing elements representing e.g. that objects move into a specific location. Extensions for Activity Diagrams and with the help of the STP profile the analysis of performance are also proposed.

The Architectures for Mobility (AGILE) project [AGILE] researched architectural approaches to model mobility.

The project proposed (i) the KLAIM language (Kernel Language for Agent Interaction and Mobility) in order to support software engineering by the Mobile UML extension mentioned above and (ii) the MTLA language (Mobile Temporal Logic of Actions) for expression of properties. The project developed several tools (e.g., ArgoMobile), available from the project's website.

Context-aware applications are applications that can adapt their functionality according to their environment (e.g. location, time, persons nearby). There are several middleware systems supporting the development of such applications. In [Ayed05] a UML profile was proposed for designing context-aware applications. The profile uses stereotypes *Context*, *CollectionProcess*, etc. to specify what kind of context information is used and how it is obtained. Other stereotypes support describing the adaptation process based on the current context, e.g. *VariableStructure* and *Version*.

In [Acharya03] a formalism called Mobicharts was proposed for mobile computing (MC) environments. In mobile computing central nodes, called Mobile Support Stations, manage Mobile Hosts in geographic locations, called cells. Because Objectcharts (which are variants of Harel's Statecharts) turned out to be inadequate to model such environments, an extension called Mobicharts was proposed. The extension contains specific states to model e.g. the situation when the mobile host is disconnected and mechanisms to express task migration.

As it can be seen from this short review, the area is under heavy research. Several approaches have been proposed, that contain many similar elements, however, each of them are specialised for a specific aspect of mobile systems, and no general standard is available at the moment. The HIDENETS environment focuses more on physical mobility in ad-hoc networks, thus the above recommendations could not be simply integrated, but they can serve as a starting point for identifying the concepts that need to be captured by the HIDENETS UML profile.

2.2.4 AUTOSAR

The AUTomotive Open System Architecture (AUTOSAR) [AUTOSAR] was founded in 2003 by major automotive Original Equipment Manufacturers (OEM) and their Tier 1 (direct) suppliers (for braking systems, semiconductor manufacturers, etc.) and now includes a large number of automotive, electronics, semiconductor and software companies. The partnership has adopted a three-tier membership structure which currently consists of 10 Core Partners, 45 Premium Members and 15 Associate Members (the whole list of members can be found on the www.autosar.org website)

AUTOSAR aims at the development and establishment of a common open industry standard for an automotive E/E (electrical/electronic) architecture, facilitating the reuse of software components between different vehicle platforms, OEMs and suppliers. To achieve this, AUTOSAR defines a methodology that supports a distributed, function-driven development process and standardizes the software architecture for each Electronic Control Unit (ECU) in such a system. AUTOSAR also specifies compatible software interfaces at the application level. The abstraction encourages effectively the usage of "commercial off the shelf" hardware.

The standardization would provide benefits like reduced system complexity, better product upgrade/update support, improved quality and reliability of E/E systems, and it shall enable detection of errors in early development phases.

2.2.4.1 Main AUTOSAR Concepts

The AUTOSAR standards deal with the following basic concepts:

- The AUTOSAR Software Component (AUTOSAR SW-C) encapsulates an application that runs on the AUTOSAR infrastructure.
- Software Component Description (SW-C Description) provides a standard description format for the integration of AUTOSAR Components.

- The Virtual Function Bus (VFB) provides standardized communication mechanisms and services for SW-Cs and it is provided by AUTOSAR on an abstract, technology independent level.
- System Constraints and ECU Descriptions provide resource descriptions for the complete system as well as for configuration of the single Electrical Control Units (ECUs).
- The VFB functionality on a specific ECU is implemented by the Run Time Environment (RTE).
- Basic Software provides services to the AUTOSAR Software Components and it is necessary to run the functional part of the software.

2.2.4.2 The AUTOSAR Methodology

The AUTOSAR Methodology is a common technical approach in system development from the system-level configuration to the ECU executable generation. Basically the methodology defines the activities on work-products. The Software Process Engineering Meta-model (SPEM) by OMG is used for describing the methodology, and only a small subset of SPEM is actually used.

The system development starts with the definition of the system configuration input which includes the following:

- *Software Component Description* which defines all the SW-Components.
- *ECU Resource Description* which describes the available hardware resources.
- *System Constraint Description* which includes the network architecture constraints.

Next, the software components are mapped to the ECUs. The result of this activity is the *System Configuration Description*.

Finally, the executables are generated for each individual ECU.

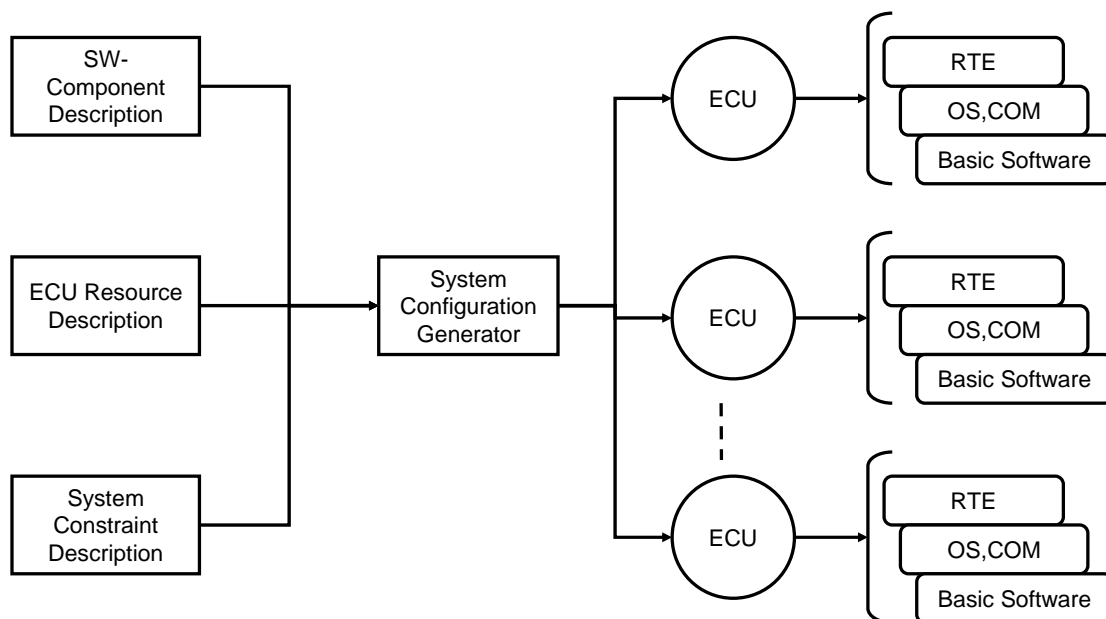


Figure 12: AUTOSAR methodology

In Figure 12 the data elements of the AUTOSAR system development methodology are shown.

2.2.4.3 The AUTOSAR metamodel and UML profile

The original AUTOSAR metamodel is created as an extension of the UML 2.0 metamodel and the Essential MOF (EMOF) metamodels. In Figure 13 the hierarchy of the AUTOSAR packages is depicted. The different packages are placed to different levels in the hierarchy from M3 to M0. M0 is the level of the user objects while M3 contains the metamodels and profiles that serve as the base for the hierarchy. The AUTOSAR metamodel exists at level M2. The metamodel itself is an instance of the MOF 2.0 metamodel, and it is instantiated as a direct MOF instance. Figure 13 shows the hierarchy of the AUTOSAR Metamodel. A specific UML profile (AUTOSAR UML Profile) is also included in the metamodel to capture the specific requirements of automotive applications. Moreover, this UML profile enables the development of AUTOSAR models in standard UML tools.

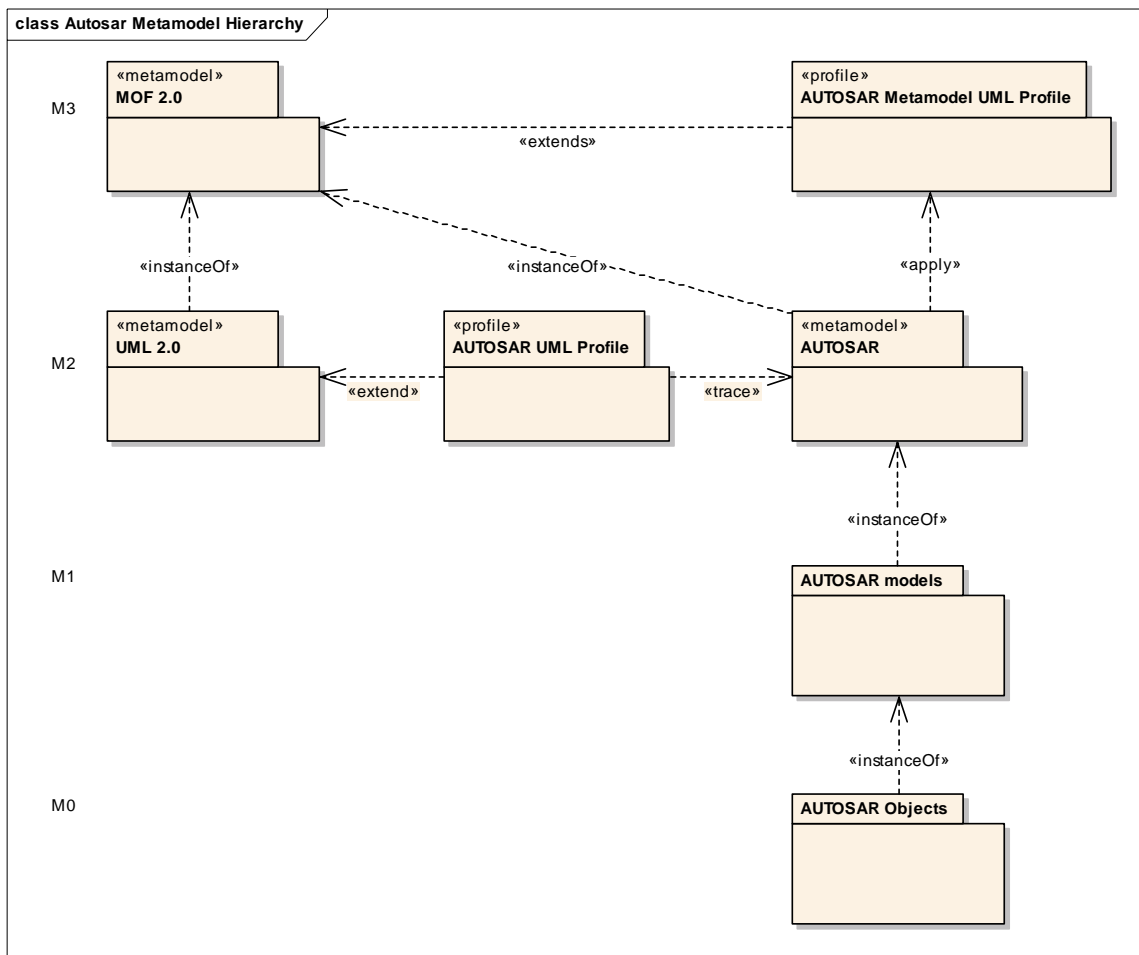


Figure 13: AUTOSAR metamodel hierarchy

2.3 Dependability standards

In this section we present the standards that are available in the field of dependability and are subject to use in the HIDENETS project.

2.3.1 Fault Tolerant UML profile

In 2006 OMG released a UML Profile to describe Quality of Service (QoS) characteristics and fault tolerant mechanisms, the “UML Profile for Modelling QoS and FT Characteristics and Mechanisms” profile [QoSFT]. The profile consists of three sub profiles: QoS Framework, Risk Assessment and Fault Tolerance (FT). This section gives a short summary of the QoS and FT parts, and highlights their relevance to the Hidenets project.

The QoS Framework of the profile helps to specify the non-functional requirements of a system. The profile contains the modelling elements that incorporate QoS aspects in a UML model; it does not restrict the actual QoS characteristics to be used. The content of the profile is grouped into three packages. The following description lists the most important elements of each package.

QoSCharacteristics: A *QoSCharacteristic* represents quantifiable characteristics of services; typical examples are latency, throughput or reliability. To each QoSCharacteristic multiple *QoSDimension* could be assigned, which specify, how the given characteristic can be quantified. These can be specified in various formats, e.g. maximum and minimum or statistical values. For example reliability can be characterized by the number of failures tolerated or by the mean time to repair. When a large number of QoSCharacteristics should be considered, they can be grouped into *QoSCategories*, well-known examples include performance or dependability.

QoSConstraints: A *QoSConstraint* limits the allowed values of one or more QoSCharacteristics. Possible types of limitations include enumeration of allowed values or specifying complex expressions. The constraints are either *QoSRequired* or *QoSOffered* constraints. Offered and required constraints can be grouped into *QoSContracts*, formalizing the QoS aspects of a component.

QoSLevels: *QoSLevel* represents the different modes of QoS that a subsystem can support. Depending on the current state of the system, e.g. available resources, the system can offer different levels for its services, which are specified in QoSLevel values.

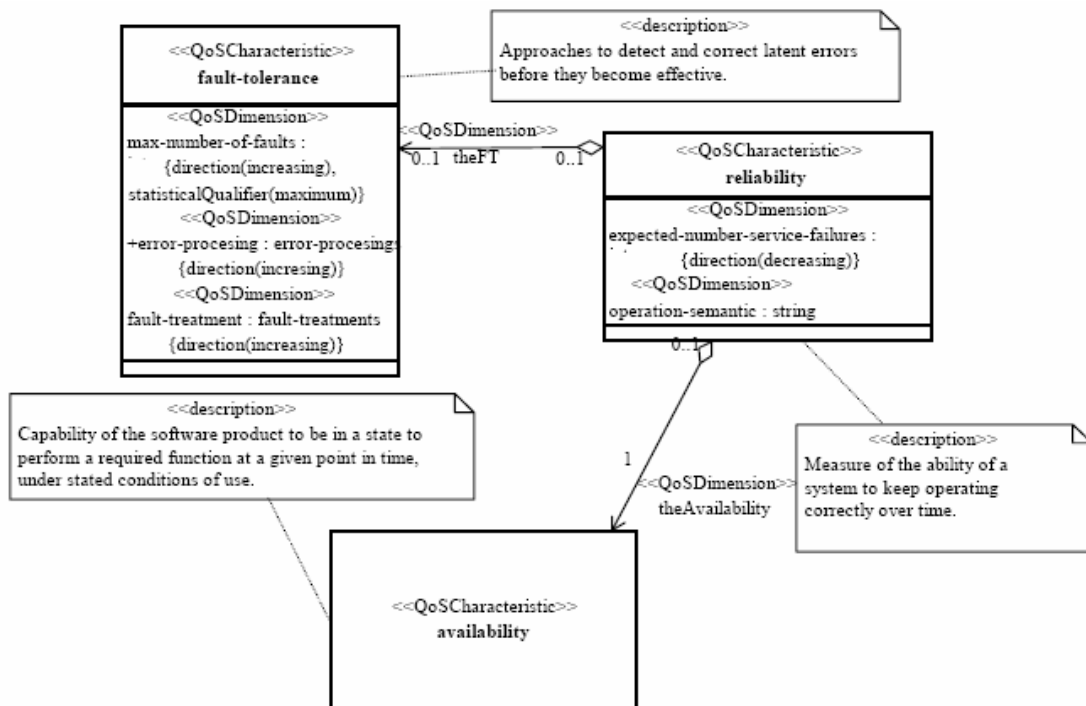


Figure 14: QoS characteristics in the dependability QoS category [QoSFT]

Most of the QoS characteristics are common to many systems, thus to help reuse, common QoS catalogs can be created. The specification includes a general QoS catalog, with categories like performance, dependability, security and coherence. The formulation of the catalog is based on international standards, the profile includes an UML-based unified notation of the concepts.

Figure 14 illustrates the dependability category included in the specification with the specified characteristics and their dimensions.

The profile does not include elements for specifying resource types and usage, for which the QoS values can be defined. These elements can be found in the SPT profile [SPT], the QoS specification includes an example how these two profiles can be combined.

Hidenets relevance: Several blocks in the Hidenets architecture deal with QoS values, e.g. QoS Coverage Manager. When applications use these blocks the QoS parameters could be specified using the notation offered by the specification.

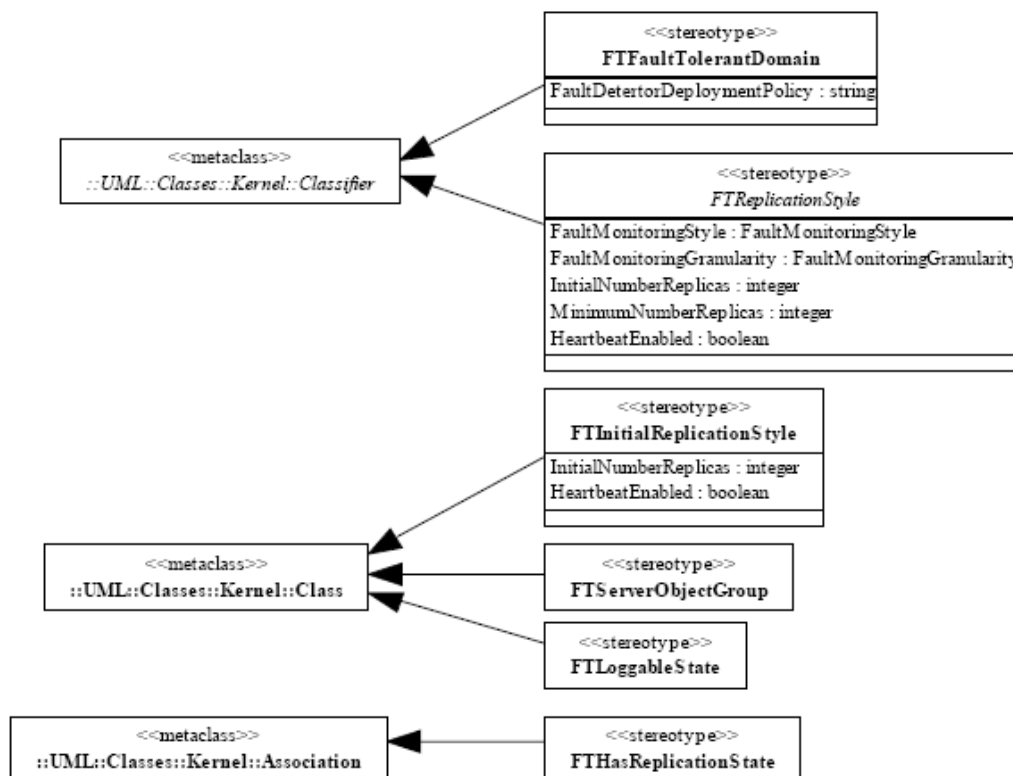


Figure: 15 Core FT Profile [QoSFT]

The second part of the profile defines extensions to model fault-tolerant software architectures. From the broad mechanisms for fault-tolerance, the profile focuses on safety mitigation means, especially replication. The profile includes the following four metamodels.

Fault Tolerant Core: The core metamodel includes definition for basic fault tolerant concepts like server groups and replicas.

Fault Detection: This package defines the different methods for carrying out fault detection, e.g. whether the fault detector is managed by the infrastructure or the application itself.

Object Group Properties: These elements define the fault tolerance policies used in the system. Typical policies include the style of fault detection (pull or push) or the controlling of consistency (managed by the infrastructure of the application).

Replication Styles: One of the most important properties of a replication-based FT structure is the replication style. Possible values include cold or warm standby passive or active replication, etc.

The stereotypes defined by the core profile are summarized on Figure: 15.

Hidenets relevance: The concepts of the profile could be useful for the fault tolerance manager, especially when modelling the Replication Manager.

2.3.2 Service Availability Forum - Application Interface Specification Metamodel

The *Service Availability™ Forum (SAF)* [SAF] aims at providing standardized solutions for making services highly available. The Application Interface Specification (AIS) of the Forum defines the standard interfaces for accessing Highly Available (HA) middleware and infrastructure services that reside logically between applications and the operating system. All AIS services are designed to operate in a cluster environment where computation nodes are connected to each other and work together for a specific goal. The interfaces are defined in the form of sub-specifications, namely:

- **Availability Management Framework (AMF)**

The Availability Management Framework is a general approach for high availability needs in environments which run redundant components. Its goal is to ensure application availability by detecting component failures and shifting service load from failed components to sane components.

- **Checkpoint Service (CKPT)**

A *checkpoint* contains application-specific data partitioned in *sections*. It is a cluster-wide entity designated by a unique name. A checkpoint is made highly available by *replication*. The Checkpoint Service is responsible for the handling and the replication of checkpoints. The application component requests the creation, the update and the deletion of checkpoints which are replicated according to configured or application-defined rules.

- **Cluster Membership Service (CLM)**

The Cluster Membership Service provides information about the current cluster configuration and the nodes that are members of the cluster. The cluster consists of a set of configured nodes.

- **Event Service (EVT)**

The Event Service permits an M:N communication between event *publishers* and event *subscribers*. It supports the distribution of information (by the publishers) to a set of “interested” applications (the subscribers), that can select this information according to specified filter criteria. Communication takes place over *event channels*. Multiple publishers and subscribers can communicate over the same event channel.

- **Information Model Management Service (IMM)**

The Information Model Management Service provides the information base of all objects handled by services attached to it. It is intended as a repository especially for the SAF

services, but not restricted to them. It keeps information about various objects belonging to the attached services, e.g. the configuration and runtime attributes of service units, checkpoints, message queues, etc.

- **Lock Service (LCK)**

The Lock Service provides cluster-wide *lock resources* and the ability to set or release locks on them. Locks are used to synchronize accesses from competing processes or nodes to shared resources.

- **Logging Service (LOG)**

The Log Service provides interfaces through which applications can act as *loggers*. They can log events of different categories (alarms, notifications, system information, application information) into cluster-wide resources maintained by the Log Service, the *log streams*.

- **Message Service (MSG)**

The Message Service provides *queues* and *queue groups* for the m:1 resp. m:n communication between Message Service clients within a cluster.

- **Naming Service (NAM)**

The Naming Service provides the storage and the retrieval of named objects.

- **Notification Service (NTF)**

The Notification Service provides APIs to work with *notifications* in a *producer*, *subscriber* or *reader* role. A notification is a data structure that describes an important event (in its natural meaning, not in the one of the SAF Event Service) during the lifetime of an HA cluster.

- **Timer Service (TMR)**

The Timer Service provides a mechanism by which client processes get notified when a *timer* expires. A timer is a logical object that is dynamically created and represents either absolute time or duration (i.e. an interval relative to a time reference point).

For more detailed description of these services see the corresponding section in [D2.1].

The SAF Information Model. The entities defined in the AIS specifications (e.g. service units, message queues, applications, etc.) are described semi-formally by the SAF Information Model (IM) in the form of UML classes.

From the perspective of the HIDENETS project the most relevant entities are the ones defined by the Availability Management Framework specification. In the following we introduce the metamodel of the AMF entities.

The AMF is the software entity that provides service availability by coordinating redundant resources within a cluster to deliver a system with no single point of failure. The AMF defines two types of entities, the *physical* and the *logical entities*.

Physical entities. Every *physical entity* managed by the Availability Management Framework is a *resource*. These physical entities are either hardware equipment or software abstractions implemented by programs running on that hardware. In Figure 16 the hierarchy of physical entities

is depicted. The **Resource** class represents the resource abstraction and it is also the base class for the specialized resource entities. These entities are the following:

- the **Physical Node**, which represents a computer with an operating system;
- the **Local Resource**, which represents a local resource from fault containment point of view, so that if the host physical node fails all of the hosted local resources become inoperable;
- all other resources are called **External Resources**, and failures of external resources are independent of physical node failures.

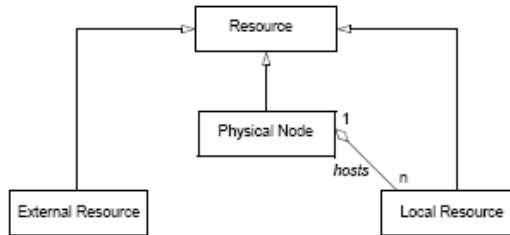


Figure 16: Physical entities of the Availability Management Framework

Logical entities are the abstract bricks of a high availability service. Each level of the service can be represented by a logical entity. The *component* is the smallest logical entity on which the Availability Management Framework performs any action, and the *application* represents the highest level of the service. The hierarchy of the AMF logical entities is depicted in Figure 17.

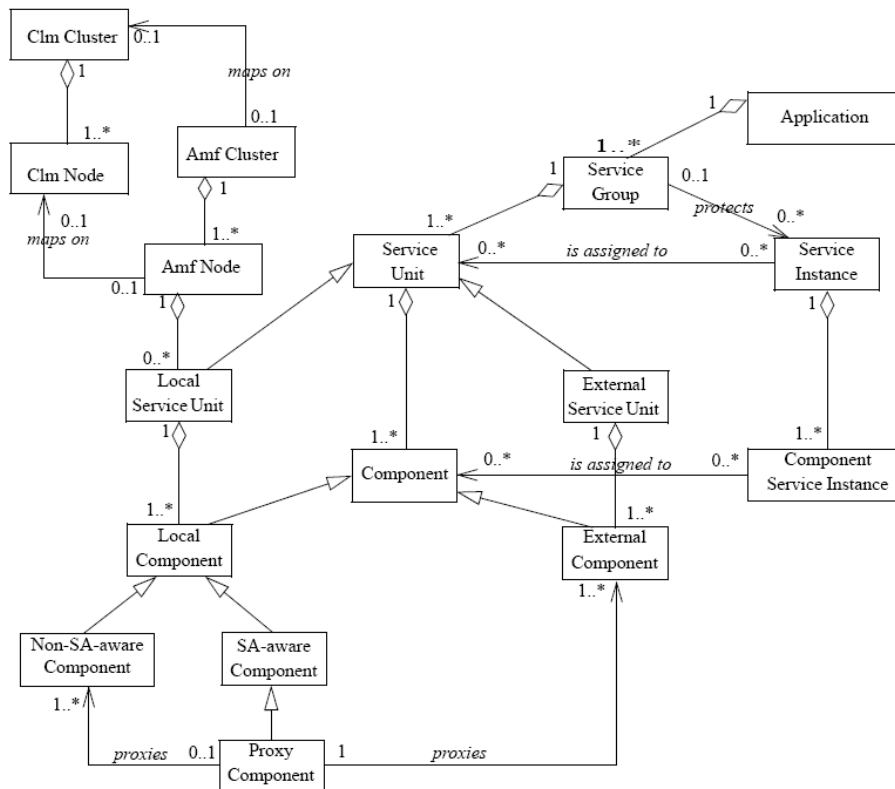


Figure 17: The logical entities of the Availability Management Framework

The **Component** represents a set of resources to the Availability Management Framework. The resources represented by the component encapsulate specific application functionality. This set can

include hardware resources, software resources or a combination of the two. It is the smallest logical entity on which the Application Management Framework performs error detection, isolation, recovery, and repair. The scope of the component must be small enough so that its failure has as little impact as possible on the services provided by the cluster. Furthermore, the component should include all the important functions that cannot be separated.

The Component class is an aggregated notion that may refer to either a *Local* or an *External Component*. A **Local Component** represents a subset of the local resources contained within a single node while an **External Component** represents a set of resources that are external to the AMF cluster.

The *Local Component* class is further specialized into **SA-Aware** and **Non-SA-Aware Component** classes that refer to whether the given *Local Component* implements the interfaces that enable the AMF to monitor the health of the component or not. In case of *Non-SA-Aware Components* and *External Components* a **Proxy Component**, which is a special *SA-Aware Component*, has to be used that uses proprietary communication methods to forward the health check requests of the AMF to the designated components.

A **Service Unit** is a logical entity that aggregates a set of *Components* combining their individual functionalities to provide a higher level service. A *Service Unit* can contain any number of *Components* but a given *Component* can be configured in only one *Service Unit*. Using this model, the *Components* can be developed in isolation, and the developer might be unaware of which *Components* constitute a *Service Unit* since they are defined at deployment time only.

From the perspective of the Availability Management Framework the *Service Unit* is the unit of the redundancy so that it is the smallest logical entity that can be instantiated in a redundant manner.

Service Units are aggregated into **Service Groups**. The *Service Group* prescribes the manner in which the *Service Units* are instantiated in order to make the individual services that are provided by the *Service Units* highly available. This manner is called the redundancy model. The redundancy model of the *Service Group* can be for example the “2N”, which means a simple failover behaviour, or the “N+M” model for N active, M standby behaviour.

As mentioned above, the **Application** represents the highest level of the service, which is provided by the cluster. It contains one or more *Service Groups* and combines the individual functionalities of the constituent *Service Groups* to provide the higher level service. An *Application* can contain any number of *Service Groups*, but a given *Service Group* can be configured in only one *Application*.

A **Component Service Instance** represents the workload that the Availability Management Framework can dynamically assign to a *Component*. High availability (HA) states are assigned to a *Component* on behalf of its *Component Service Instances*.

Each *Component Service Instance* has a set of attributes (name/value pair), which characterize the workload assigned to the *Component*. These attributes are not used by the Availability Management Framework, and are just passed to the *Components*. The Availability Management Framework supports the notion of *Component Service Instance Type*. All *Component Service Instances* of the same type share the same list of attribute names.

In the same way as *Components* are aggregated into *Service Units*, the Availability Management Framework supports the aggregation of *Component Service Instances* into a logical entity called a **Service Instance**. A *Service Instance* aggregates all *Component Service Instances* to be assigned to

the individual *Components* of the *Service Unit* in order for the *Service Unit* to provide a particular service.

When a *Service Unit* is available to provide service, the Availability Management Framework can assign HA states to the *Service Unit* for one or more *Service Instances*. When a *Service Unit* becomes unavailable to provide service, the Availability Management Framework removes all *Service Instances* from the *Service Unit*. A *Service Unit* might be available to provide service but not have any assigned *Service Instance*.

The Availability Management Framework assigns a *Service Instance* to a *Service Unit* programmatically by assigning each individual *Component Service Instance* of the *Service Instance* to a specific *Component* within the *Service Unit*.

AMF Cluster and Node entities. An **AMF Node** is the logical representation of a *Physical Node* that has been administratively configured in the Availability Management Framework configuration. An *AMF Node* is also a logical entity whose various states are managed by the Availability Management Framework using designated administrative operations that are defined for such nodes.

The complete set of *AMF Nodes* in the Availability Management Framework configuration defines the **AMF Cluster**. The *AMF Cluster* is one of the entities that are under the Availability Management Framework control, and its various states are managed by the Availability Management Framework. There are Availability Management Framework administrative operations that are defined on the *AMF Cluster*.

The restart of an *AMF Node* will only stop and start entities under Availability Management Framework control, without any impact on the cluster membership. The restart of the *AMF Cluster* will restart all *AMF Nodes*.

Applications to be made highly available are supposed to be configured in the Availability Management Framework configuration. Each *Application* is configured to be hosted in one or more *AMF Nodes* within the *AMF Cluster*.

One or more *Service Units* can be assigned to an *AMF Node* to provide service.

2.4 Evaluation

In this section we summarize the usability and compliance of the previously mentioned standards and specifications with the objectives of the HIDENETS project.

2.4.1 Reusable Asset Specification

Compliance to HIDENETS objectives: Reusability of solutions is essential for reducing development time and number of faults introduced by re-elaboration of existing solutions. The Reusable Asset Specification provides a framework that facilitates the storage of solutions for given application domains in a well-structured form. In HIDENETS RAS packages will be used for facilitating the application of elaborated solutions during application development. Furthermore, HIDENETS applications may be deployed in the form of RAS packages.

Different domains may use the Default Profile for asset definition or may develop specific profiles. For HIDENETS, both cases are possible. The usage of the Default Profile provides generality, while the specific profile supports the storage of the additional HIDENETS related aspects as well.

Industrial relevance: The Reusable Asset Specification is a standard of the OMG group and provides solutions for problems that mean everyday hardness for businesses. Furthermore, it makes use of XML technology that is one of the most widely used formalisms for structured data storage.

Extensibility: Since the RAS specification was developed by considering extensibility as a primary feature there are well-defined methods for customizing the basic structure and relations.

Extension requirements: For the HIDENETS specific use of RAS assets a HIDENETS specific profile has to be developed that extends the RAS *Default Profile* and refines the meaning of the basic elements. E.g. it has to provide ways to distinguish the components of an application and the service implementations of third party vendors.

Tool support: There are more and more tools that support the storage of solutions in a centralised architecture, e.g. the Rational Software Architect of IBM or LogicLibrary Logidex.

Openness: The RAS standard itself is public and freely accessible, however, the currently known tools are not free.

2.4.2 UML 2.0 Testing profile

Compliance to HIDENETS objectives: The profile can be relevant to the testing activity in WP5 when the test environments and scenarios have to be specified. However, UML may not be the appropriate formalism for testing communication protocols and complex scenarios like the ones in HIDENETS environment. For this reason, this profile will be useful for describing the static structure of the testing components.

Industrial relevance: The profile was developed by industrial partners and can be mapped to the two most well-known test execution frameworks, TTCN-3 and JUnit.

Extensibility: The profile can be extended through standardized way.

Extension requirements: The elements in the profile intend to describe a fixed test environment; it should be extended to support highly dynamic mobile scenarios.

Tool support: The Eclipse Test & Performance Tools Platform (previously Hyades) supports the profile.

Openness: The profile is published as an OMG specification.

2.4.3 SysML

Compliance to HIDENETS objectives: The system modelling and notational conventions introduced by the standard are possible to be used as is in the HIDENETS metamodel.

Industrial relevance: SysML is used by many industries including automotive, aerospace and defence.

Extensibility: The profile can be extended through the OMG standardization organization.

Extension requirements: No required extensions have been identified for the HIDENETS project yet.

Tool support: Many industrial quality products support the SysML profile, e.g. MagicDraw UML or Enterprise Architect.

Openness: The profile is published as an OMG specification.

2.4.4 UML profile for Schedulability, Performance and Time

Compliance to HIDENETS objectives: The SPT, also known as the Real-time UML profile, introduces notions that are essential for the modelling of real-time systems like HIDENETS. E.g. service usage, concurrent execution, workload or timeout.

Industrial relevance: The SPT profile is often referenced in articles and reports, however, it is not common to use it as is. Usually it serves as a starting point and then it is customized according to the needs of the specific domain.

Extensibility: The SPT profile is defined in the form of UML models and diagrams and this provides high degree of flexibility for it.

Extension requirements: No required extensions have been identified for the HIDENETS project yet.

Tool support: Most UML tools offer methods for applying profiles, however, during our research we have not found any tools that out of the box contain the SPT profile.

Openness: The SPT profile is defined by the OMG and is freely accessible.

2.4.5 Profiles for mobile systems

Compliance to HIDENETS objectives: The area is especially relevant to HIDENETS objectives, mobility is a key concept in HIDENETS. However, most of the profiles are focused on mobile agents and not on mobile nodes.

Industrial relevance: The profiles listed in Section 2.2.3 are proposals from ongoing research works, not industrial ready specifications.

Extensibility: Some of the approaches are defined as UML profiles, in this case they can be easily extended. Others define their own formalism, which is not as easy to modify.

Extension requirements: The profiles can only be used as a starting point when developing the mobility modelling profile for HIDENETS, they cannot be adopted in straight-forward way.

Tool support: For the modelling approaches developed in the Agile project tools have been created. Other approaches are defined as UML profiles, thus they can be included in tools through a standardized way.

Openness: All profiles were created in research projects, and they were published in scientific papers, not in an open specification.

2.4.6 Specifications of AUTOSAR

Compliance to HIDENETS objectives: The objectives of AUTOSAR and the HIDENETS project are complementary considering that HIDENETS defines the higher level services and software components while AUTOSAR provides a framework for the development and deployment of hardware and low level software that directly manipulates the hardware.

Industrial relevance: AUTOSAR is a de-facto standard in the automotive industry.

Extensibility: The models and metamodels of AUTOSAR have been defined in UML, thus, providing high level of extendibility.

Extension requirements: No required extensions have been identified for the HIDENETS project yet.

Tool support: According to the principles of the AUTOSAR consortium an Eclipse based open tool framework initiative was proposed in 2006 in order to support the interchange of models between the different modelling tools and long terms stability.

Openness: The AUTOSAR specifications are freely accessible at the AUTOSAR web site.

2.4.7 Fault tolerant UML profile

Compliance to HIDENETS objectives: The profile defines concepts that are heavily used in HIDENETS services. It offers a standardized way to represent QoS characteristics and fault tolerant mechanisms.

Industrial relevance: The profile is a specification from OMG, the maintainer of the UML-related profiles and UML itself.

Extensibility: It is defined as a UML profile, thus it can be extended in a standardized way.

Extension requirements: The profile includes only the basic fault-tolerant mechanism, but includes support for describing other redundancy strategies, e.g. the ones developed in HIDENETS.

Tool support: As far as we know the profile is not included in tools, however many tools include support to define a profile and apply it to models. The metamodel description can be downloaded from OMG in XMI format.

Openness: Request and issues can be reported to OMG, which are handled in new versions of the specifications.

2.4.8 Specifications of the Service Availability Forum

Compliance to HIDENETS objectives: The Application Interface Specification standardizes the access interfaces to several commonly used services such as the checkpointing or messaging. Such interfaces are required to ensure the portability of the applications between different HIDENETS compliant platforms.

Industrial relevance: SAF specifications are widely accepted in the industry. There are a dynamically growing number of technology adapters and implementers.

Extensibility: The SAF interfaces are designed to be used as is and thus there is no support for customizing them.

Extension requirements: No required extensions have been identified for the HIDENETS project yet.

Tool support: Since the SAF specifications are emerging standards currently there is no wide range of tools that support the development for SAF based systems.

Openness: The SAF specifications are freely accessible and there are several open source implementations e.g. OpenAIS, OpenClovis.

3 The HIDENETS metamodel

This section presents a structural overview on HIDENETS artifacts by organizing the concepts introduced or referenced by the previously published deliverables [D2.1, D3.1] in the form of a *metamodel*. A metamodel consists of *metaclasses* corresponding to key modelling concepts (e.g., authentication service) and *associations* between them indicating containment (primarily in data modelling), usage relations (typically between high-level services and lower-level utility functions), plain navigation possibilities or some kind of logical dependency.

3.1 Introduction

The first step of constructing this metamodel was building a detailed *use case model* of the high-level requirements and the corresponding solution proposals (note that this chapter uses the *use case* notion according to the original UML meaning i.e., not only for top-level concepts like previous HIDENETS deliverables (e.g., “car accident use case” [D1.1 Sec 4.3 CAR ACCIDENT]) but also for lower-level usage modes). First we introduced the *top-level use cases* that were identified at the beginning of the project (car accident, platooning, etc.). Next we represented their *requirements* by other use cases being in “include” relation with the top-level ones. All approaches proposed in the framework of the HIDENETS project aim at providing solutions for some of these requirements thus various research areas and solution proposals (e.g., authentication, cooperative data backup, ad-hoc topology control etc.) correspond to further use cases – these use cases can be organized similarly to the “*architectural hybridization*” figure presented in the previous HIDENETS deliverable 2.1.1 [D2.1]. *Our new structural overview* is aligned similarly with the usual representation of layered architectures: (i) top level use cases correspond to the *application level*; (ii) WP2 and WP3 services (i.e., the ones defined in the second and third work packages of HIDENETS) provide various *services* for the application level running on (iii) an underlying *hardware* layer. This structure is presented in Figure 18.

With respect to the *model structure*, our use cases corresponding to various services (WP2 oracles and high-level services and WP3 services) are grouped into three *use case packages*:

- Simple trusted services of WP2 (also called as resiliency kernel services) are collected into metaclass package *Services::Use cases corresponding to WP2 oracles*.
- Complex, less-trusted middleware services of WP2 are collected into metaclass package *Services::Use cases corresponding to WP2 high-level services*.
- Services of WP3 are collected into metaclass package *Services::Use cases corresponding to WP3 services*.

This structure of use case packages is shown in Figure 19.

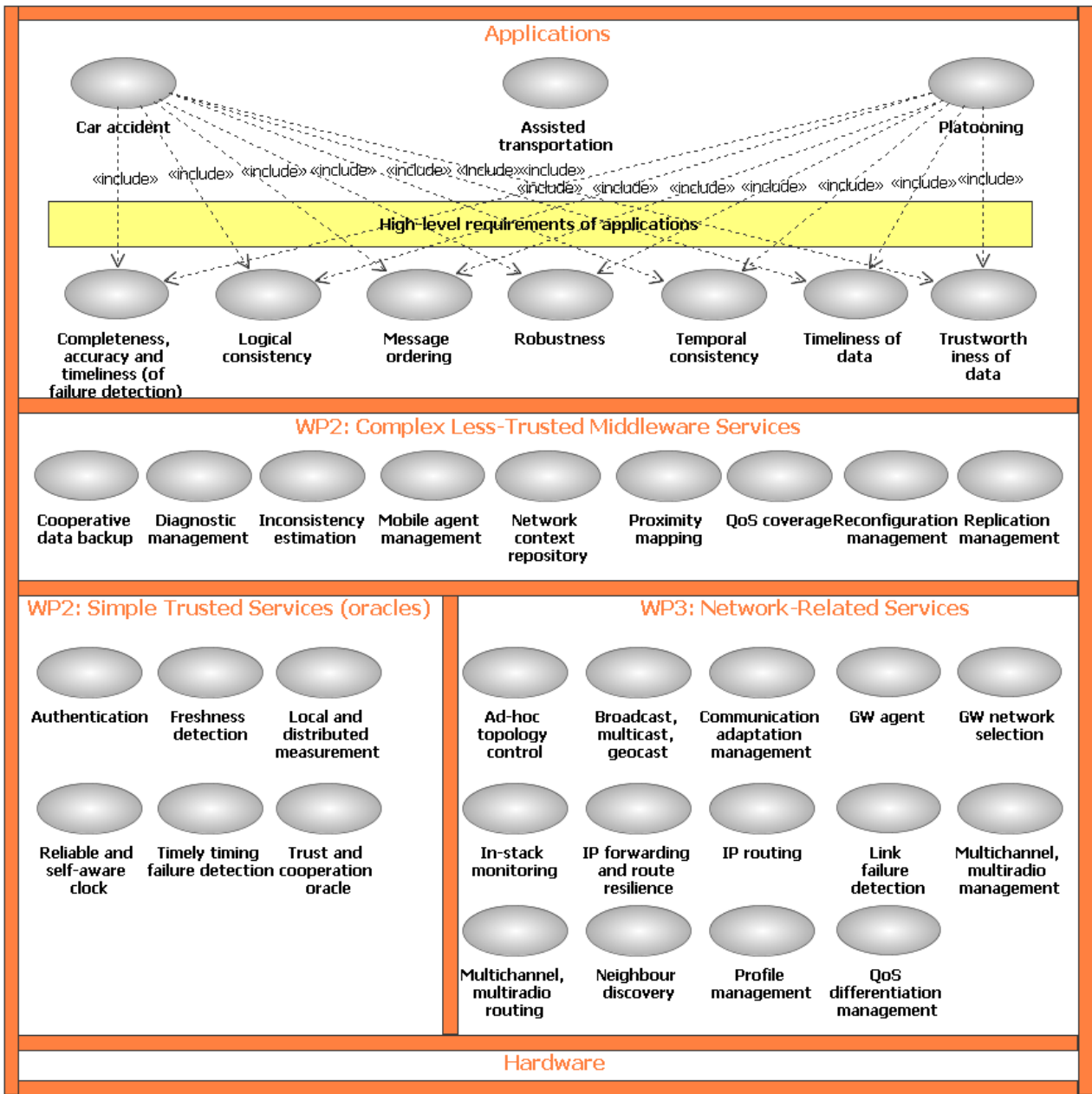
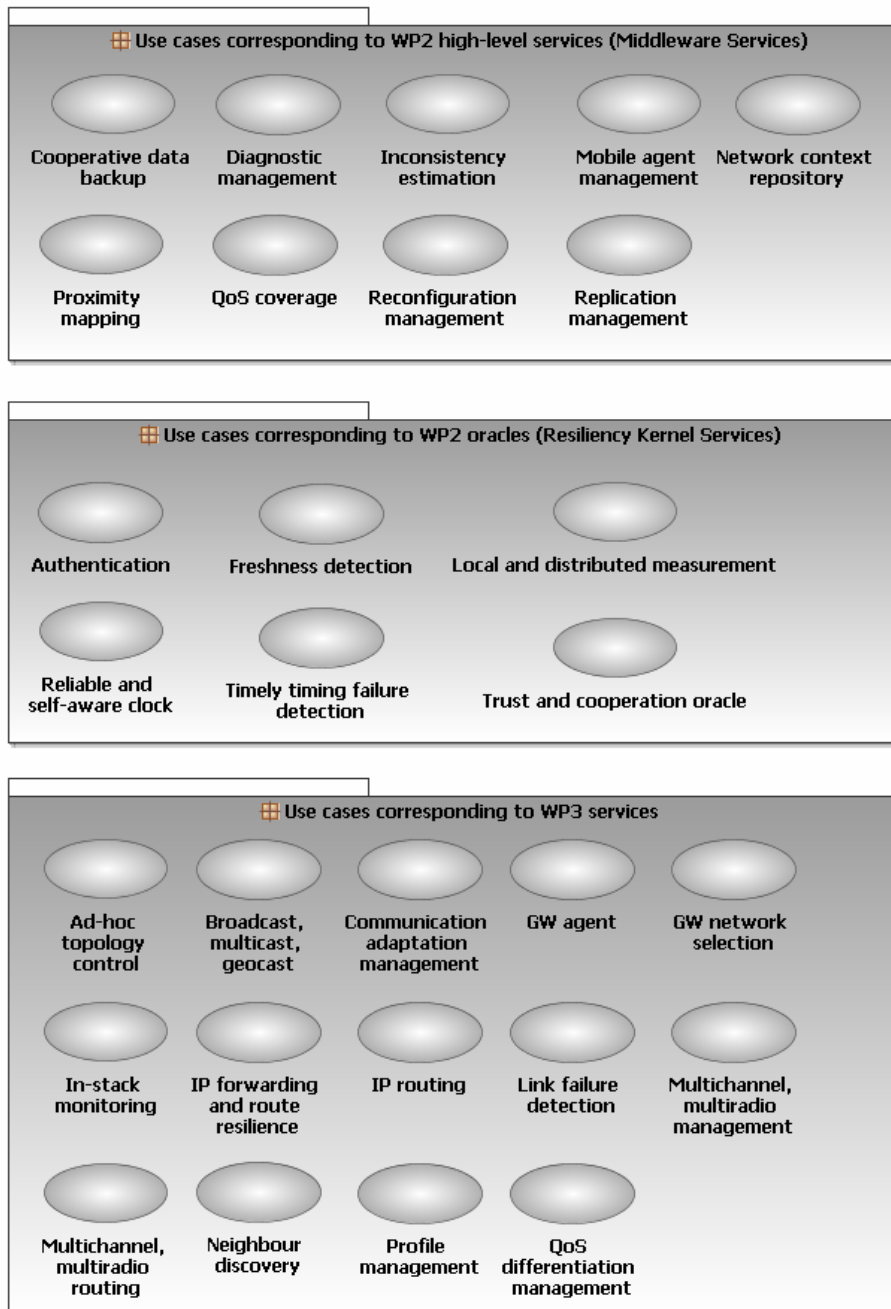


Figure 18: Alignment of use cases according to the architectural hybridization concept



This diagram presents an overview on use-case packages.

Figure 19: Use case packages

Having introduced our basic use case model we assigned *metaclasses implementing them*. Since the input documents (previous deliverables of the project) were explicitly talking about services to be defined or implemented, we had neither the need nor the possibility of modifying the basic concepts thus the assignment of metaclasses to use cases was nearly trivial (e.g., we assigned the metaclass *Authentication Service* to the *Authentication* use case, etc.). (Note again: (i) *use cases* were introduced for the organization of our modelling efforts (but use cases themselves are not metamodeling concepts) while (ii) *metaclasses* were introduced as actual modelling concepts of the metamodel.)

This diagram presents the association of use cases and metaclasses implementing them in the context of WP2 low-level services (oracles).

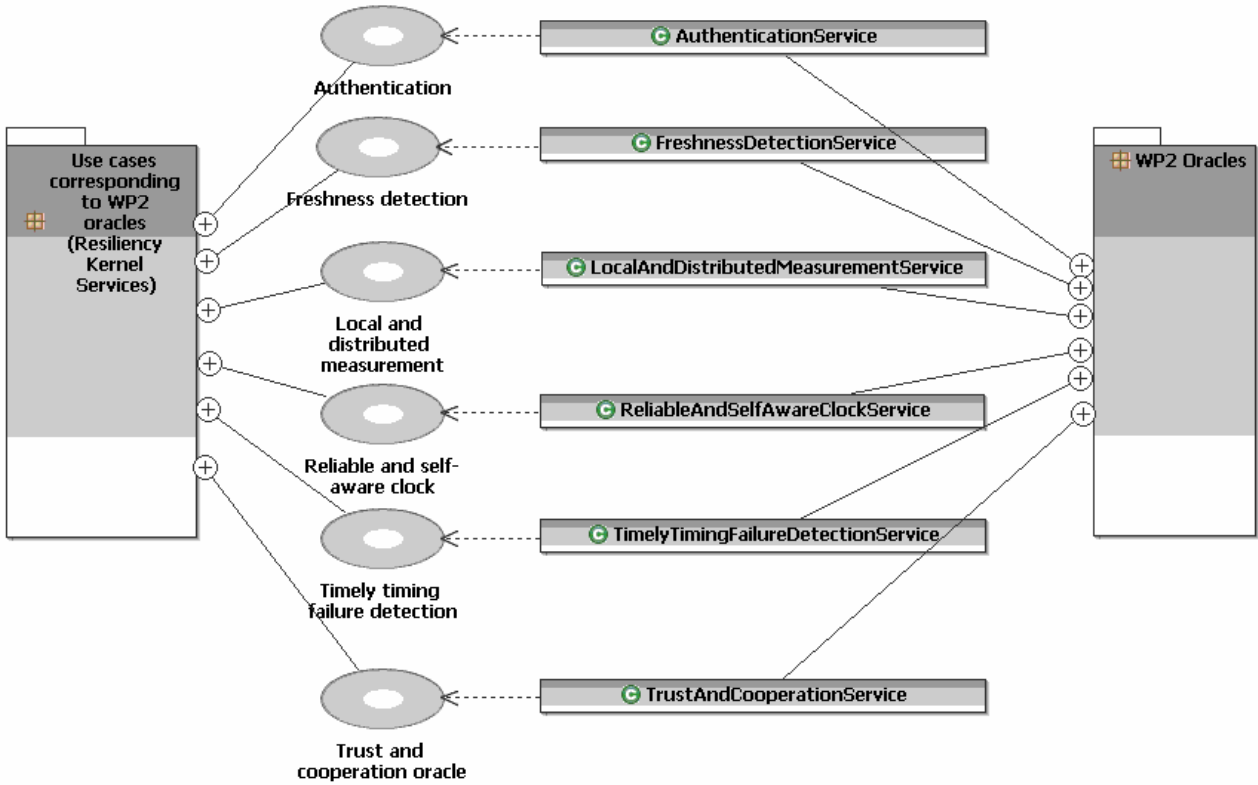


Figure 20: WP2 oracles: use cases and metaclasses implementing them

The use cases corresponding to WP2 oracles (simple trusted resiliency kernel services) and the metaclasses implementing them (indicated by a dependency relation) are shown in Figure 20. As indicated in the figure the metaclasses are assigned to the *WP2 oracles* metaclass package.

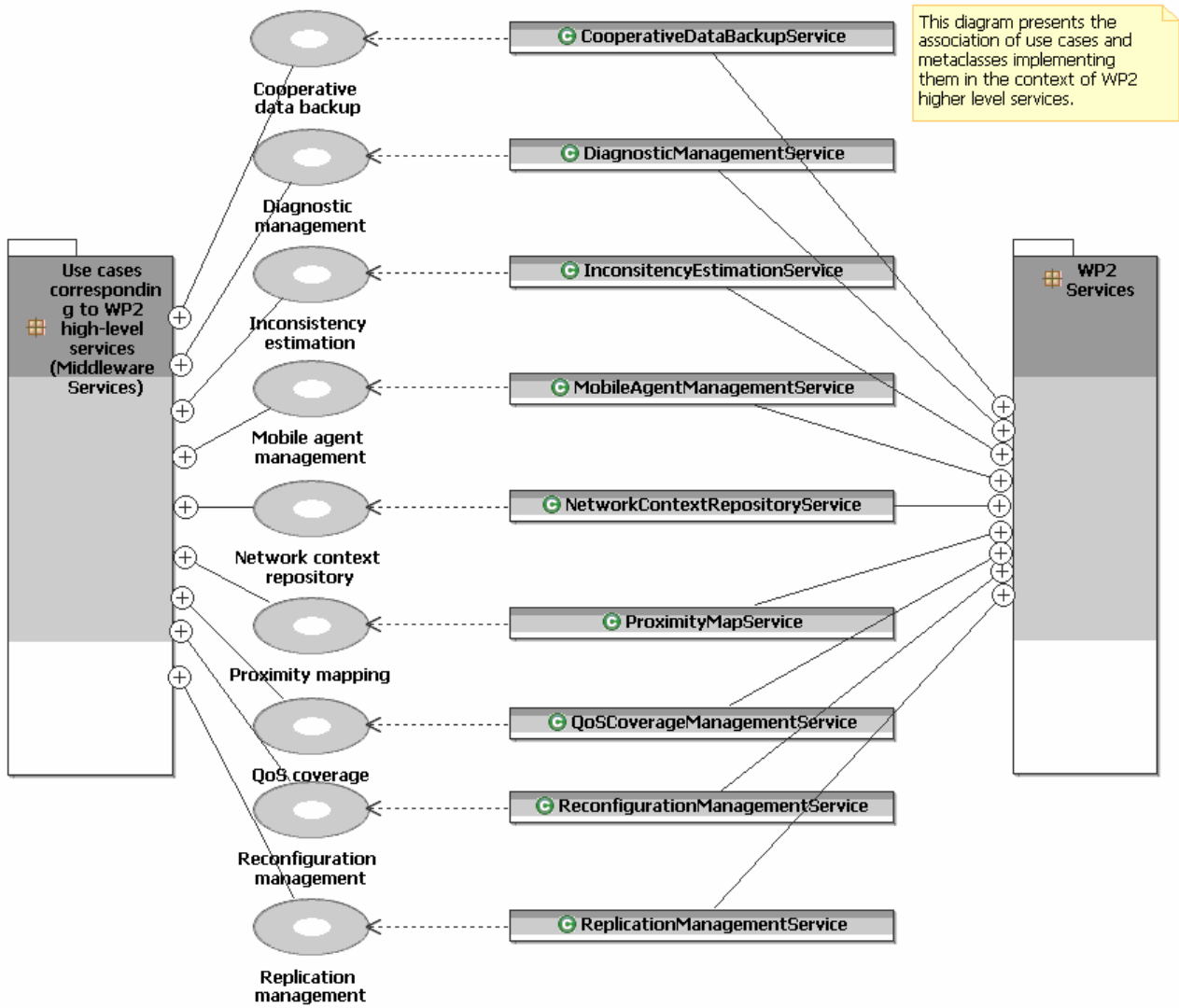


Figure 21: WP2 high-level services: use cases and metaclasses implementing them

The use cases corresponding to high-level WP2 services (less trusted middleware services) and the metaclasses implementing them (indicated by a dependency relation) are shown in Figure 21. As indicated in the figure the metaclasses are assigned to the *WP2 services* metaclass package.

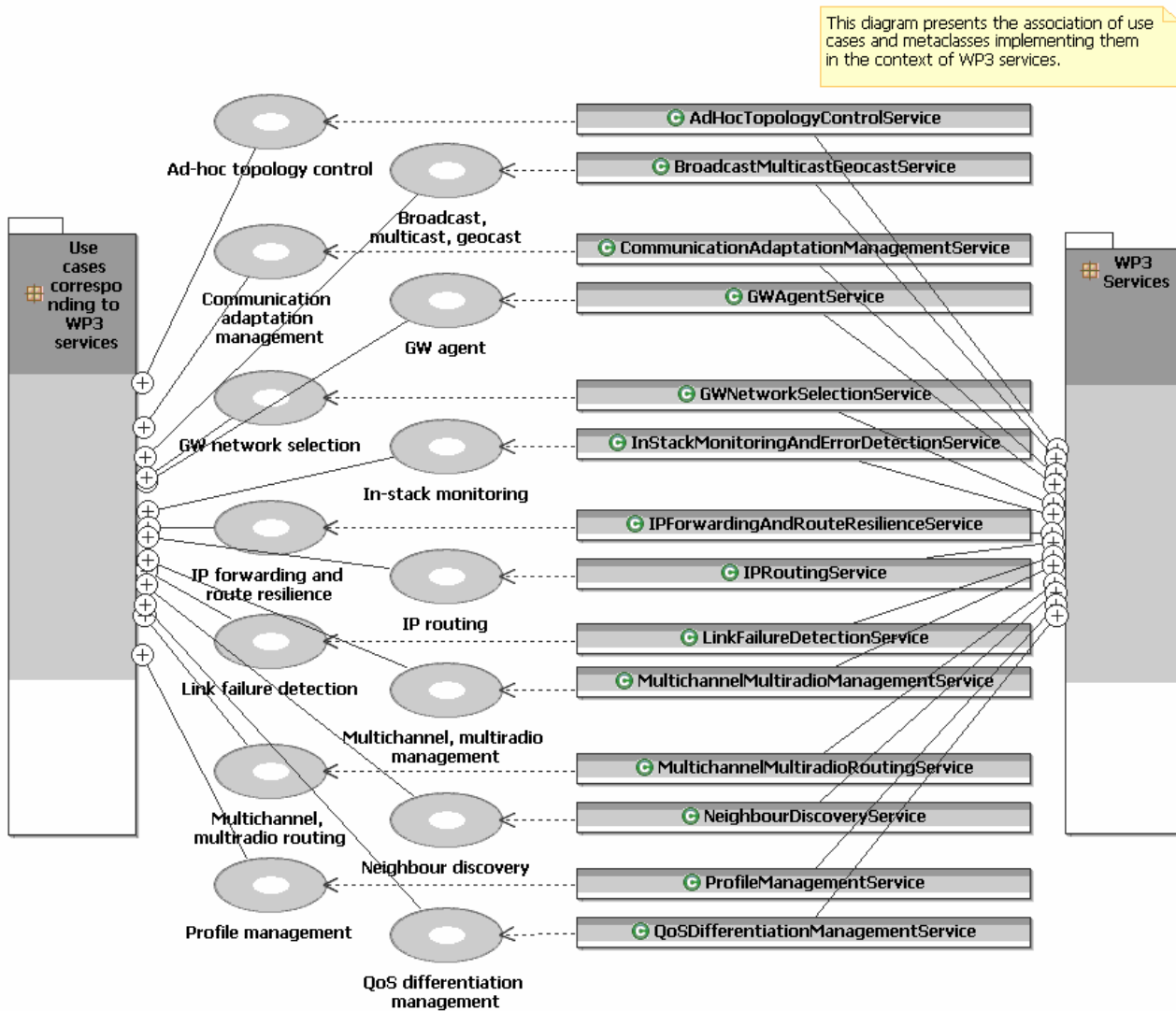


Figure 22: WP3 services: use cases and metaclasses implementing them

The use cases corresponding to WP3 services and the metaclasses implementing them (indicated by a dependency relation) are shown in Figure 22. As indicated in the figure the metaclasses are assigned to the *WP3 services* metaclass package.

The rest of this section discusses the various properties, associations and inheritance relations of the metaclasses introduced above. We will define various *new data types* (based on previously published standards where possible) and indicate the fundamental *operations* provided by services as operations (functions) of metaclasses.

3.2 Notation

Since this document contains references to various previously published standards it is important to *visually distinguish* the concepts borrowed from some external sources and the ones newly introduced by us. We refer to four external standards: (i) the *UML Profile for Performance, Schedulability and Time* (SPT) [SPT], (ii) the *UML Profile for Modelling Quality of Service and Fault Tolerance Characteristics and Mechanisms* (QoS & FT) [QoSFT], (iii) the *Service Availability Forum* (SAF) Metamodel [SAF] and (iv) the General Resource Modelling Framework

(GRM). This way we will use seven colours for highlighting the classes belonging to specific metamodels (blue for .QoS & FT, green for SPT, yellow for SAF, cyan for GRM, orange for original UML metamodel elements, a black-and-white scheme for HIDENETS artifacts and red for indicating various issues or aspects that require more in-depth consideration in the future); see Figure 23.

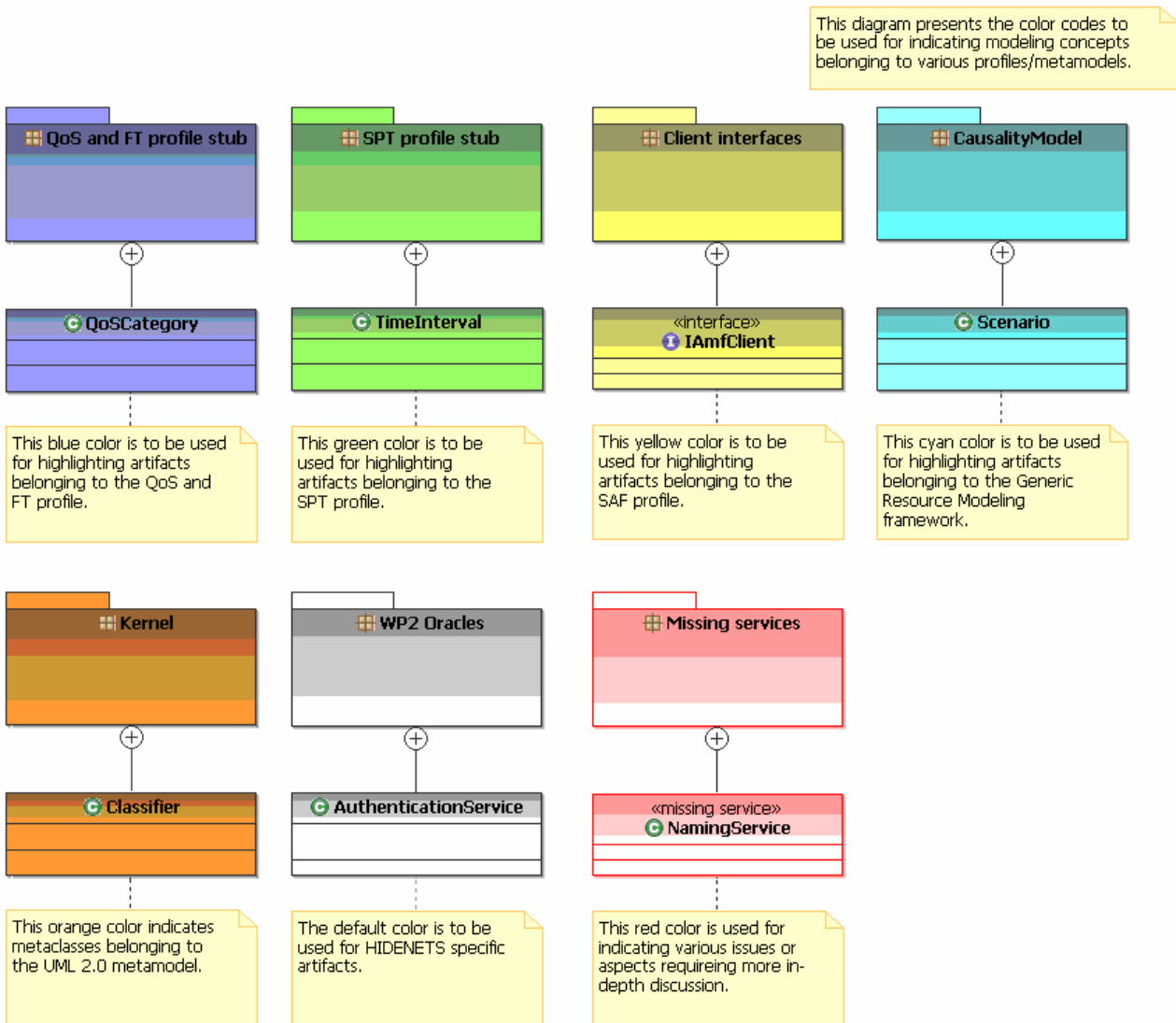


Figure 23: Colour codes used for distinguishing metaclasses

It is important to highlight however that the colour coding is just a straightforward visual support for the reader; the metaclasses are stored in their corresponding packages as also shown in Figure 23.

3.3 Overview on the model structure

The *basic structure of the metamodel* is shown in Figure 24: (i) the three metaclass packages at the bottom of the figure are the ones introduced above (WP2 oracles, WP services and WP3 services); as we will see below the services belonging to various service packages may use each others' operations – these dependencies are highlighted by usage relations between packages in the figure;

(ii) services are derived from a small number of abstract base metaclasses, these are collected into the package *Base metaclasses*; (iii) operations are discussed below in a function-like notation (i.e., input parameters, output type, etc.) this necessitates the introduction of well-defined data types; the data types used for defining HIDENETS services are collected into the package *Data types*; (iv) when defining data types it is definitely beneficial to establish our discussion on the basis of some previously published standards; this way the data type package of our approach uses some concepts borrowed from the SPT and the QoS & FT standards as indicated at the top of the figure (note the colour coding).

The rest of this section traverses through these packages similarly to the usual discussion of metamodelling i.e., by first presenting the abstract syntax (metaclasses with their attributes and relations) then textually explaining the meaning of metaclasses one by one. Note that the most of the textual discussions presented here were copied from previous deliverables of the project or the standards mentioned above; our contributions are the visual representation, the unambiguous structure, the definition of data types and the organization of concepts defined by HIDENETS and the ones borrowed from previously published approaches into a single model.

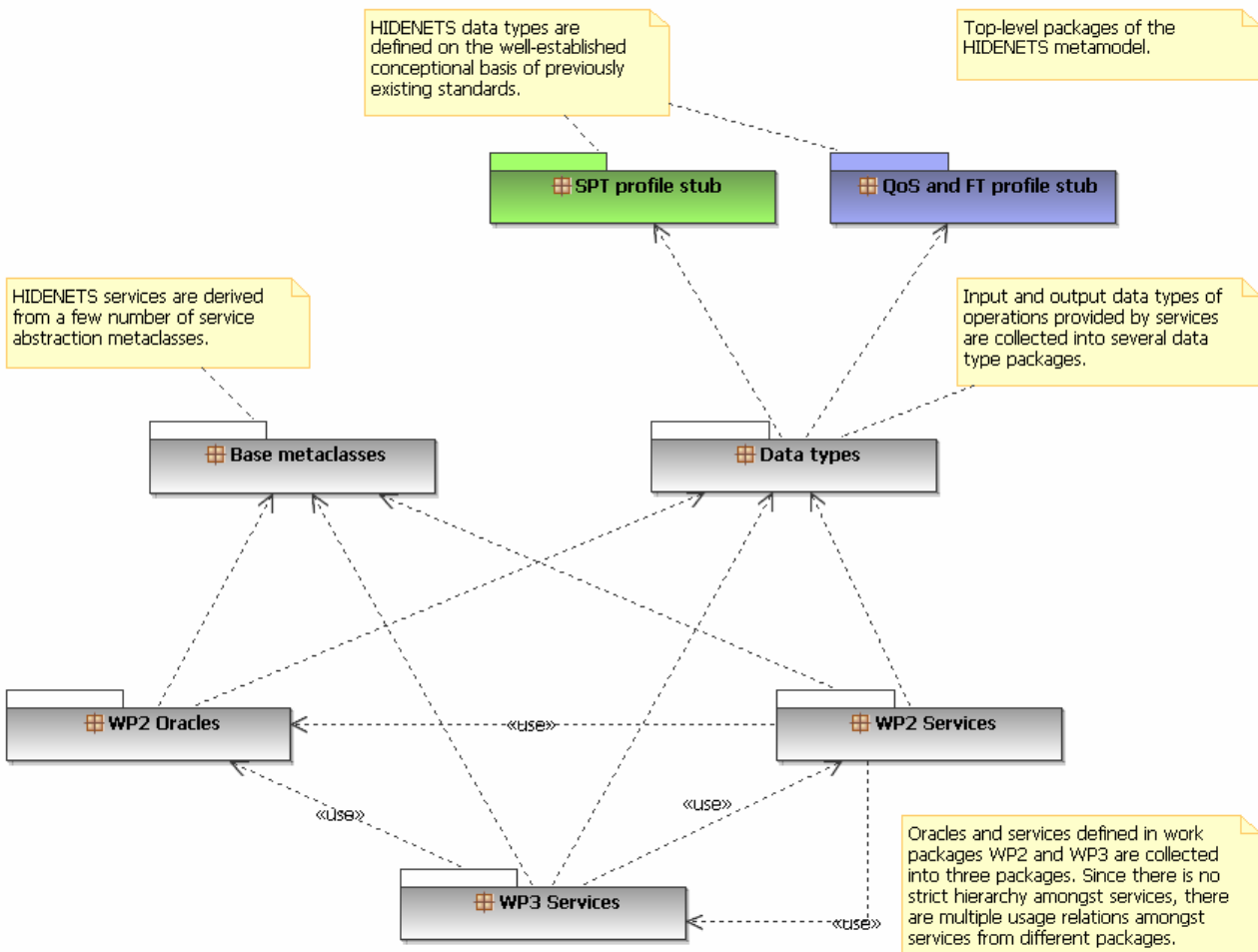


Figure 24: Basis structure of the metamodel

3.4 Fundamental data types introduced by HIDENETS

Below we introduce those primitive classes that are used as basic *data types* by some HIDENETS services (e.g., time-related concepts to be used for reasoning about reliable and self-aware clock service, etc.).

3.4.1 Overview

The data types introduced in various HIDENETS contexts are organized into seven packages (Figure 25): (i) communication, (ii) data security, (iii) geographical data, (iv) identifiers, (v) time-related concepts, (vi) quality of service and (vii) a package of such utility data types that did not fit to the packages mentioned above (and may require more in-depth analysis).

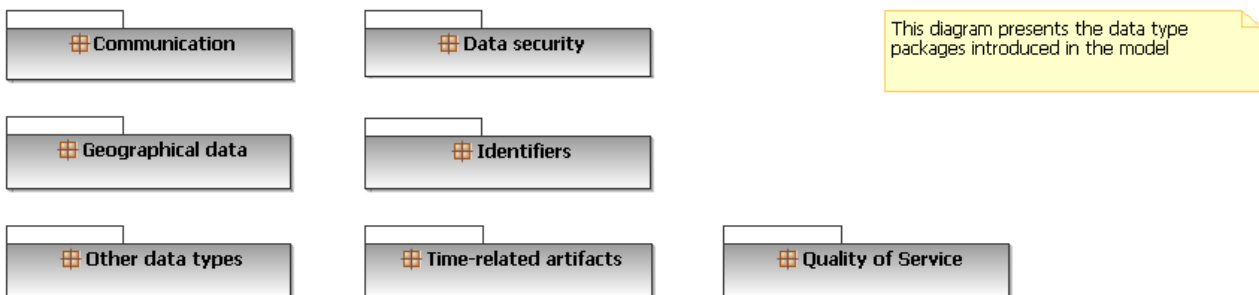


Figure 25: Data type packages

3.4.2 Abstract syntax

Data types related to the *communication* of HIDENETS entities are shown in Figure 26.



Figure 26: Communication-related data types

Data types related to *data security* are shown in Figure 27.

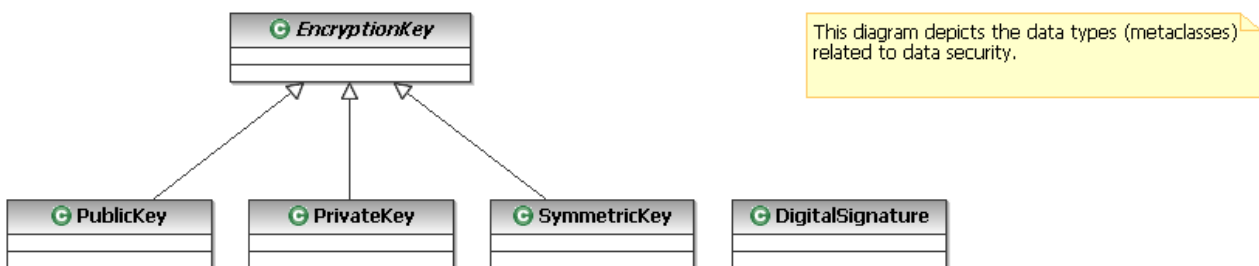
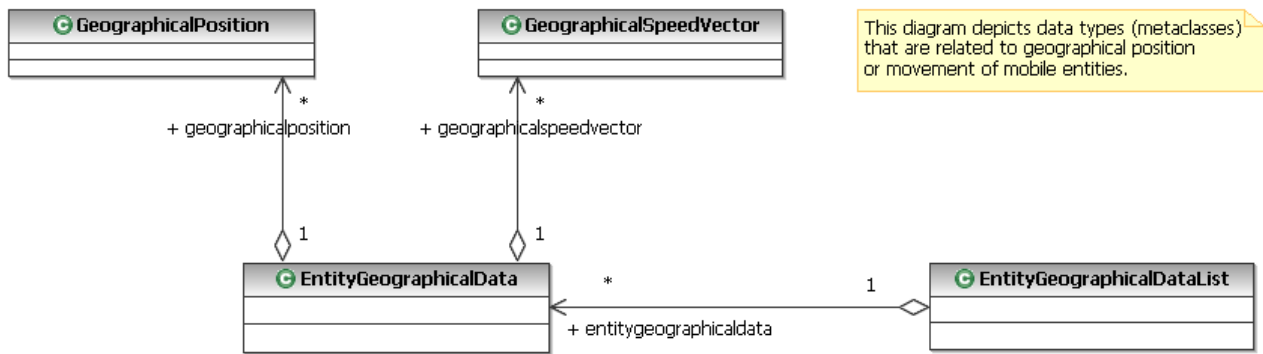


Figure 27: Data types related to data security

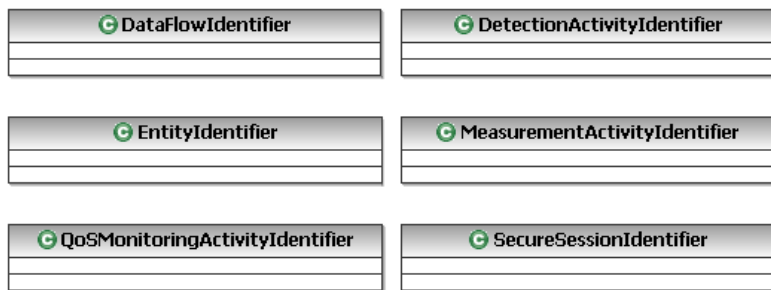
Data types related to the *geographical position of entities* are shown in Figure 28.



This diagram depicts data types (metaclasses) that are related to geographical position or movement of mobile entities.

Figure 28: Geographical data types

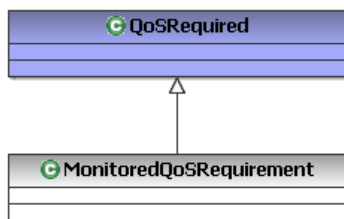
Data types used as *identifiers* of various objects are shown in Figure 29.



This diagram depicts data types (metaclasses) that are used as identifiers of some entities, services, etc.

Figure 29: Identifiers

Data types related to the *quality of some services* are shown in Figure 30. Note that (i) the HIDENETS concept was derived from the corresponding metaclass of the QoS & FT profile and (ii) more in-depth modelling of QoS related concepts is needed in the future based on the field expertise of corresponding project partners.



QoS-related data types and their ancestors in the QoS & FT profile (note that QoS-related data types are not yet fully elaborated).

Figure 30: QoS requirement derived from the corresponding QoS & FT profile concept

Data types used for *modelling time*, time intervals and timing failures are shown in Figure 31. For semantically well-established discussion the most fundamental concepts were derived from metaclasses of the SPT profile as shown in Figure 32.

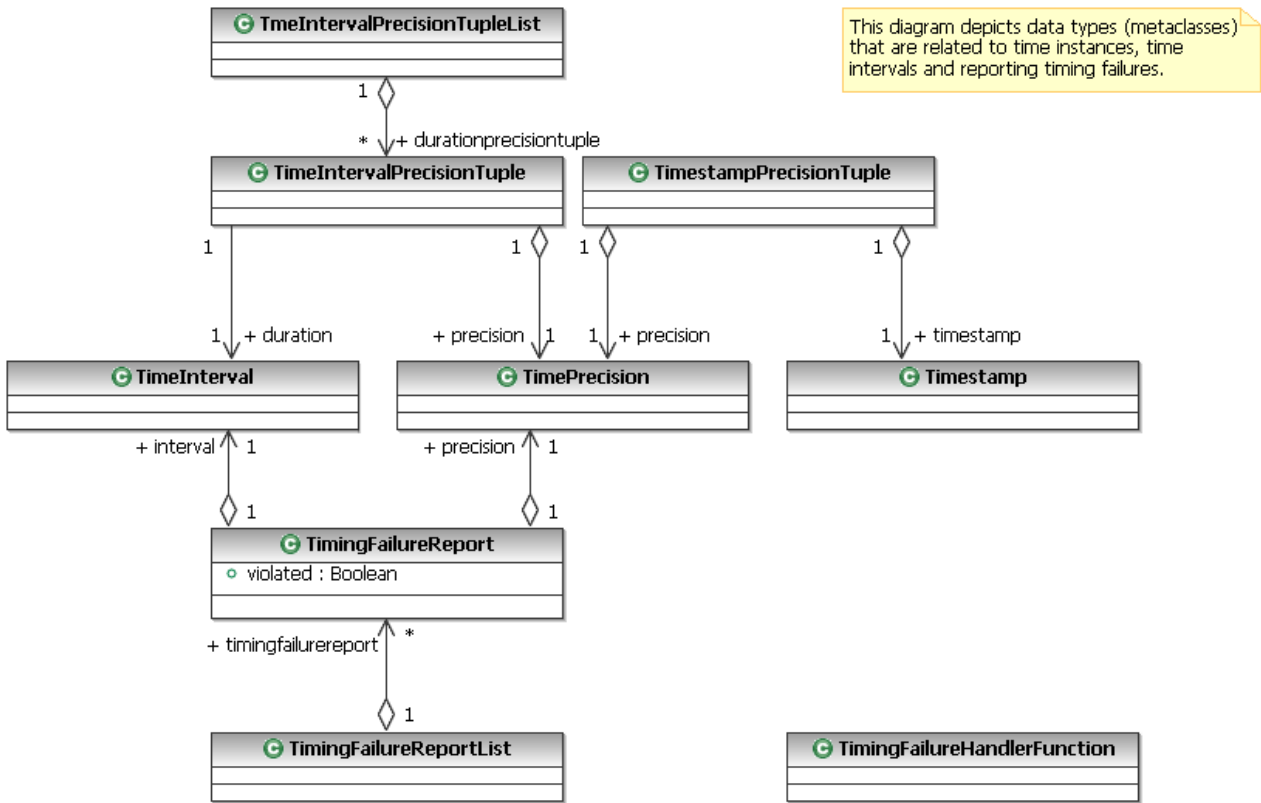


Figure 31: Time-related concepts

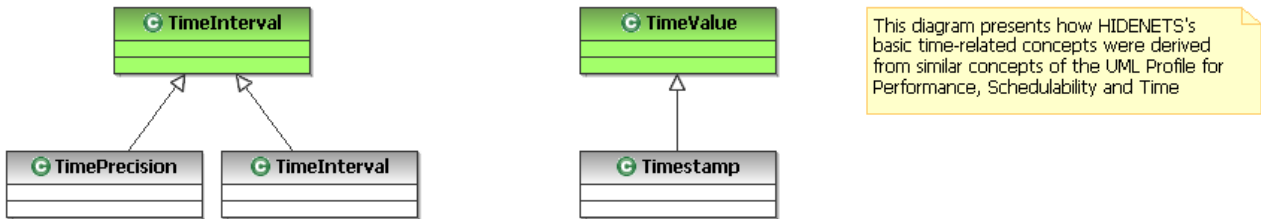


Figure 32: Deriving time-related concepts from SAF profile metaclasses

Those utility data types that did not directly belong to the packages mentioned above are presented in Figure 33.

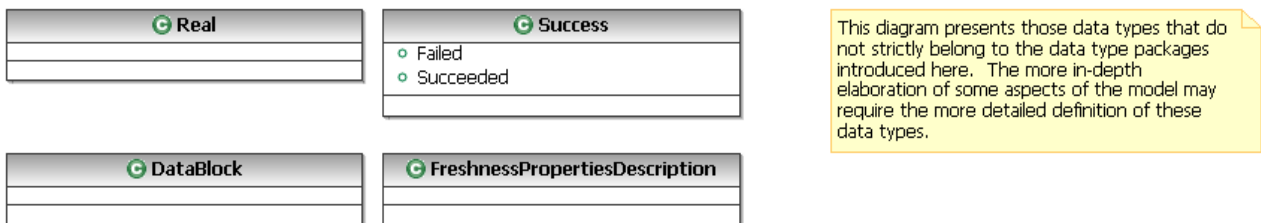


Figure 33: Miscellaneous data types

3.4.3 Class descriptions

3.4.3.1 Metaclass Message

HIDENETS deliverables sometimes talk about application level messages without indicating whether they are signed, encrypted etc. This metaclass provides a common base for them where the encrypted or digitally signed state is indicated by attributes. Attributes:

- IsEncrypted: Indicates that the message is encrypted.
- IsSigned: Indicates that the message is digitally signed.

3.4.3.2 Metaclass Data Flow Position

Position in a data flow.

3.4.3.3 Metaclass Encryption Key

Authentication and related services talk about encryption keys and related concepts. This metaclass provides their common conceptual basis.

3.4.3.4 Metaclass Public Key

Public key for some encryption, digital signature etc. algorithm.

3.4.3.5 Metaclass Private Key

Private key for some encryption, digital signature etc. algorithm.

3.4.3.6 Metaclass Symmetric Key

Symmetric key for some encryption algorithm.

3.4.3.7 Metaclass Digital Signature

A digital signature calculated for a block of data.

3.4.3.8 Metaclass Geographical Position

Position of a mobile entity.

3.4.3.9 Metaclass Geographical Speed Vector

Speed vector of an entity.

3.4.3.10 Metaclass Entity Geographical Data

Composite data containing information about the position and the speed of a mobile entity.

3.4.3.11 Metaclass Entity Geographical Data List

Composite data structure containing information about the position and the speed of several mobile entities.

3.4.3.12 Metaclass Data Flow Identifier

Unique identifier of a data flow.

3.4.3.13 Metaclass Entity Identifier

Unique identifier of an entity.

3.4.3.14 Metaclass QoS Monitoring Activity Identifier

Unique identifier of an activity performed by the QoS Coverage Management service.

3.4.3.15 Metaclass Detection Activity Identifier

Unique identifier of an activity performed by the Timely Timing Failure Detection service.

3.4.3.16 Metaclass Measurement Activity Identifier

Unique identifier of an activity performed by the Local and Distributed Measurement service.

3.4.3.17 Metaclass Secure Session Identifier

Unique identifier of a secure session.

3.4.3.18 Metaclass Monitored QoS Requirement

A QoS constraint that is required by an application or service; e.g., such a constraint can be monitored by the QoS Coverage Manager.

3.4.3.19 Metaclass Timestamp

The value of a clock at a specific physical time instance.

3.4.3.20 Metaclass Time Interval

A time interval measured by a clock.

3.4.3.21 Metaclass Time Precision

Precision (i.e., possible deviation from the theoretical correct value) of some time-related concept (e.g., timestamp, duration, etc.).

3.4.3.22 Metaclass Time Interval Precision Tuple

A compound data type containing an interval coupled with its precision.

3.4.3.23 Metaclass Timestamp Precision Tuple

A compound data type containing a time stamp coupled with its precision.

3.4.3.24 Metaclass Time Interval Precision Tuple List

List of time interval precision tuples.

3.4.3.25 Metaclass Timing Failure Report

A tuple describing a timing failure (i.e., the length of the measured interval, the precision of measurement and the Boolean indication whether the measurement detected a timing failure or the observed activity was finished in time. Attributes:

- violated: indicates whether a timing failure was detected.

3.4.3.26 Metaclass Timing Failure Report List

A list of timing failure reports.

3.4.3.27 Metaclass Timing Failure Handler Function

A function to be called in case of some timing-failure.

3.4.3.28 Metaclass Real

Floating point number.

3.4.3.29 Metaclass Data Block

Arbitrary unstructured block of data.

3.4.3.30 Metaclass Success

This enumerated data type indicates the success or the failure of some operation.

3.4.3.31 Metaclass Freshness Properties Description

Description of freshness properties.

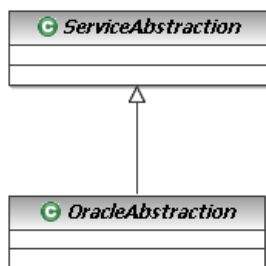
3.5 Base metaclasses for HIDENETS artifacts

3.5.1 Overview

Below we present the two abstract metaclasses that HIDENETS services are derived from.

3.5.2 Abstract syntax

As presented in Figure 34 we use two abstract metaclasses as ancestors of HIDENETS services: (i) service abstractions and (ii) oracles abstraction. As shown in Figure 34 oracles are considered to be special services (they provide more trusted services, etc.).



This diagram depicts the base metaclasses that services are derived from.

Figure 34: Base metaclasses for HIDENETS services

3.5.3 Class descriptions

3.5.3.1 Metaclass Service Abstraction

This abstract metaclass is the base of all metaclasses introduced in the context of HIDENETS concepts.

3.5.3.2 Metaclass Oracle Abstraction

According to the HIDENETS terminology, six services (authentication, freshness detection, local and distributed measurements, reliable and self-aware clock, timely timing failure detection and

trust and cooperation services) are considered to be relatively simple but essential services that are to be highly dependable. These services are considered to belong to the so-called "Resiliency kernel" and services themselves are referred to as "oracles". This metaclass provides a common basis for all oracles.

3.6 Fundamental oracles defined by WP2

3.6.1 Overview

Below we present the metaclasses corresponding to the six *oracles* defined in WP2.

3.6.2 Abstract syntax

Metaclasses corresponding to the six *oracles* defined in WP2 are presented in Figure 35 coupled with the operations provided by them. The *co-operation of oracles* is indicated as usage relations in Figure 36.

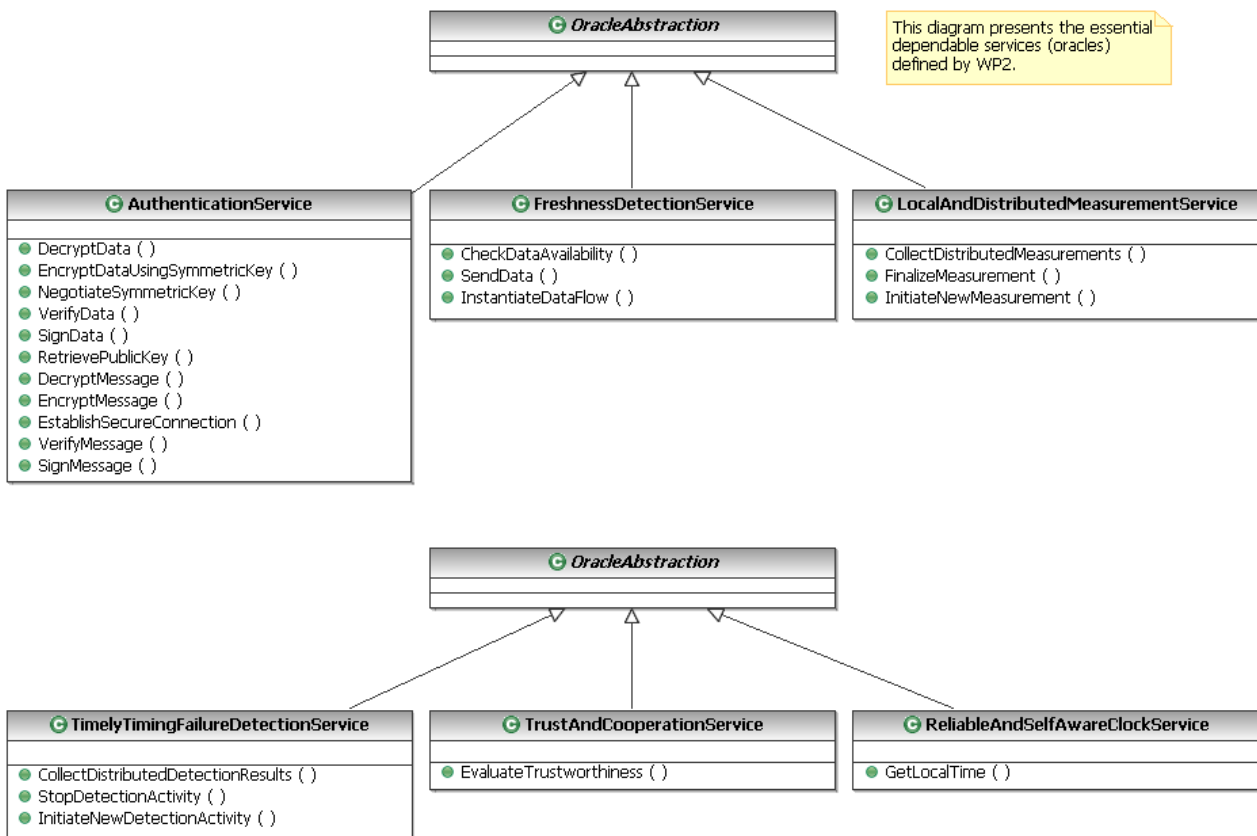


Figure 35: WP2 oracles

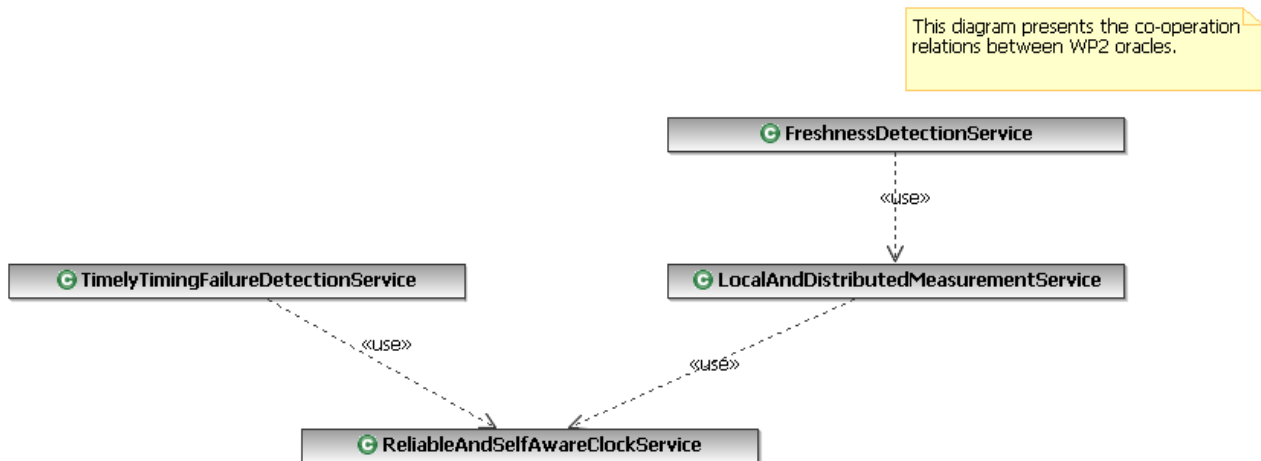


Figure 36: Co-operation of WP2 oracles

3.6.3 Class descriptions

3.6.3.1 Metaclass Authentication Service

The Authentication service serves to secure the identity of the sender. By providing this service as an oracle, it is ensured that the authentication module is kept safe from potential intrusions and, consequently, that the dependent oracles always receive correct authentication information about the identity of the entities.

Operations:

Name of the operation	DecryptData
Return type	DataBlock
Argument list	Class SymmetricKey, Class DataBlock
Description	Decrypts arbitrary data using a symmetric key.
Name of the operation	DecryptMessage
Return type	Message
Argument list	Class Message, Class SecureSessionIdentifier
Description	Decrypts a message within the context of a previously established secure session.
Name of the operation	EncryptDataUsingSymmetricKey
Return type	DataBlock
Argument list	Class SymmetricKey, Class DataBlock
Description	Encrypts arbitrary data using a symmetric key.

Name of the operation	EncryptMessage
Return type	Message
Argument list	Class Message, Class SecureSessionIdentifier
Description	Encrypts a message within the context of a previously established secure session.
Name of the operation	EstablishSecureConnection
Return type	SecureSessionIdentifier
Argument list	Class EntityIdentifier
Description	Establishes a secure communication session with the oracle of another entity by negotiating a symmetric key.
Name of the operation	NegotiateSymmetricKey
Return type	SymmetricKey
Argument list	Class EntityIdentifier
Description	Negotiates a symmetric key with the oracle of another entity.
Name of the operation	RetrievePublicKey
Return type	PublicKey
Argument list	Class EntityIdentifier
Description	Retrieves the public-key of a given entity which can be either a car or a server in the infrastructure.
Name of the operation	SignData
Return type	DigitalSignature
Argument list	Class DataBlock
Description	Digitally signs arbitrary data using the private key.
Name of the operation	SignMessage
Return type	Message
Argument list	Class Message
Description	Digitally signs a message using the private key.
Name of the operation	VerifyData

Return type	Boolean
Argument list	Class DataBlock, Class DigitalSignature, Class EntityIdentifier
Description	Verifies if given signature on arbitrary data is valid.

Name of the operation	VerifyMessage
Return type	Boolean
Argument list	Class Message, Class EntityIdentifier
Description	Verifies if a given message signature is valid.

3.6.3.2 Metaclass Freshness Detection Service

The Freshness Detector is the component able to check temporal properties related to freshness of connection-oriented data flows; it is a component able to detect violations of a particular class of real-time requirements.

Operations:

Name of the operation	CheckDataAvailability
Return type	DataBlock
Argument list	Class DataFlowIdentifier
Description	Checks if the required predicates of freshness on the data flow are kept and, if available, it returns data to the user.

Name of the operation	InstantiateDataFlow
Return type	DataFlowIdentifier
Argument list	Class EntityIdentifier, Class FreshnessPropertiesDescription
Description	Instantiates logical connections between the end-hosts, on which the user can define predicates of freshness on the data flow.

Name of the operation	SendData
Return type	Success
Argument list	Class DataFlowIdentifier, Class DataBlock
Description	Sends data through a data flow currently instantiated.

3.6.3.3 Metaclass Local and Distributed Measurement Service

The duration measurement service can be used to measure local or distributed durations with bounded precision. This building block provides routines to start and stop the measurement of activities taking place locally or in a distributed setting.

Operations:

Name of the operation	CollectDistributedMeasurements
Return type	TimeIntervalPrecisionTupleList
Argument list	Class MeasurementActivityIdentifier
Description	Allows any participant in a distributed activity to collect measurements of the duration of that activity.
Name of the operation	FinalizeMeasurement
Return type	TimeIntervalPrecisionTuple
Argument list	Class MeasurementActivityIdentifier
Description	Stops a given measurement (in principle invoked after the end of the measure activity).
Name of the operation	InitiateNewMeasurement
Return type	MeasurementActivityIdentifier
Argument list	Class Timestamp
Description	Requests a new measurement activity to be performed by the oracle.

3.6.3.4 Metaclass Reliable and Self Aware Clock Service

The Reliable and Self-Aware Clock (R&SA Clock) is a software 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. UTC, Universal Time Coordinated). The R&SA Clock provides information about time and it is aware of the current precision level of the information provided.

Operations:

Name of the operation	GetLocalTime
Return type	TimestampPrecisionTuple
Argument list	
Description	Provides an abstraction of the local clock.

3.6.3.5 Metaclass Timely Timing Failure Detection Service

The Timely timing failure detector checks if individual timed actions incur in timing failures, and does that as timely as allowed by the synchronicity of the trusted part.

Operations:

Name of the operation	CollectDistributedDetectionResults
Return type	TimingFailureReportList
Argument list	Class DetectionActivityIdentifier

Description	Allows any participant in a distributed activity to collect timing failure detection results.
Name of the operation	InitiateNewDetectionActivity
Return type	DetectionActivityIdentifier
Argument list	Class TimeInterval, Class Timestamp, Class TimingFailureHandlerFunction
Description	Requests a new timing failure detection activity to be performed by the oracle.
Name of the operation	StopDetectionActivity
Return type	TimingFailureReport
Argument list	Class DetectionActivityIdentifier
Description	Stops an ongoing detection activity, typically after the timed activity has been concluded.

3.6.3.6 Metaclass Trust and Cooperation Service

The Trust and Cooperation service is able to evaluate locally the level of trust of neighbouring entities and to manage cooperation incentives, which is necessary for building cooperative services.

Operations:

Name of the operation	EvaluateTrustworthiness
Return type	Real
Argument list	Class EntityIdentifier
Description	Provides an evaluation of the node's trustworthiness (between 0 and 1, 1 meaning trusted).

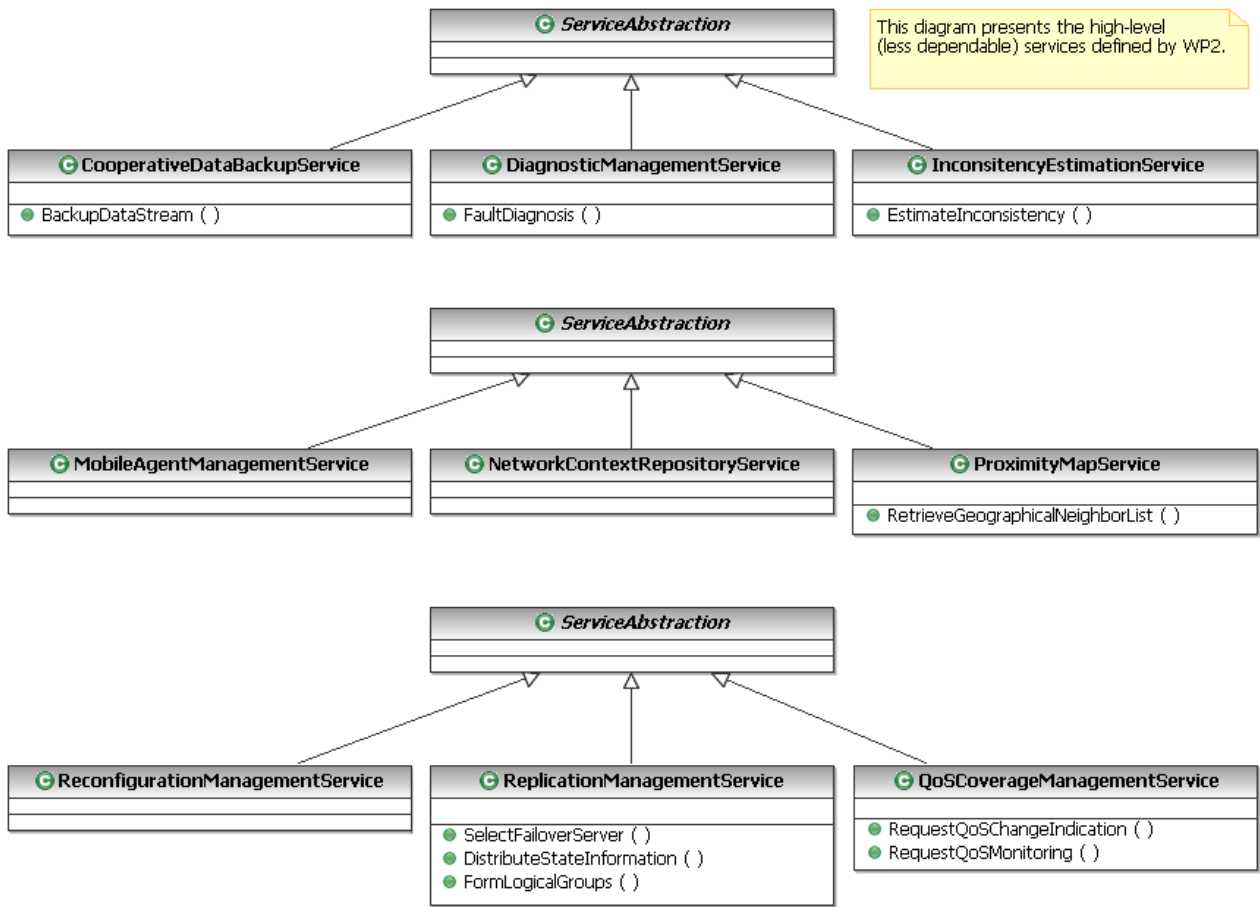
3.7 High-level services defined in WP2

3.7.1 Overview

Below we present the metaclasses corresponding to the nine high-level (i.e., more complex and less trusted) services defined in WP2.

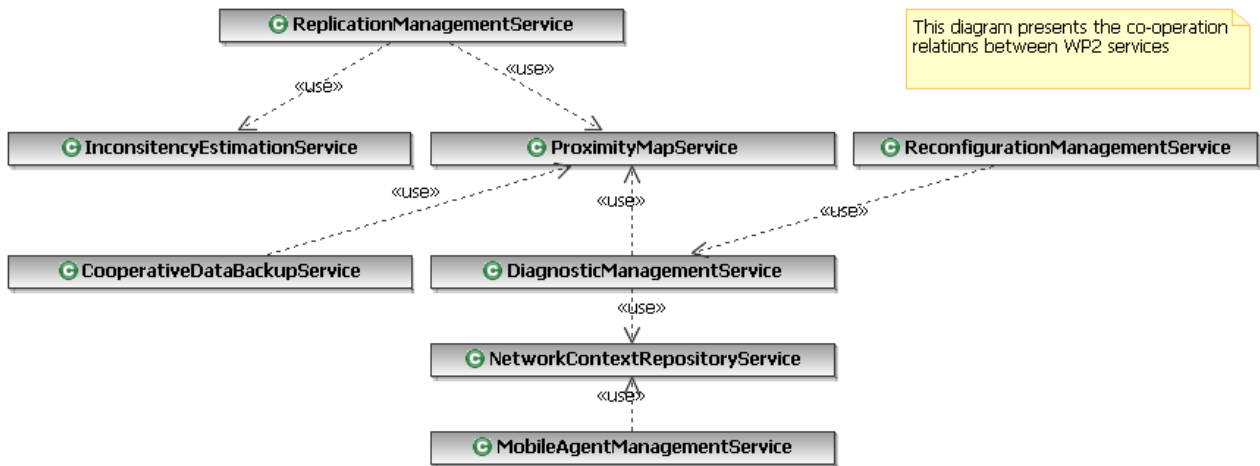
3.7.2 Abstract syntax

The metaclasses representing high-level WP2 services are shown in Figure 37. The co-operation of these services (i.e., how high-level WP2 services use other high-level WP2 services) is shown in Figure 38. The interoperation between high-level WP2 services and WP2 oracles (i.e., how high-level less trusted WP2 services use fundamental highly dependable WP2 oracles) is presented in Figure 39. The interoperation between WP2 and WP3 services (i.e., how high-level WP2 services use some WP3 services) is shown in Figure 40.



This diagram presents the high-level (less dependable) services defined by WP2.

Figure 37: High-level WP2 services



This diagram presents the co-operation relations between WP2 services

Figure 38: Co-operation of high-level WP2 services

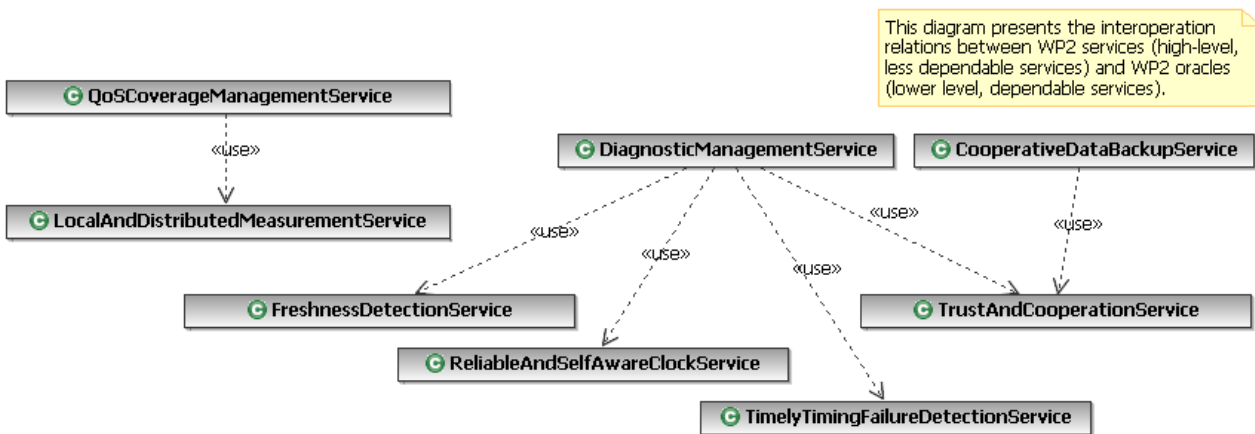


Figure 39: Interoperation relations between high-level WP2 services and oracles



Figure 40: Interoperation relations between high-level WP2 services and WP3 services

3.7.3 Class descriptions

3.7.3.1 Metaclass Cooperative Data Backup Service

The problem of cooperative backup of critical data consists essentially of discovering storage resources in the vicinity, negotiating a contract with the neighbouring cars for the use of their resources, handling a set of data chunks to backup and assigning these chunks to the negotiated resources.

Operations:

Name of the operation	BackupDataStream
Return type	DataFlowPosition
Argument list	Class DataFlowIdentifier
Description	Ensures that a data stream is backed up to available neighbors and eventually on the infrastructure.

3.7.3.2 Metaclass Diagnostic Management Service

The Diagnostic Manager (DM) judges whether the system (or parts of it) is (are) working properly or not. Improper component behaviour could be the result of manifestation of faults affecting the component itself, changes in the requirements of the application using the component, environmental changes which lead to a change in the QoS provided. DM works on-line, diagnosing the status of monitored components on the basis of information collected over-time by appropriate error/deviation detection mechanisms the system is equipped with.

Operations:

Name of the operation	FaultDiagnosis
-----------------------	----------------

Return type

Argument list

Description Judges whether the system (or parts of it) is (are) working properly or not. Improper component behaviour could be the result of (i) manifestation of faults affecting the component itself (ii) changes in the requirements of the application using the component or (iii) environmental changes which lead to a change in the QoS provided DM works on-line, diagnosing the status of monitored components on the basis of information collected over-time by appropriate error/deviation detection mechanisms the system is equipped with. Diagnosis is performed both locally to a node (basing on local information in order to assess the internal state of the node) and globally to a group of nodes (in order to agree on the status of every node in the group).

3.7.3.3 Metaclass Inconsistency Estimation Service

The goal of the inconsistency evaluation module is to quantitatively estimate inconsistency for the replicated stateful HIDENETS services - possibly for a given replicated server and at any given time. Several inconsistency evaluation frameworks have been proposed for the infrastructure domain. The main foreseen contribution will be the extension/adaptation of those frameworks (or the creation of a new framework if needed) for HIDENETS services in the ad-hoc domain.

Operations:

Name of the operation	EstimateInconsistency
Return type	Real
Argument list	
Description	Responsible for estimating data inconsistency based on measured metrics (delays).

3.7.3.4 Metaclass Mobile Agent Management Service

A mobile agent is a piece of software that traverses the network. On one hand, it gets local information from the nodes it is visiting and makes decisions about the next step for its own activity. On the other hand, it may affect the visited node by leaving information or trigger local activities. Mobile Agent Manager is a software component supporting the execution of a mobile agent.

Operations:

No operation defined so far.

3.7.3.5 Metaclass Network Context Repository Service

This building block acts as a database for network and communication related information.

Operations:

No operation defined so far.

3.7.3.6 Metaclass Proximity Map Service

The goal of the proximity map service is to provide a map of neighbouring 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.

Operations:

Name of the operation	RetrieveGeographicalNeighborList
Return type	EntityGeographicalDataList
Argument list	Class TimePrecision
Description	Retrieves the list of vehicles in the vicinity with positioning and speed information, assuring that the returned list is not older than the given TimePrecision.

3.7.3.7 Metaclass QoS Coverage Management Service

The QoS coverage manager will be in charge of evaluating whether the QoS requirements of an application are satisfied, or else if it is necessary to provide an indication that QoS may no longer be guaranteed and may need to be renegotiated. In order to do that, the QoS coverage manager will need information concerning the performance of the network and, in particular, it will need to use histories of measured parameters to perform statistical calculations.

Operations:

Name of the operation	RequestQoSMonitoring
Return type	QoSMonitoringActivityIdentifier
Argument list	Class MonitoredQoSRequirement, Class DataFlowIdentifier
Description	Allows setting up a monitoring activity, which will be used to provide information and indications to help an application adapt in a dependable way, that is, to let it adapt in order to meet a specified coverage criteria.

Name of the operation	RequestQoSChangeIndication
Return type	QoSMonitoringActivityIdentifier
Argument list	Class QoSMonitoringActivityIdentifier
Description	Blocks the application (or thread) waiting for an indication from the manager, which will contain updated information relative to bounds to be used.

3.7.3.8 Metaclass Reconfiguration Management Service

The Reconfiguration Manager component is part of the "Fault Tolerance Manager" and it is specifically in charge of deciding both when some reconfiguration is needed and which reconfiguration policy has to be applied.

Operations:

No operation defined so far.

3.7.3.9 Metaclass Replication Management Service

The replication manager is a tool that is used to take care of state sharing between replicated stateful services. A stateful service is one where the server needs to keep a state of the ongoing communication with its clients as opposed to a service where the server receives all the information that is required to perform its task with each incoming request. The main focus of the replication manager is on providing state replication for services provided by nodes in the ad-hoc domain.

Operations:

Name of the operation `DistributeStateInformation`

Return type

Argument list

Description Distribute the state information from the primary/local server to the backup/remote replicated servers.

Name of the operation `FormLogicalGroups`

Return type

Argument list

Description Forms logical groups of redundant servers (based on a cost function and a selection algorithm) into server pools.

Name of the operation `SelectFailoverServer`

Return type

Argument list

Description Select one backup server to take over when the primary server crashes. The selected server should have up-to-date state information.

3.7.4 Integration of Application Interface Specification services (façade abstraction)

The HIDENETS middleware provides a wide range of services. However, they are not derived from standardized services in order to fully conform to the requirements of the HIDENETS environment. On the one hand, this avoids the constraints of a standardized implementation and allows creating optimally efficient applications; on the other hand, it reduces the portability of complete solutions.

The Application Interface Specification (AIS) includes a set of standardized interfaces for services that are commonly used in HA middleware implementations. In HIDENETS we use a selected set of these interfaces for providing the middleware services in a standardized way.

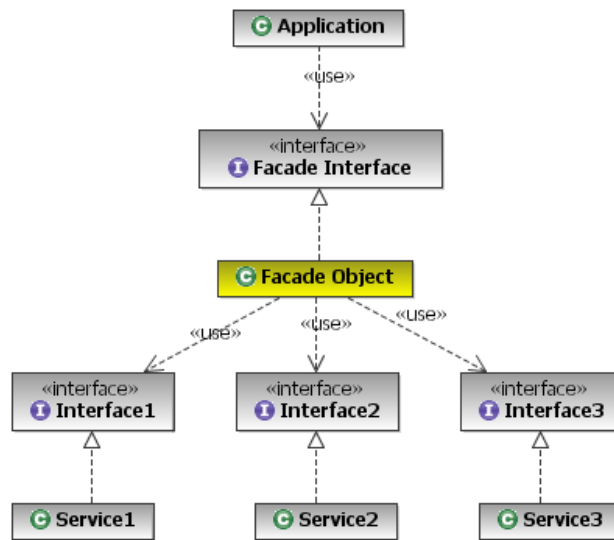


Figure 41: Façade object wrapping three service interfaces

A *service façade* is an object that provides a well-designed interface to wrap a collection of interfaces and includes lightweight business logic to describe how the provided operations can be carried out by using the operations of the wrapped interfaces (see Figure 42). The selected AIS services are implemented as façade objects and provide the different functionalities by interoperating with the existing HIDENETS services.

In the metamodel each AIS service is presented by a particular class and a corresponding interface. In order to distinguish the AIS façade classes from the HIDENETS services they are derived from the Façade Abstraction meta-class rather than the Service Abstraction. The hierarchy of the AIS façade classes is depicted in Figure 42.

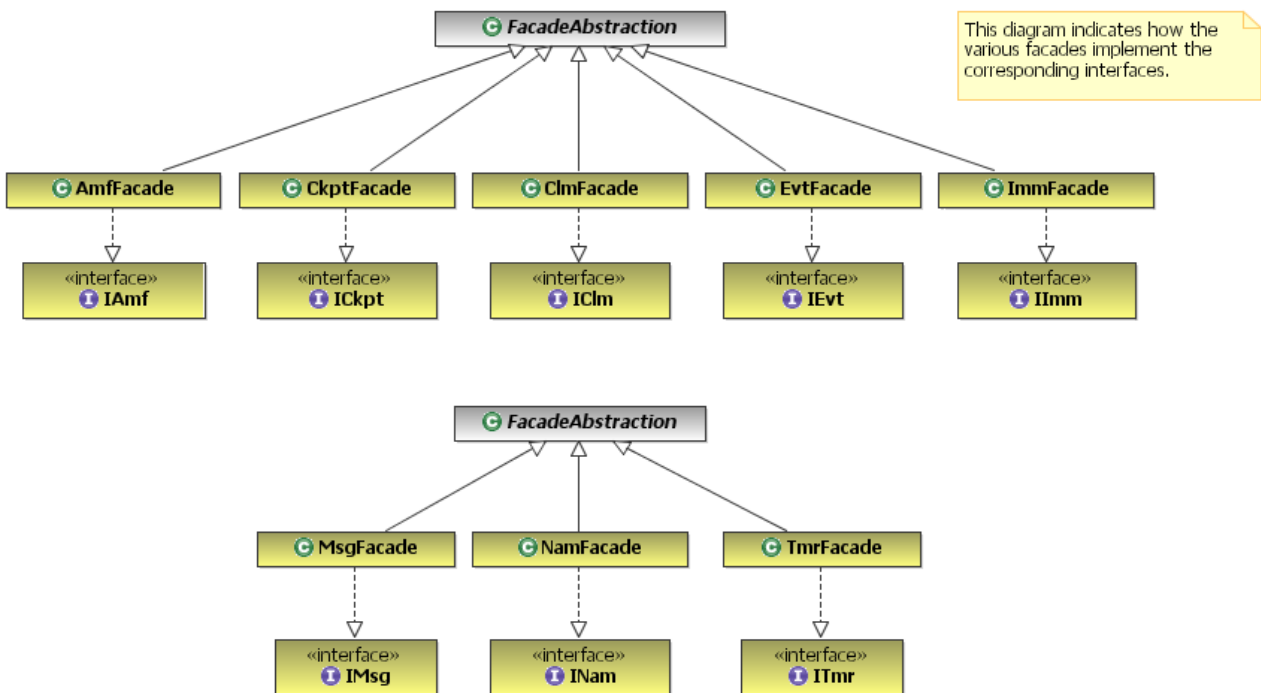


Figure 42: Application Interface Specification façade services and the corresponding interfaces

These façade services provide the functionalities they expose to clients by interoperating with the other HIDENETS specific services. E.g. the CImFacade, that implements the CLM interface of AIS, interoperates with the ProximityMapService and the GroupMembershipService and provides the functionalities of the CLM service by using the operations of these services. The dependencies of the different HIDENETS services and the AIS service façade classes are depicted in Figure 43.

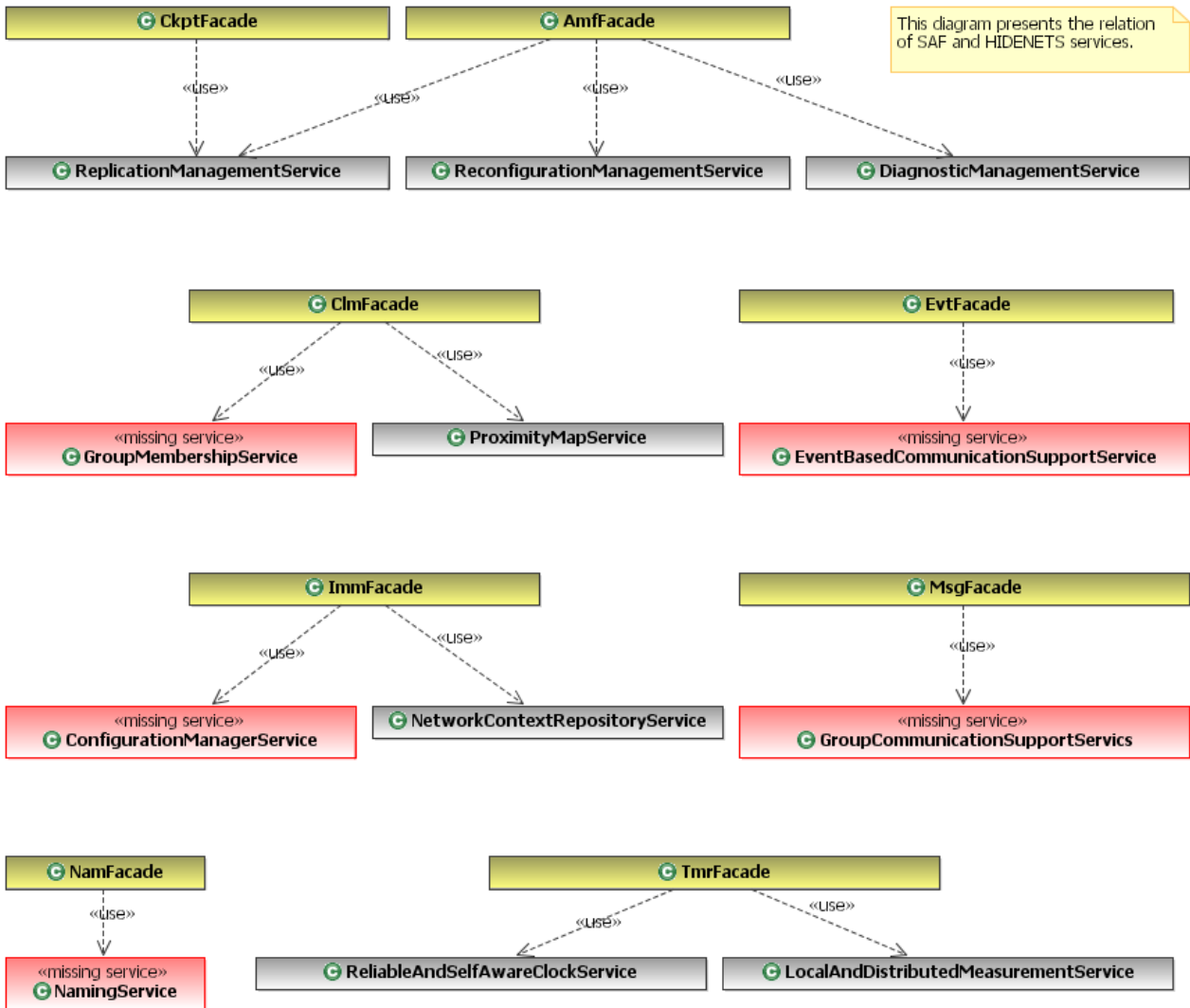


Figure 43: Interoperation of AIS façade services and HIDENETS services

3.7.5 Missing services

During the development of the metamodel we found that there are services that have not been defined in the HIDENETS architecture yet, however, they are required for providing particular functionalities for applications. These services are enumerated in Figure 44. In the current document we handle these services as if they had a description in the HIDENETS architecture.

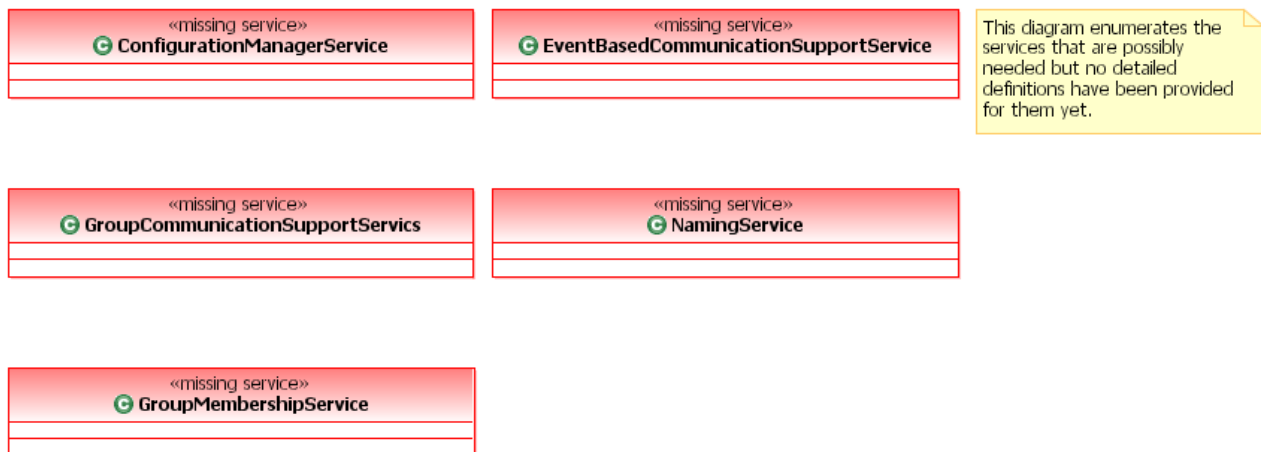


Figure 44: Services that still not have a description in the HIDENETS architecture

3.8 Services defined by WP3

3.8.1 Overview

Below we present the metaclasses corresponding to the services defined in WP3.

3.8.2 Abstract syntax

The metaclasses representing high-level WP3 services are shown in Figure 45. The co-operation of these services (i.e., how WP3 services use other WP3 services) is shown in Figure 46. The interoperation between WP3 and WP2 services (i.e., how WP3 services use WP2 services) is presented in Figure 47.

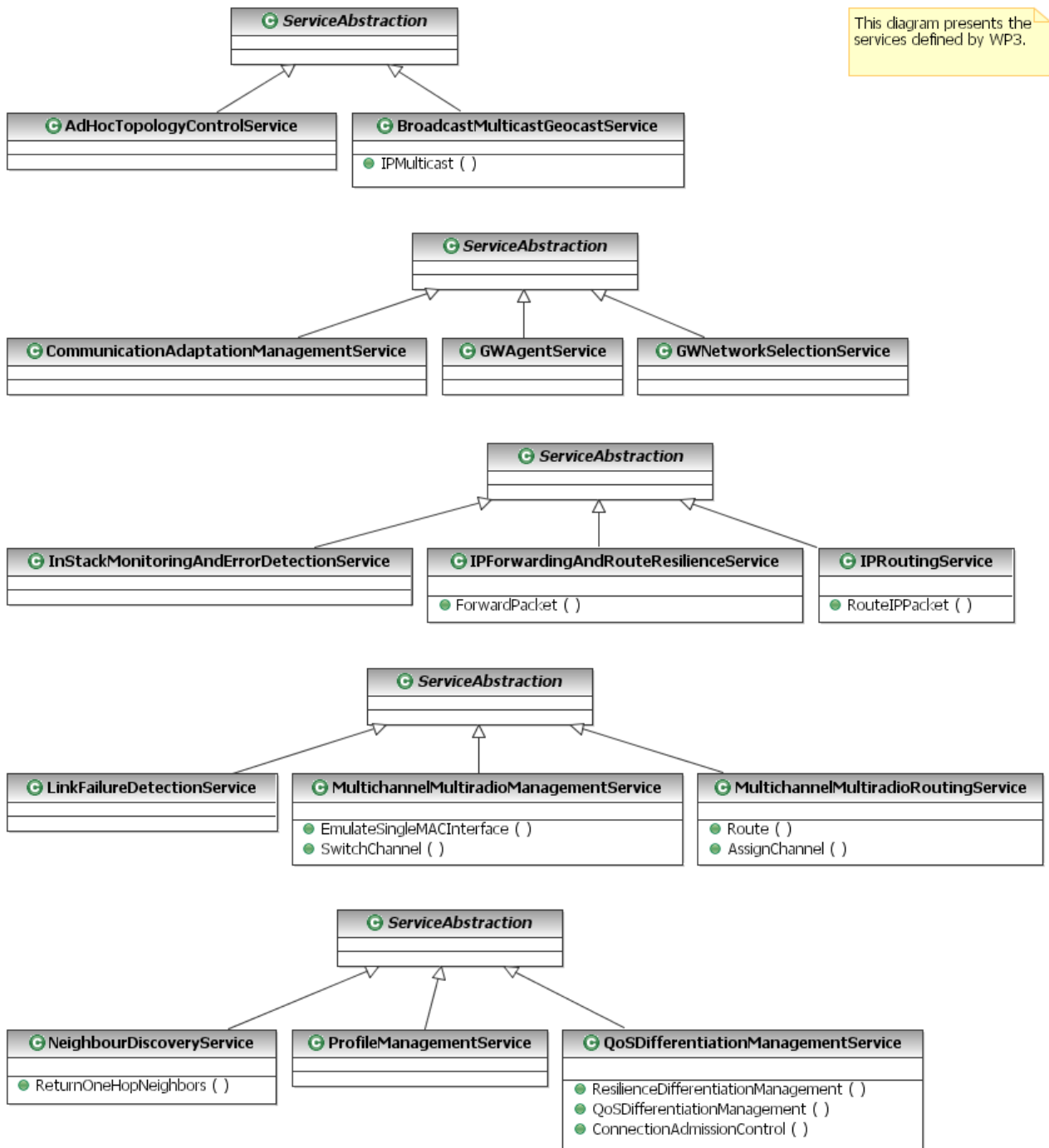


Figure 45: WP3 services

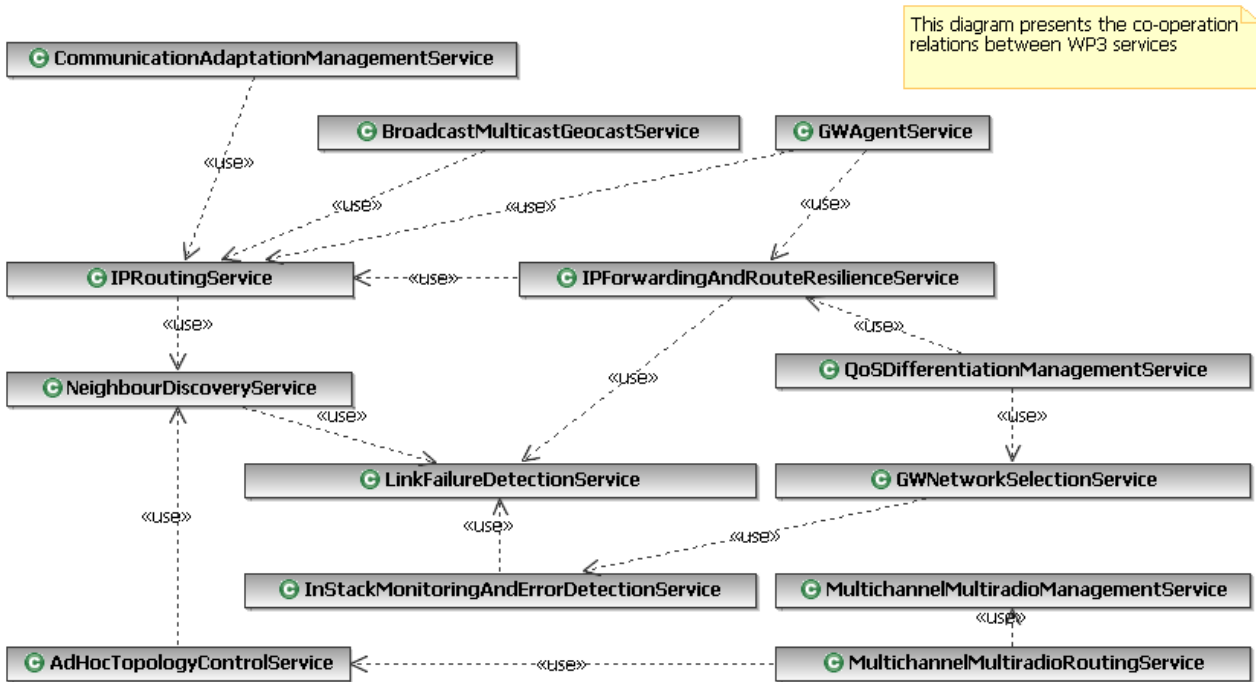


Figure 46: Co-operation of WP3 services

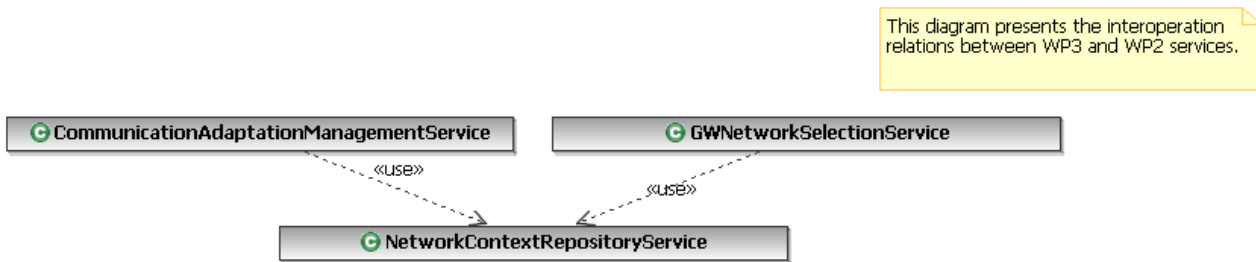


Figure 47: Interoperation of WP3 and WP2 services

3.8.3 Class descriptions

3.8.3.1 Metaclass Ad-hoc Topology Control Service

This building block computes and maintains a connected topology among the ad-hoc network nodes by selecting a subset of links out of the complete topology.

Operations:

No operation defined so far.

3.8.3.2 Metaclass Broadcast Multicast Geocast Service

IP Multicast is an IP layer service for packet replication in routers inside the network. The IP Multicast protocol suite consists of several protocols. PIM-SM (Protocol Independent Multicast Sparse Mode) and PIM-SSM (PIM Source-Specific Multicast) are the most relevant protocols for managing the multicast replication trees inside the network. In addition there is a protocol for signalling interest to receive traffic for specific multicast groups named IGMP (Internet Group

Management Protocol). The protocols provide means for routers and end systems for building and managing the replication tree between the senders and the receivers.

Operations:

No operation defined so far.

3.8.3.3 Metaclass Communication Adaptation Management Service

This building block coordinates the potential adaptation schemes at various layers. By monitoring certain parameters, it could determine the best values for the current situation, and adjust the behaviour of various protocols.

Operations:

No operation defined so far.

3.8.3.4 Metaclass GW Agent Service

The gateway agent ensures connectivity between ad-hoc and infrastructure networks using the preferred GW/network.

Operations:

No operation defined so far.

3.8.3.5 Metaclass GW Network Selection Service

The component takes a policy description as input that describes the rules for selecting a gateway/network. This description can be simple or complex, depending on the specifics of the implementation. Based on these rules the component will, at any time, evaluate which gateway/network currently is the best match. When the best match is not the current Gateway/network, it will initiate a switchover.

Operations:

No operation defined so far.

3.8.3.6 Metaclass In-Stack Monitoring and Error Detection Service

This building block handles the monitoring and error detection inside the protocol stack, from layer 2, up to layer 5 (in case a session layer protocol such as SIP is implemented in the protocol stack). At this stage of its specifications, this module only deals with remote monitoring and faults.

Operations:

No operation defined so far.

3.8.3.7 Metaclass IP Forwarding and Route Resilience Service

This module handles forwarding of packets on a hop-by-hop basis. 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.

Operations:

No operation defined so far.

3.8.3.8 Metaclass IP Routing Service

The main targeting problem of the IP Routing service is to find routes, and particularly next hops, to relevant destinations. The service achieves this goal by exchanging routing packets throughout the ad hoc network, to establish routes throughout the network.

Operations:

No operation defined so far.

3.8.3.9 Metaclass Link Failure Detection Service

The module monitors the link for periodic messages like hello messages and general degradation of quality. Link failure detection at layers 2 and 3 is an in-stack monitoring function that deals with failures in the point-to-point neighbour association. This may include loss of signal, loss of hello messages, degradation in signal quality, packet loss, bit errors etc. It is used directly by the layer 2 and 3 protocols to handle these important error situations. It detects packet loss/errors between neighbour nodes on layer 2 and 3.

Operations:

No operation defined so far.

3.8.3.10 Metaclass Multichannel Multiradio Management Service

Although by using multiple frequency channels for WLAN communication can significantly increase performance, most existing solutions use only one channel. Thus when using multiple channels, a multi-channel management service is needed.

Operations:

Name of the operation EmulateSingleMACInterface

Return type

Argument list

Description Manage the network traffic on multiple radio interfaces, i.e. form an interface between the radio interfaces and the network layer. This includes channel switching and assignment when no multichannel routing is used.

Name of the operation SwitchChannel

Return type

Argument list

Description Performs the channel switching of WLAN radios, only exposed in the case of multichannel/multiradio routing.

3.8.3.11 Metaclass Multichannel Multiradio Routing Service

Routing determines the route through a network for communication, plays an important role in load-balancing and can also influence the tolerance of the network against node failures by using alternative paths.

Operations:

Name of the operation AssignChannel

Return type

Argument list

Description Decides on the channel assignment of an interface.

Name of the operation Route

Return type

Argument list

Description Routing or meshing (depending on the implementation), functionality is equal to normal routing/meshing, but multiple channels are used to enhance performance, so see IP routing component for details.

3.8.3.12 Metaclass Neighbour Discovery Service

The Neighbour Discovery service checks bidirectional connectivity with surrounding nodes and their capabilities by exchanging hello packets with immediate next-hop neighbours.

Operations:

Name of the operation ReturnOneHopNeighbors

Return type

Argument list

Description Returns the neighbours that are accessible through a single hop path.

3.8.3.13 Metaclass Profile Management Service

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. This module manages the configuration, update and dissemination of the profile. This profile might be configured automatically based on context and location.

Operations:

No operation defined so far.

3.8.3.14 Metaclass QoS Differentiation Management Service

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

Operations:

Name of the operation ConnectionAdmissionControl

Return type

Argument list

Description Used to decide if a connection / flow may get access to certain resources (bandwidth).

Name of the operation QoSDifferentiationManagement

Return type

Argument list

Description Used to decide the relative time priority between packets regarding packet handling in the node.

Name of the operation ResilienceDifferentiationManagement

Return type

Argument list

Description Used to pre-empt or change priority/QoS class of connections, sessions or flows in case of failure.

3.9 Application interfaces

3.9.1 Overview

The implementation of the HIDENETS services can be different on each node. However, a HIDENETS application must be able to run on any HIDENETS capable node. In order to ensure this, uniform service interfaces are provided for the applications for accessing the services of a HIDNENETS node.

In the following sections we present the interfaces that the application uses to access the services of the middleware and the primitives that are used to build up a HIDENETS application in the form of a UML profile.

3.9.2 Interfaces of WP2 services and oracles

Figure 48 shows the interfaces of the WP2 oracles while Figure 49 shows interfaces of WP2 services.



Figure 48: Interfaces of WP2 oracle services

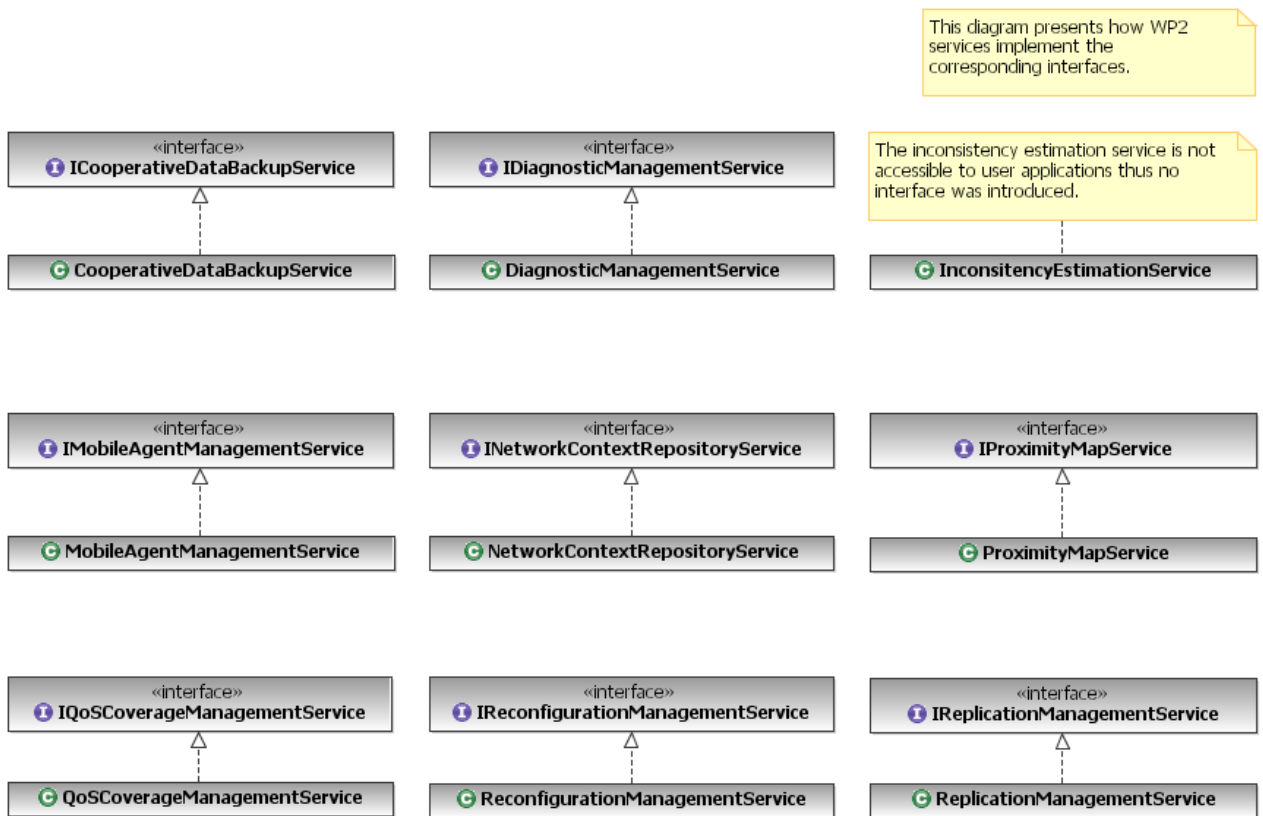


Figure 49: Interfaces of WP2 oracle services

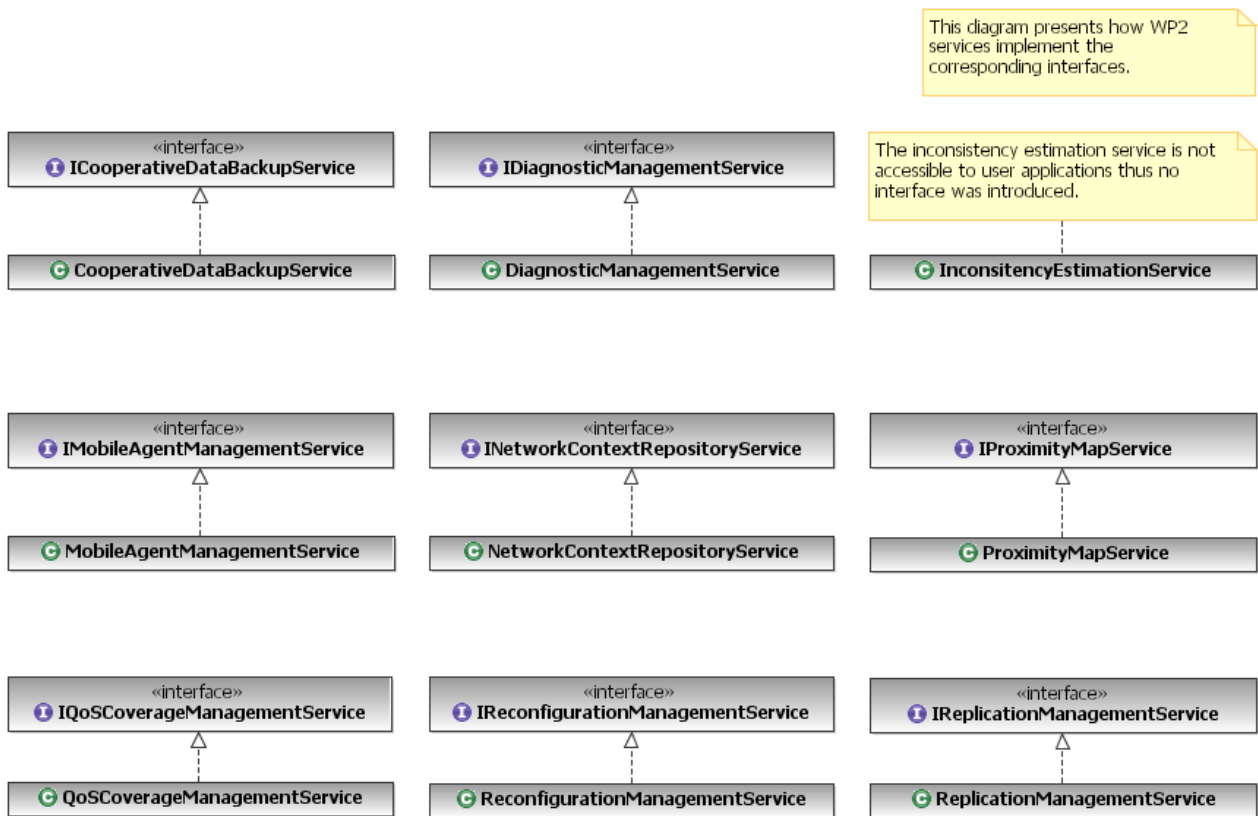


Figure 50: Interfaces of WP2 services

3.9.3 Interfaces of WP3 services

The interfaces of WP3 services are shown in Figure 51.



Figure 51: Interfaces of WP3 services

3.9.4 Interfaces for AIS façade services

The interfaces of the AIS façade services are shown in Figure 52.

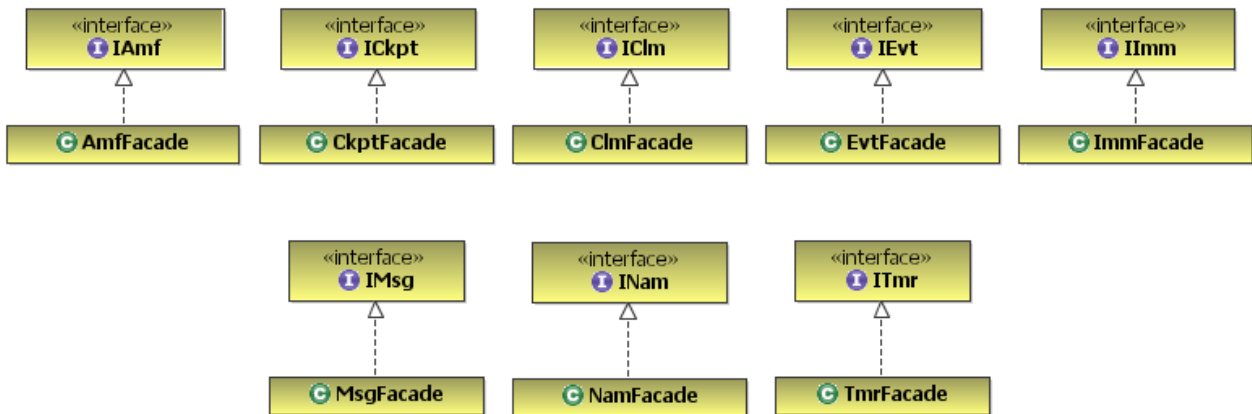


Figure 52: Interfaces of AIS façade services

The description of the interfaces can be found in the AIS specifications [AIS].

3.9.5 Interfaces for client applications of the Service Availability Forum façade services

The AIS client application interfaces are shown in Figure 53. These interfaces are needed to be implemented by the applications that are using the corresponding AIS services provided by the HIDENETS node. They contain the different callback functions that are called by the middleware.

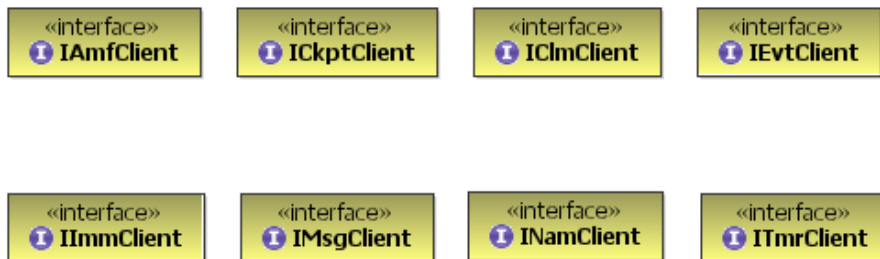


Figure 53: AIS client application interfaces

4 Application design support

The HIDENETS metamodel provides support for describing the configuration of a HIDENETS node and handle the dependencies of the different services, however, there is still lack of support for application development.

In this section we present two different techniques to facilitate the application development. First we introduce a partial UML profile that contains notations for describing HIDENETS applications and to represent their dependability needs. Next we show why design patterns are useful for application development.

4.1 The UML profile for modelling HIDENETS applications

Below we present the UML profile developed for modelling HIDENETS applications. The profile consists of two main parts. First we connect HIDENETS services and nodes to UML concepts by (i) deriving the HIDENETS *service* abstraction concept from the UML *Component* metaclass and (ii) deriving the HIDENETS *physical node* concept from the UML *Device* metaclass. The second part of the profile introduces various metaclasses aimed at supporting the application developer to indicate the HIDENETS-specific relevance of application-level artifacts (e.g., indicating that a communication link is expected to provide fresh data, specifying the required minimal freshness and the freshness detector service implementation allocated for the channel).

Both parts introduce first the referenced or newly introduced metaclasses, then define the corresponding stereotypes and tags.

4.1.1 Representing HIDENETS-specific artifacts in UML models

4.1.1.1 Overview

The first part of our profile focuses on the logical connection of HIDENETS-specific artifacts (i.e., metaclasses representing oracles and higher-level services) to UML concepts. We will derive our service metaclasses from the UML component concept and the resource notion of the General Resource Modelling framework.

4.1.1.2 Abstract syntax

Figure 54 presents the key inheritance relation between the UML metaclasses referenced in the HIDENETS profile and indicates how the HIDENETS service and physical node abstraction is derived from the corresponding UML ancestor metaclasses. (Note that the description of the UML metaclasses can be found in [UML2.0])

This diagram presents the most important inheritance relations between the UML metaclasses referenced by the HIDENETS profile and the introduction of HIDENETS-specific metaclasses as descendants of UML metaclasses.

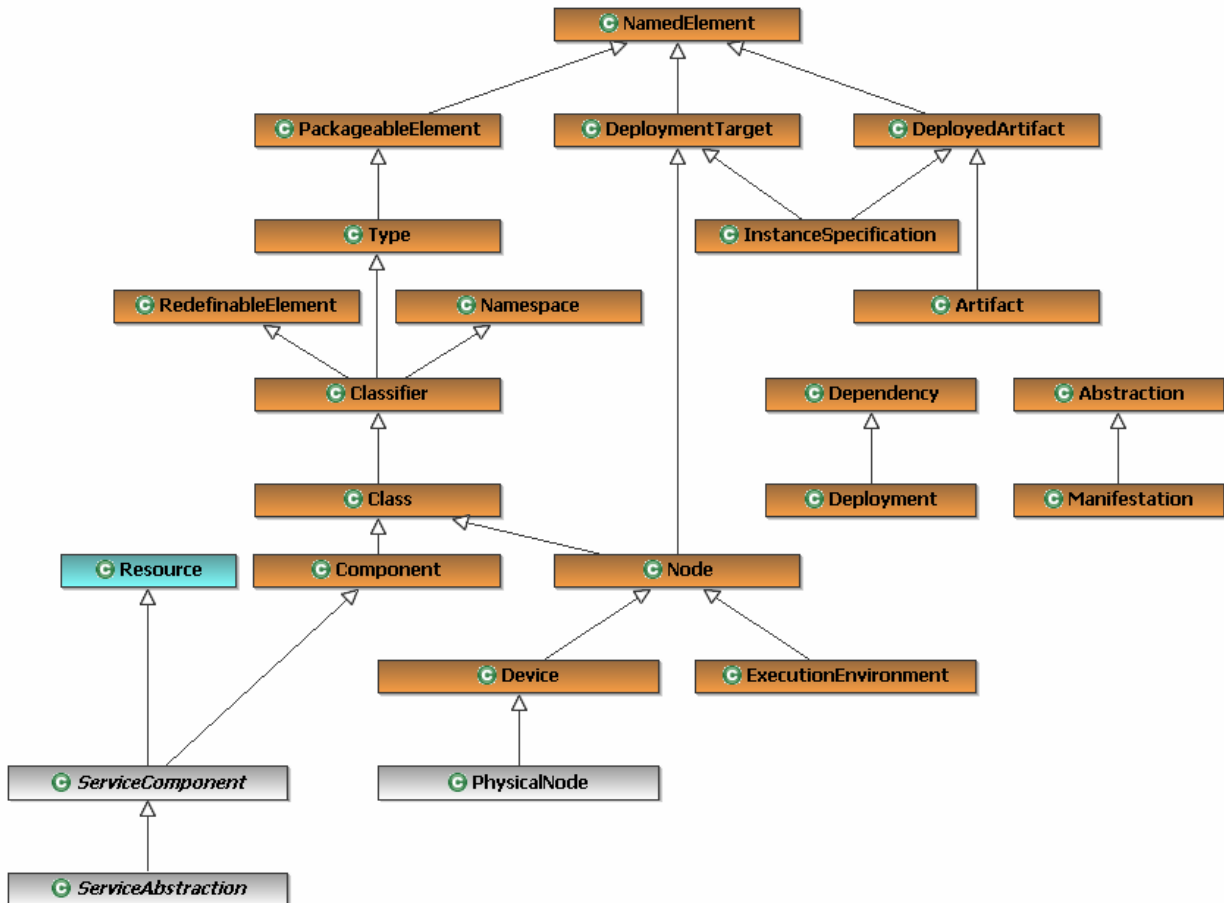


Figure 54: Key inheritance relations

Figure 55 highlights that having derived the HIDENETS physical node concept from the UML device abstraction and the HIDENETS service concept from the UML component abstraction, the standard deployment notation is applicable i.e., the modeller can unambiguously indicate which HIDENETS services are to be deployed to which HIDENETS physical nodes.

This diagram indicates how HIDENETS services are deployed to HIDENETS nodes (i.e., the navigation between the physical node and the service).

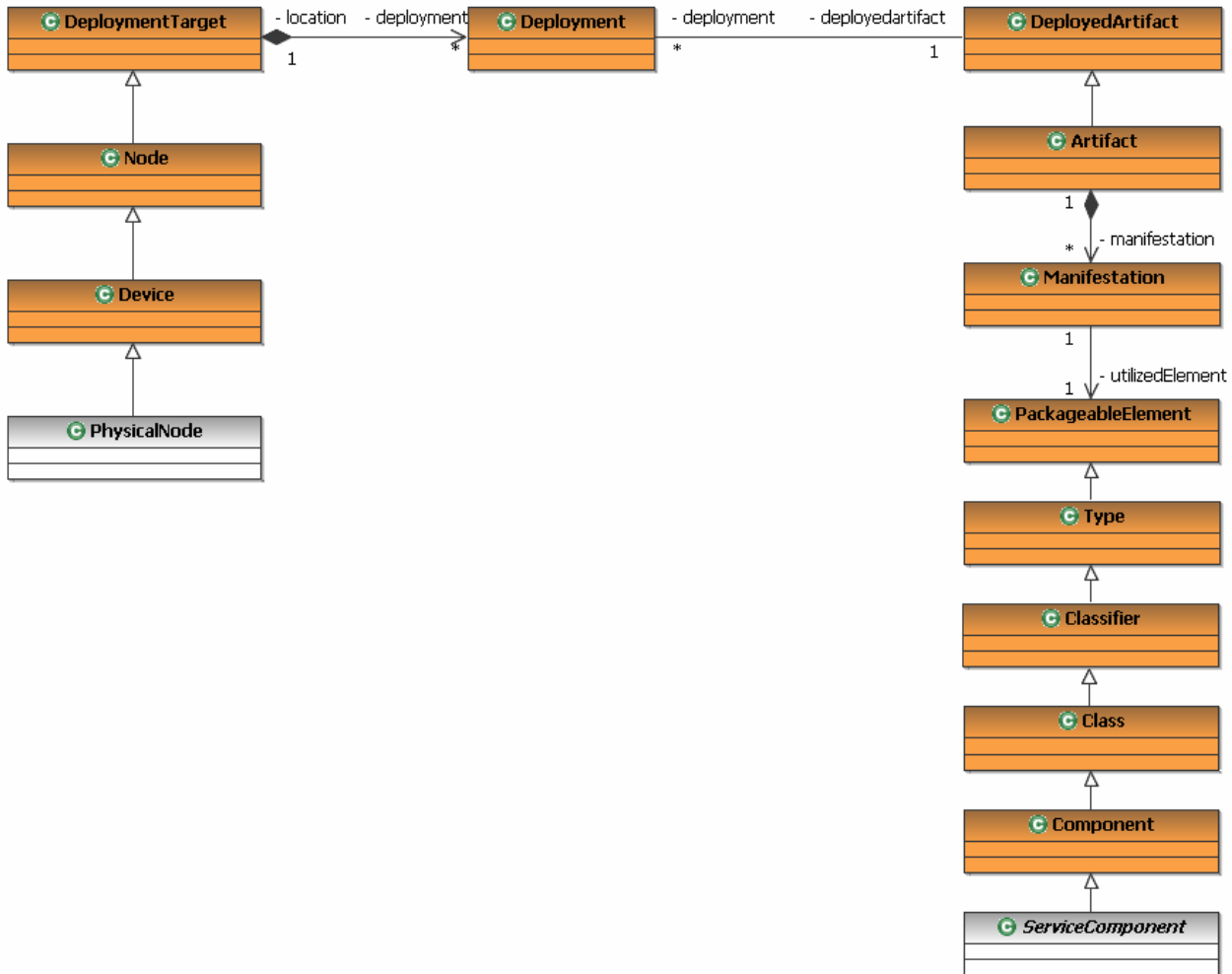


Figure 55: Deployment of HIDENETS services

4.1.1.3 HIDENETS class descriptions (metaclasses connecting the HIDENETS profile to UML)

Below we present the short descriptions of metaclasses introduced by us to semantically connect UML and HIDENETS concepts.

4.1.1.3.1 Metaclass Physical Node

A computational resource on which HIDENETS services can be deployed.

4.1.1.3.2 Metaclass Service Component

A specialization of the UML Component abstraction representing HIDENETS services (i.e., WP2 oracles and higher-level services and WP3 services).

4.1.1.4 Stereotypes and tags

In order to reduce the possibility of name clashes, all HIDENETS-specific stereotypes are prefixed by “hi”; otherwise the naming convention is very straightforward: the stereotype representing metaclass HidenetsMetaclass is hiHidenetsMetaclass (i.e., the stereotype is practically the name of the metaclass introduced by us after a “hi” prefix). A tag corresponding to the attribute HidenetsMetaclass::attribute is “hiAttribute”.

4.1.1.4.1 Stereotypes and tags corresponding to HIDENETS services

4.1.1.4.1.1 WP2 oracles

Stereotype	<<hiAuthenticationService>>
Represented HIDENETS metaclass	AuthenticationService
Base UML metaclass	Component
Short description	The component marked with this stereotype is an implementation of the AuthenticationService HIDENETS service.
Stereotype	<<hiFreshnessDetectionService>>
Represented HIDENETS metaclass	FreshnessDetectionService
Base UML metaclass	Component
Short description	The component marked with this stereotype is an implementation of the FreshnessDetectionService HIDENETS service.
Stereotype	<<hiLocalAndDistributedMeasurementService>>
Represented HIDENETS metaclass	LocalAndDistributedMeasurementService
Base UML metaclass	Component
Short description	The component marked with this stereotype is an implementation of the LocalAndDistributedMeasurementService HIDENETS service.
Stereotype	<<hiReliableAndSelfAwareClockService>>
Represented HIDENETS metaclass	ReliableAndSelfAwareClockService
Base UML metaclass	Component
Short description	The component marked with this stereotype is an implementation of the ReliableAndSelfAwareClockService HIDENETS service.
Stereotype	<<hiTimelyTimingFailureDetectionService>>

Represented HIDENETS metaclass TimelyTimingFailureDetectionService
 Base UML metaclass Component
 Short description The component marked with this stereotype is an implementation of the TimelyTimingFailureDetectionService HIDENETS service.

Stereotype <<hiTrustAndCooperationService>>

Represented HIDENETS metaclass TrustAndCooperationService

Base UML metaclass Component

Short description The component marked with this stereotype is an implementation of the TrustAndCooperationService HIDENETS service.

4.1.1.4.1.2 WP2 high-level services

Stereotype <<hiCooperativeDataBackupService>>

Represented HIDENETS metaclass CooperativeDataBackupService

Base UML metaclass Component

Short description The component marked with this stereotype is an implementation of the CooperativeDataBackupService HIDENETS service.

Stereotype <<hiDiagnosticManagementService>>

Represented HIDENETS metaclass DiagnosticManagementService

Base UML metaclass Component

Short description The component marked with this stereotype is an implementation of the DiagnosticManagementService HIDENETS service.

Stereotype <<hiInconsistencyEstimationService>>

Represented HIDENETS metaclass InconsistencyEstimationService

Base UML metaclass Component

Short description The component marked with this stereotype is an implementation of the InconsistencyEstimationService HIDENETS service.

Stereotype <<hiMobileAgentManagementService>>

Represented HIDENETS metaclass MobileAgentManagementService

Base UML metaclass	Component
Short description	The component marked with this stereotype is an implementation of the MobileAgentManagementService HIDENETS service.
Stereotype	<<hiNetworkContextRepositoryService>>
Represented HIDENETS metaclass	NetworkContextRepositoryService
Base UML metaclass	Component
Short description	The component marked with this stereotype is an implementation of the NetworkContextRepositoryService HIDENETS service.
Stereotype	<<hiProximityMapService>>
Represented HIDENETS metaclass	ProximityMapService
Base UML metaclass	Component
Short description	The component marked with this stereotype is an implementation of the ProximityMapService HIDENETS service.
Stereotype	<<hiQoScoverageManagementService>>
Represented HIDENETS metaclass	QoScoverageManagementService
Base UML metaclass	Component
Short description	The component marked with this stereotype is an implementation of the QoScoverageManagementService HIDENETS service.
Stereotype	<<hiReconfigurationManagementService>>
Represented HIDENETS metaclass	ReconfigurationManagementService
Base UML metaclass	Component
Short description	The component marked with this stereotype is an implementation of the ReconfigurationManagementService HIDENETS service.
Stereotype	<<hiReplicationManagementService>>
Represented HIDENETS metaclass	ReplicationManagementService
Base UML metaclass	Component
Short description	The component marked with this stereotype is an implementation of the ReplicationManagementService

HIDENETS service.

4.1.1.4.1.3 WP3 Services

Stereotype <<hiAdHocTopologyControlService>>
 Represented HIDENETS metaclass AdHocTopologyControlService
 Base UML metaclass Component
 Short description The component marked with this stereotype is an implementation of the AdHocTopologyControlService HIDENETS service.

Stereotype <<hiBroadcastMulticastGeocastService>>
 Represented HIDENETS metaclass BroadcastMulticastGeocastService
 Base UML metaclass Component
 Short description The component marked with this stereotype is an implementation of the BroadcastMulticastGeocastService HIDENETS service.

Stereotype <<hiCommunicationAdaptationManagementService>>
 Represented HIDENETS metaclass CommunicationAdaptationManagementService
 Base UML metaclass Component
 Short description The component marked with this stereotype is an implementation of the CommunicationAdaptationManagementService HIDENETS service.

Stereotype <<hiGWAgentService>>
 Represented HIDENETS metaclass GWAgentService
 Base UML metaclass Component
 Short description The component marked with this stereotype is an implementation of the GWAgentService HIDENETS service.

Stereotype <<hiGWNetworkSelectionService>>
 Represented HIDENETS metaclass GWNetworkSelectionService
 Base UML metaclass Component
 Short description The component marked with this stereotype is an implementation of the GWNetworkSelectionService HIDENETS service.

Stereotype	<<hiInStackMonitoringAndErrorDetectionService>>
Represented HIDENETS metaclass	InStackMonitoringAndErrorDetectionService
Base UML metaclass	Component
Short description	The component marked with this stereotype is an implementation of the InStackMonitoringAndErrorDetectionService HIDENETS service.
Stereotype	<<hiIPForwardingAndRouteResilienceService>>
Represented HIDENETS metaclass	IPForwardingAndRouteResilienceService
Base UML metaclass	Component
Short description	The component marked with this stereotype is an implementation of the IPForwardingAndRouteResilienceService HIDENETS service.
Stereotype	<<hiIPRoutingService>>
Represented HIDENETS metaclass	IPRoutingService
Base UML metaclass	Component
Short description	The component marked with this stereotype is an implementation of the IPRoutingService HIDENETS service.
Stereotype	<<hiLinkFailureDetectionService>>
Represented HIDENETS metaclass	LinkFailureDetectionService
Base UML metaclass	Component
Short description	The component marked with this stereotype is an implementation of the LinkFailureDetectionService HIDENETS service.
Stereotype	<<hiMultichannelMultiradioManagementService>>
Represented HIDENETS metaclass	MultichannelMultiradioManagementService
Base UML metaclass	Component
Short description	The component marked with this stereotype is an implementation of the MultichannelMultiradioManagementService HIDENETS service.
Stereotype	<<hiMultichannelMultiradioRoutingService>>

Represented HIDDENETS metaclass	MultichannelMultiradioRoutingService
Base UML metaclass	Component
Short description	The component marked with this stereotype is an implementation of the MultichannelMultiradioRoutingService HIDDENETS service.
Stereotype	<<hiNeighbourDiscoveryService>>
Represented HIDDENETS metaclass	NeighbourDiscoveryService
Base UML metaclass	Component
Short description	The component marked with this stereotype is an implementation of the NeighbourDiscoveryService HIDDENETS service.
Stereotype	<<hiProfileManagementService>>
Represented HIDDENETS metaclass	ProfileManagementService
Base UML metaclass	Component
Short description	The component marked with this stereotype is an implementation of the ProfileManagementService HIDDENETS service.
Stereotype	<<hiQoSDifferentiationManagementService>>
Represented HIDDENETS metaclass	QoSDifferentiationManagementService
Base UML metaclass	Component
Short description	The component marked with this stereotype is an implementation of the QoSDifferentiationManagementService HIDDENETS service.

4.1.1.4.2 Stereotypes and tags corresponding to physical nodes

Stereotype	<<hiPhysicalNode>>
Represented HIDDENETS metaclass	PhysicalNode
Base UML metaclass	Device
Short description	The component marked with this stereotype is a HIDDENETS PhysicalNode.

4.1.1.5 Examples

Below we present some examples for using the HIDDENETS profile artifacts introduced in this first part of the profile.

4.1.1.5.1 Indicating components that implement HIDENETS services

The figures correspond to an imaginary development of a platooning software, where the required HIDENETS services are purchased as third party COTS components. According to the discussion of deliverable D2.1.1, the platooning software uses the (i) authentication, (ii) freshness detection, (iii) local and distributed measurement, (iv) reliable and self aware clock and (v) timely timing failure detection HIDENETS services; the implementations of the corresponding services are provided as COTS components combined in a package as presented in Figure 56. Note that components implementing HIDENETS services are indicated by the corresponding stereotype.

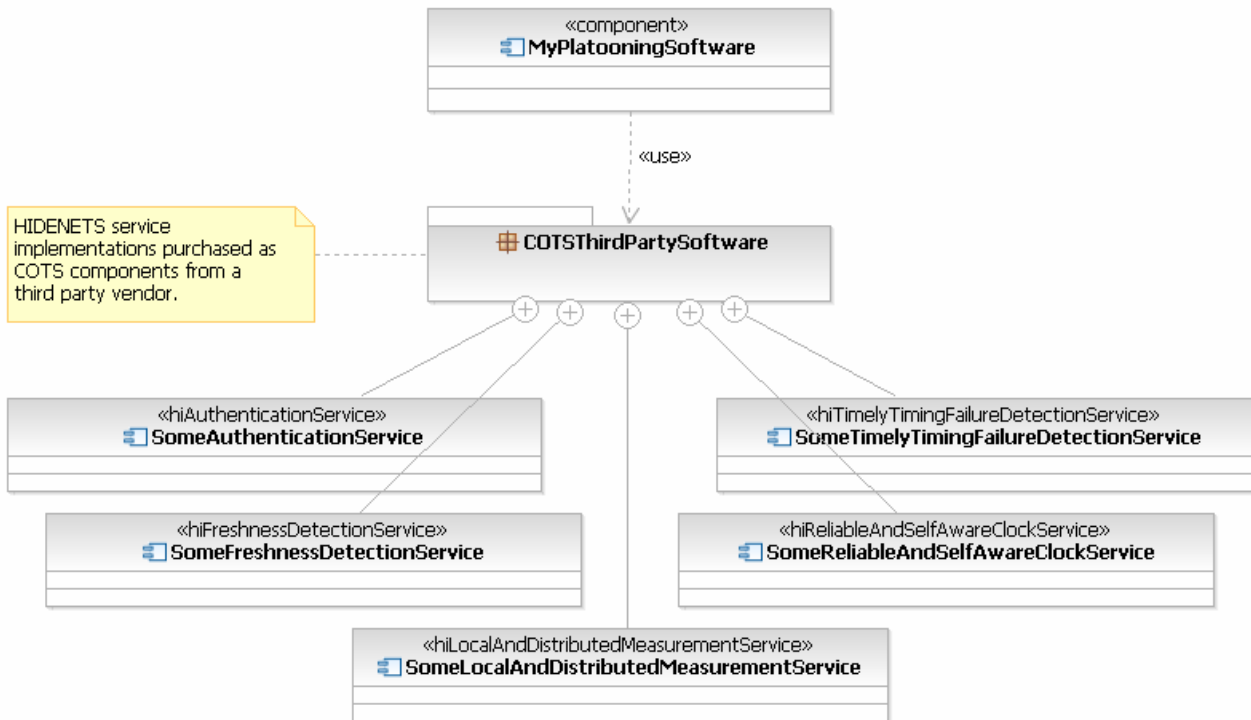


Figure 56: Indicating COTS components that implement some HIDENETS services

4.1.1.5.2 Resource usage modelling

Since HIDENETS services were derived from the *resource* concept of the General Resource Modelling framework, the fact that some application-level classes directly require the operations provided by some HIDENETS service implementations can be indicated by applying the *requires* GRM stereotype to an abstraction association. In Figure 57 an authentication-related application-level class requires the authentication HIDENETS service and a data processing-related one requires the local and distributed measurement and freshness detection HIDENETS services.

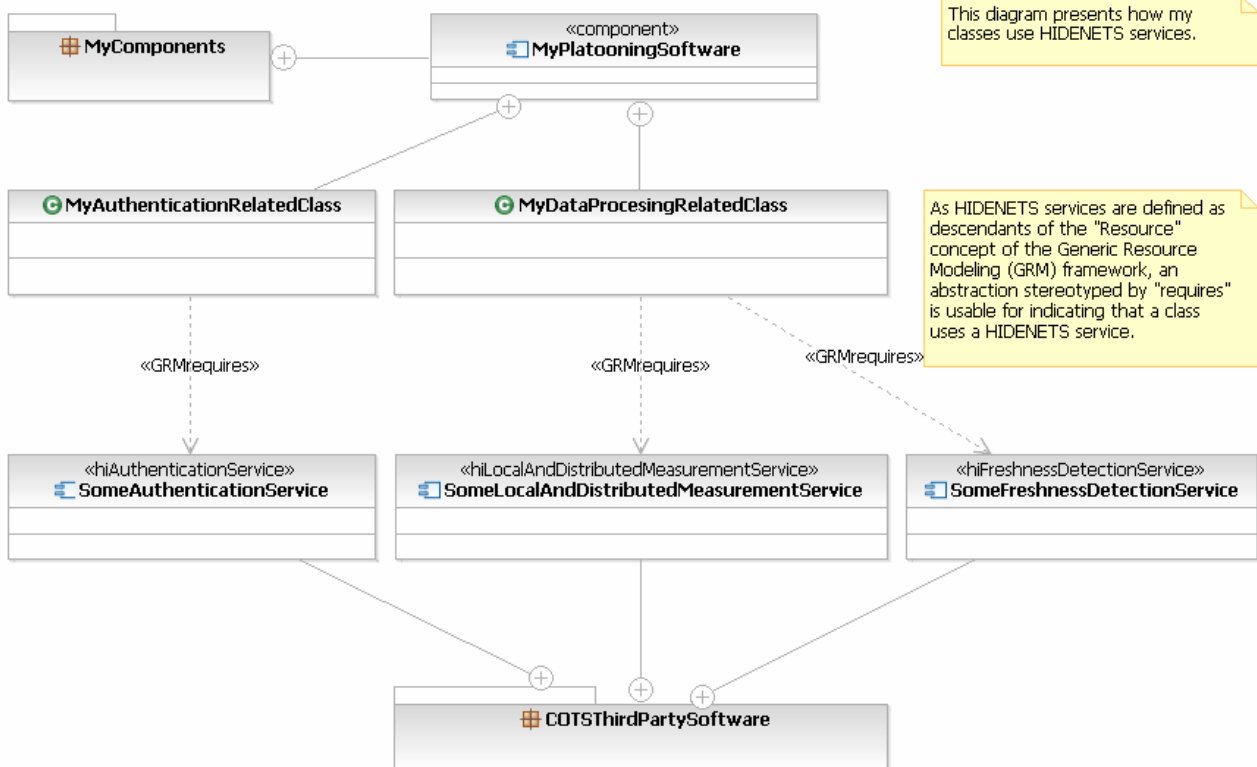


Figure 57: Modelling resource usage

4.1.1.5.3 Deployment

As HIDDENETS physical nodes were derived from the UML *device* concept, devices may be stereotyped to indicate their being HIDDENETS physical nodes and software artifacts can be deployed on them. In Figure 58 the first vehicle in the platoon is called *FirstVehicle*, on which two software artifacts are deployed: (i) the component implementing the user-level platooning application and (ii) the COTS components implementing the required HIDDENETS services.

This diagram presents how to deploy my platooning software to physical nodes.

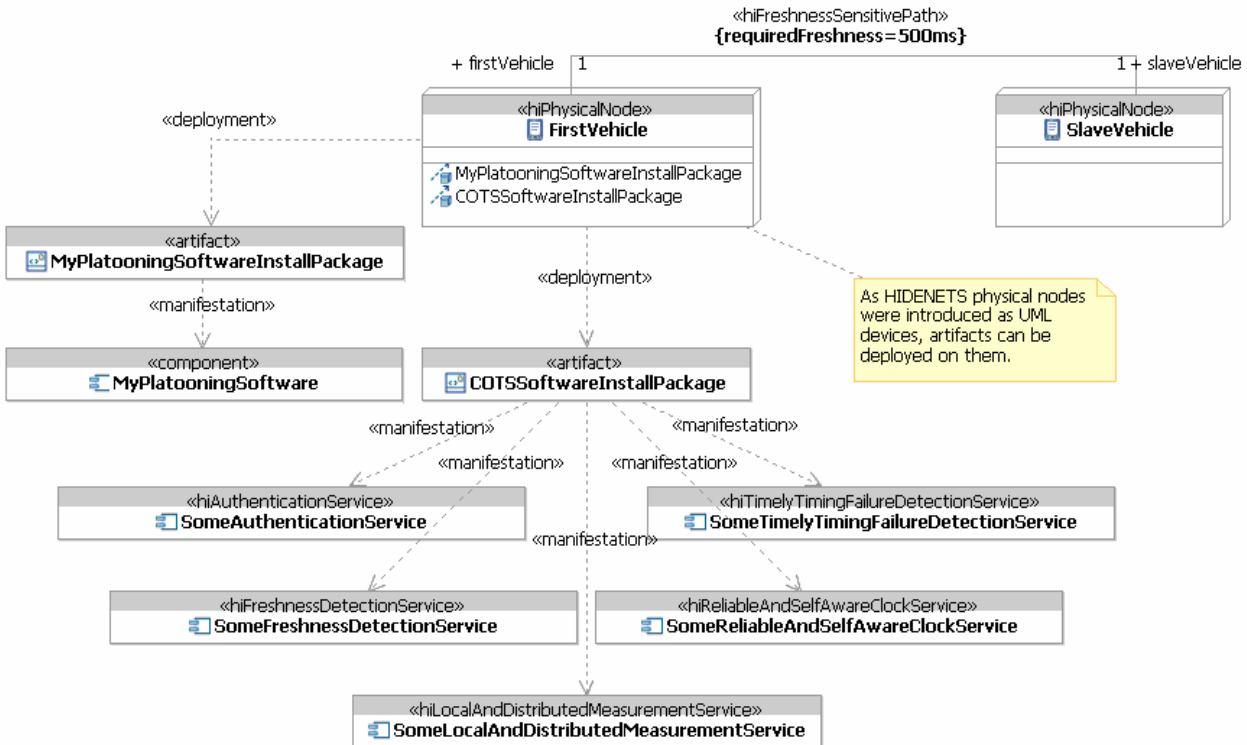


Figure 58: Application deployment to physical nodes

4.1.2 Utility concepts introduced for supporting application development

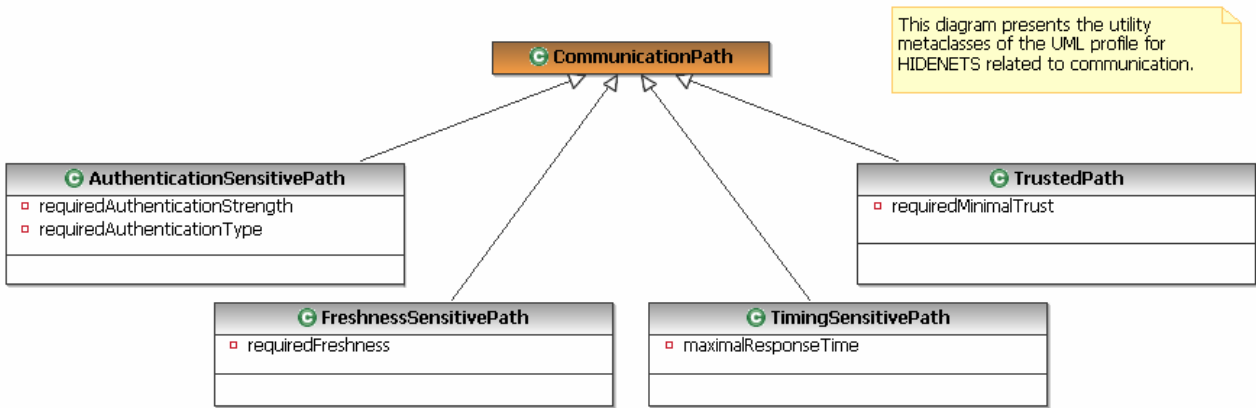
4.1.2.1 Overview

The second part of our profile introduces some utility metaclasses that provide support for the application modeller to include HIDENETS-related information in UML models. The three fields addressed here are related to (i) communication paths, (ii) data modelling and (iii) the replication and mobility of software components.

The utility metaclasses introduced here result from considering the application developers' possible needs for modelling applications running on the HIDENETS platform, thus the discussion below should be considered as a first proposal to be discussed and refined in the future based on the field experience and application development routine of the corresponding project partners.

4.1.2.2 Abstract syntax

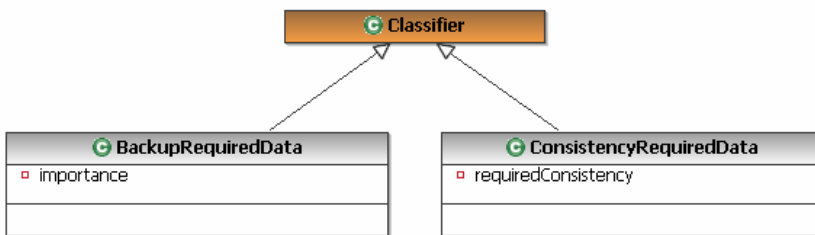
Figure 59 introduces four metaclasses representing communication paths with authentication, freshness, timing and trust requirements respectively. The introduction of these metaclasses enables the modeller to indicate the corresponding requirements about a communication path (e.g., by applying the corresponding stereotype to the path and specifying the required freshness as a tagged value).



This diagram presents the utility metaclasses of the UML profile for HIDENETS related to communication.

Figure 59: HIDENETS-related aspects of modelling communication paths

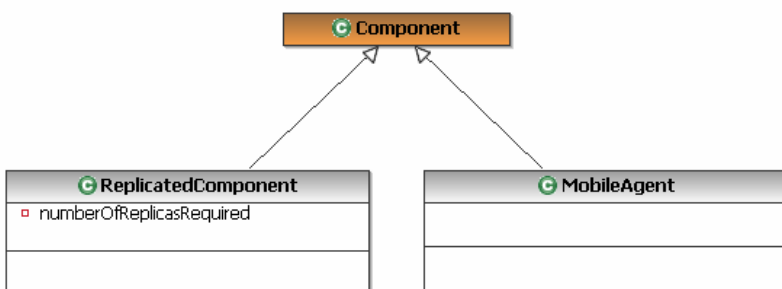
Figure 60 introduces two metaclasses indicating that (i) a specific data structure needs to be backed up in the infrastructure or (ii) some consistency requirements are posed against the corresponding data instances. The introduction of these metaclasses enables the modeller to highlight the corresponding requirements about a data structure (e.g., by applying the corresponding stereotype to the classifier and specifying the required consistency level).



This diagram presents the utility metaclasses of the UML profile for HIDENETS related to data modeling

Figure 60: HIDENETS-related aspects of data modelling

Figure 61 introduces two metaclasses that indicate that a software component is (i) replicated or (ii) the component is actually a mobile agent. The introduction of these metaclasses enables the modeller to clearly indicate mobile agents or to highlight the replication requirements about a component (e.g., by applying the corresponding stereotype to the component and specifying the required number of replicas).



This diagram presents the utility metaclasses of the UML profile for HIDENETS related to the replication and mobility of components

Figure 61: Modelling HIDENETS-related aspects of component replication and mobility

4.1.2.3 Class descriptions

Below we present the short descriptions of the utility metaclasses mentioned above.

4.1.2.3.1 Metaclass Authentication Sensitive Path

An authentication sensitive path is a communication path for which authentication requirements apply (e.g., because the peers must be sure about the identity of the communication partner). This metaclass enables the modeller to clearly indicate this requirement about a communication path and allows the definition of the required cryptographical strength of the authentication and the method used for authentication (e.g., some PKI implementation, etc.). Attributes:

- **requiredAuthenticationStrength:** Required cryptographical strength of the authentication. (The exact type of this attribute is to be elaborated in the future by the cooperation of the project partners involved in the HIDENETS authentication service.)
- **requiredAuthenticationType:** Required type of authentication (e.g., some PKI implementation etc.). (The exact type of this attribute is to be elaborated in the future by the cooperation of the project partners involved in the HIDENETS authentication service.)

4.1.2.3.2 Metaclass Backup Required Data

A backup required data is a piece of data that should be dependably backed up to the infrastructure (e.g., vehicle movement and diagnostic data recording the last some seconds before an accident). This metaclass enables the modeller to clearly indicate this kind of requirement about a data structure. Attributes:

- **importance:** Importance of the data (e.g., in case of a network or infrastructure overload the involved entities may provide precedence to saving highly important data first). (The exact type of this attribute is to be elaborated in the future by the cooperation of the project partners involved in the HIDENETS cooperative data backup service.)

4.1.2.3.3 Metaclass Consistency Required Data

A consistency required data is a piece of data for which consistency requirements apply (e.g., a computation has to be sure about the consistency of its inputs before performing some dependability-related activity). This metaclass enables the modeller to clearly indicate this kind of requirement about a data structure. Attributes:

- **requiredConsistency:** The required consistency of the data for being used in the computation. (The exact type of this attribute is to be elaborated in the future by the cooperation of the project partners involved in the HIDENETS inconsistency estimation service.)

4.1.2.3.4 Metaclass Freshness Sensitive Path

A freshness sensitive path is a communication path for which freshness requirements apply (e.g., because the peers must be sure that the information received on the channel is fresh enough for their purposes). This metaclass enables the modeller to clearly indicate this requirement about a communication path and allows the definition of the required freshness (e.g., maximal time elapsed between the production and the reception of the data). Attributes:

- **requiredFreshness:** Required freshness of information interchanged on the path. (The exact type of this attribute is to be elaborated in the future by the cooperation of the project partners involved in the HIDENETS freshness detection service.)

4.1.2.3.5 *Metaclass Mobile Agent*

A mobile agent is a piece of software that travels in the network. This metaclass enables the modeller to clearly indicate that a software component is actually a mobile agent.

4.1.2.3.6 *Metaclass Replicated Component*

A replicated component is a piece of software that is replicated on multiple physical nodes in order to achieve high service dependability. This metaclass enables the modeller to clearly indicate this requirement about a software component. Attributes:

- numberOfReplicasRequired: Number of required replicas of this component.

4.1.2.3.7 *Metaclass Timing Sensitive Path*

A timing sensitive path is a communication path for which timing requirements apply (e.g., because the peers must be sure that their communication partner will respond to their requests within a maximal time interval). This metaclass enables the modeller to clearly indicate this requirement about a communication path and allows the definition of the maximal acceptable response time. Attributes:

- maximalResponseTime: The acceptable maximal response time in the communication. (The exact type of this attribute is to be elaborated in the future by the cooperation of the project partners involved in the HIDENETS timely timing failure detection service.)

4.1.2.3.8 *Metaclass Trusted Path*

A trusted path is a communication path for which the communicating entities have to trust in their peers. This metaclass enables the modeller to clearly indicate this requirement about a communication path and allows the definition of the required minimal trust in the peers. Attributes:

- requiredMinimalTrust: The minimal level of trust required for communication. (The exact type of this attribute is to be elaborated in the future by the cooperation of the project partners involved in the HIDENETS trust and cooperation service.)

4.1.2.4 **Stereotypes and tags**

Stereotype	<<hiAuthenticationSensitivePath>>
Represented HIDENETS metaclass	AuthenticationSensitivePath
Base UML metaclass	CommunicationPath
Short description	A communication path for which authentication requirements apply.
Tags	
	<ul style="list-style-type: none"> • <<hiRequiredAuthenticationStrength>>: representing the attribute requiredAuthenticationStrength of metaclass AuthenticationSensitivePath. • <<hiRequiredAuthenticationType>>: representing the attribute requiredAuthenticationType of metaclass AuthenticationSensitivePath.

Stereotype	<<hiBackupRequiredData>>
------------	--------------------------

Represented HIDENETS metaclass	BackupRequiredData
Base UML metaclass	Classifier
Short description	A piece of data that should be dependably backed up to the infrastructure.
Tags	<ul style="list-style-type: none"> • <<hiImportance>>: representing the attribute importance of metaclass BackupRequiredData.
Stereotype	<<hiConsistencyRequiredData>>
Represented HIDENETS metaclass	ConsistencyRequiredData
Base UML metaclass	Classifier
Short description	A piece of data for which consistency requirements apply.
Tags	<ul style="list-style-type: none"> • <<hiRequiredConsistency>>: representing the attribute requiredConsistency of metaclass ConsistencyRequiredData.
Stereotype	<<hiFreshnessSensitivePath>>
Represented HIDENETS metaclass	FreshnessSensitivePath
Base UML metaclass	CommunicationPath
Short description	A communication path for which freshness requirements apply.
Tags	<ul style="list-style-type: none"> • <<hiRequiredFreshness>>: representing the attribute requiredFreshness of metaclass FreshnessSensitivePath.
Stereotype	<<hiMobileAgent>>
Represented HIDENETS metaclass	MobileAgent
Base UML metaclass	Component
Short description	A piece of software that travels in the network.
Stereotype	<<hiReplicatedComponent>>
Represented HIDENETS metaclass	ReplicatedComponent
Base UML metaclass	Component
Short description	A piece of software that is replicated on multiple physical nodes in order to achieve high service dependability.

Tags

- `<<hiNumberOfReplicasRequired>>`: representing the attribute `numberOfReplicasRequired` of metaclass `ReplicatedComponent`.

Stereotype	<code><<hiTimingSensitivePath>></code>
Represented HIDENETS metaclass	<code>TimingSensitivePath</code>
Base UML metaclass	<code>CommunicationPath</code>
Short description	A communication path for which timing requirements apply.

Tags

- `<<hiMaximalResponseTime>>`: representing the attribute `maximalResponseTime` of metaclass `TimingSensitivePath`.

Stereotype	<code><<hiTrustedPath>></code>
Represented HIDENETS metaclass	<code>TrustedPath</code>
Base UML metaclass	<code>CommunicationPath</code>
Short description	A communication path for which the communicating entities have to trust in their peers.

Tags

- `<<hiRequiredMinimalTrust>>`: representing the attribute `requiredMinimalTrust` of metaclass `TrustedPath`.

4.1.2.5 Example

Figure 62 presents an example diagram from the platooning software model used in previous examples where the modeller inserted a communication path between the first vehicle and another one in the platoon and specified that for safety reasons the information interchanged on the path should be fresh i.e., not older than 500ms.

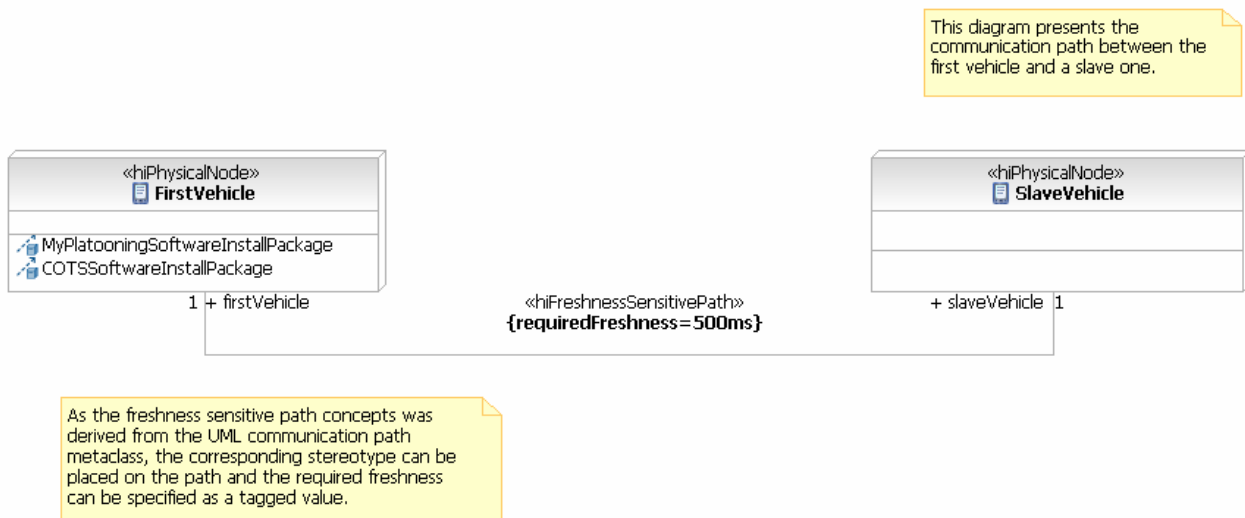


Figure 62: Indicating freshness requirements about a communication path

4.2 Dependability design patterns

In the following we describe some design patterns that are frequently used in software engineering and seem to be applicable in HIDENETS applications.

4.2.1 Application patterns

Below we present some architectural design patterns that are used for making applications fault tolerant. First we describe the data replication, next we deal with failover scenarios and last we describe the methods of application recovery. The reason for choosing these patterns is that these are the most frequently used ones and their application is straightforward in HIDENETS applications.

4.2.1.1 Replication

In the replication pattern multiple copies, replicas, are stored which are kept synchronous. Depending on which replica can be modified the following two scenarios can be distinguished:

- Multiple active replicas,
- Primary-backup configuration.

In the first case multiple replicas can be modified, thus the consistency of the replicas has to be guaranteed. Depending on the failure types to tolerate, several techniques exist, e.g. if also partitioning of replicas can occur then the quorum consensus method could be used. This pattern is used e.g. in distributed databases.

In the second case only one replica can be modified, and the changes are copied to the other replicas. If the modification is only finished when all backup replicas have been updated then it is a *synchronous replication*, otherwise the technique is called *asynchronous replication*. Well-known usage of the pattern is how the DNS Servers store their DNS zone in a redundant way.

4.2.1.2 Failover

In the failover pattern multiple instances of a component exist in the system, which can provide the same functionality. If one of them fails, another takes its place, this reconfiguration is called failover. The state of the failed component can be maintained by:

- Checkpointing,
- Shared storage.

When checkpointing is used, the state of the application is periodically saved to a stable storage. In case of a failover, the new active instance restores this checkpoint, and continues providing the service. This pattern is used in SA Forum's AMF and CKPT module.

In case of shared storage the state of the application is stored in a shared device, which is accessible by all components. After a failover the new instance simply uses the data of the failed instance. Thus, this approach does not tolerate the corruption of the data structures, only the failure of the components. Failover server cluster use this pattern.

4.2.1.3 Recovery

Recovery techniques utilize time redundancy to provide fault tolerant behaviour. Recovery methods can be characterized as:

- Forward recovery,
- Backward recovery.

In *forward recovery* for each identified erroneous state a new correct state is defined. If an error is detected, then the system is moved forward to the given predefined state. The technique can be fast for specialized errors, however, it works only for errors that are previously defined. Precise damage assessment is needed and the concrete implementation of forward recovery may depend on the application.

Backward recovery restores a previously, error free system state in case of an error. The recovery process can be *state-based*, a checkpoint containing the data structures of the application is restored, or *operation-based*, the operations of the application are audited and the inverse operations applied.

4.2.2 Safe measurement

Below we present some architectural design patterns aiming at providing support for the implementation of a dependable sensor–controller–actuator chain focusing on the data channels involved. The patterns outlined below were suggested by Bruce Powell Douglass [Douglass99, Douglass02].

This kind of design pattern should probably be used in the platooning application. In that case the sensor would be a real sensor (e.g. speed, distance, etc.), the role of the controller would be played by the services that interact, and finally the actuators are the steering mechanism or the fuel injection system.

4.2.2.1 Single channel architectures

In the simplest case, the actuation channel receives input from *sensors*, performs *input processing*, provides the processed input to the *data transformation* and *data integrity checker* modules, finally the *actuators* of the system are provided by the signals emitted by the *output processing* unit.

Figure 63 presents the single channel structure for an open loop case while Figure 64 presents the closed loop case.

When using a *single channel* only in a safety critical system, there should be mechanisms to identify and handle faults in the channel within the fault tolerance time. Unfortunately it may be difficult or even impossible to test for all faults within the available time and to remove common mode failures from the single channel resulting in low safety coverage. The obvious benefit of the approach is the relatively low cost.

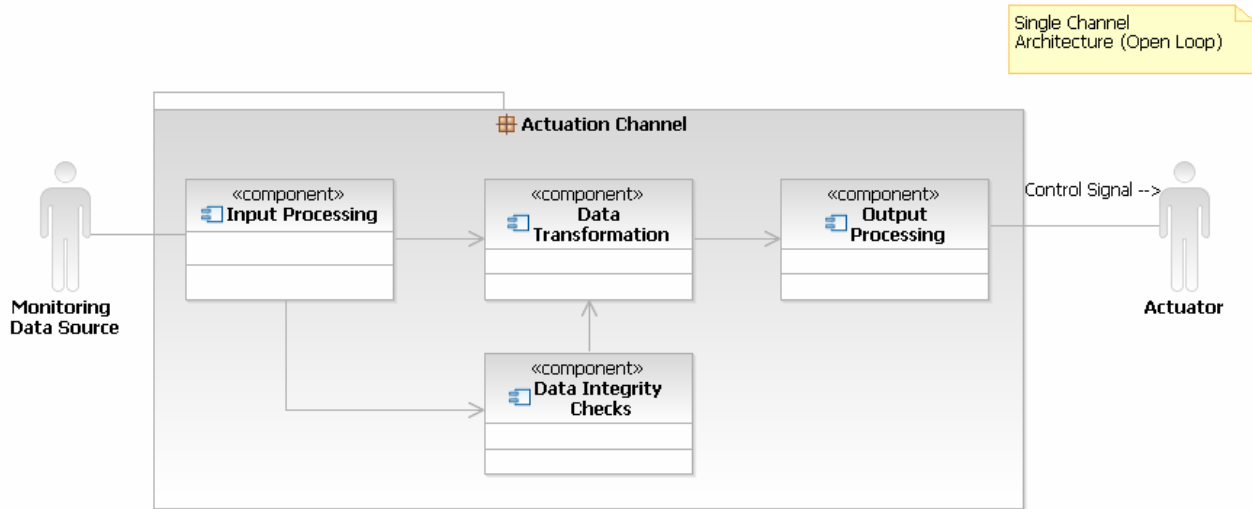


Figure 63: Single channel architecture (open loop)

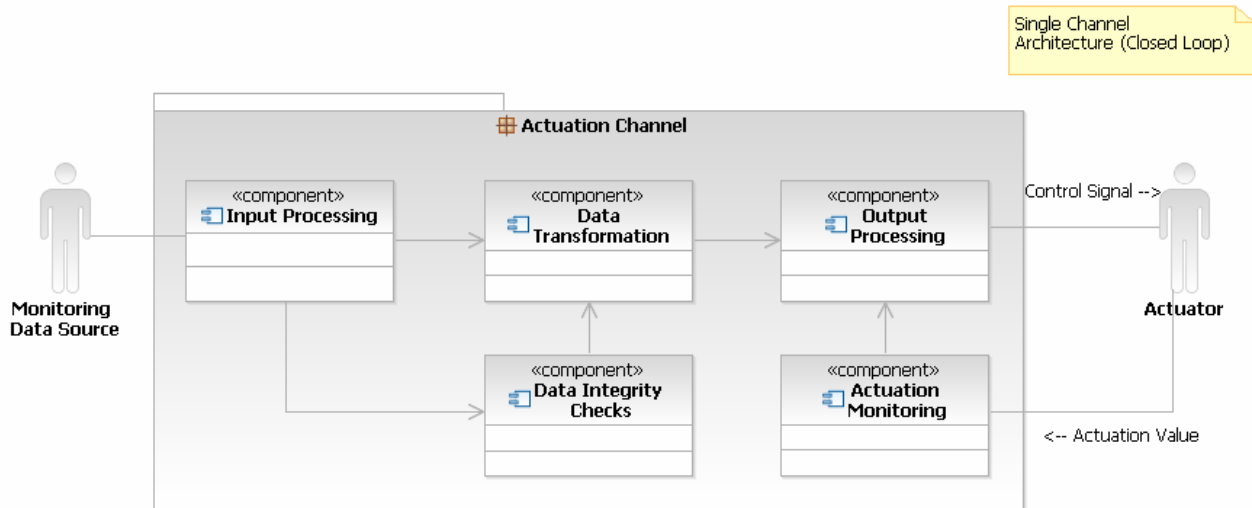


Figure 64: Single channel architecture (closed loop)

4.2.2.2 Multichannel architectures

In a *dual channel architecture* (Figure 65) two independent sensor–controller–actuator paths exist; this redundancy enables (i) the separation of safety-critical and less important services and (ii) the implementation of a single controller function with high dependability requirements.

- In the first approach where the designers aim at the *separation of safety critical control functions* from the rest of the system, critical functions can continue operation even in case

of the failure of a typically more complex and less reliable non-critical function. In this case it is also easier to meet hard real time requirements of safety-critical operations and to avoid some common failures.

- In cases where the dual channel allocation is used for the *dependable implementation of a single control function* the two channels may be (i) homogeneous or (ii) heterogeneous. In the homogeneous dual channel architecture two identical channels are used; this structure is applicable for identifying random faults but obviously can not address systematic faults. In the heterogeneous dual channel architecture two functionally equivalent but differently implemented channels are used, typically implemented by different developer teams (design diversification); generally this is the safest implementation but its development costs are high due to the duplication of the development efforts.

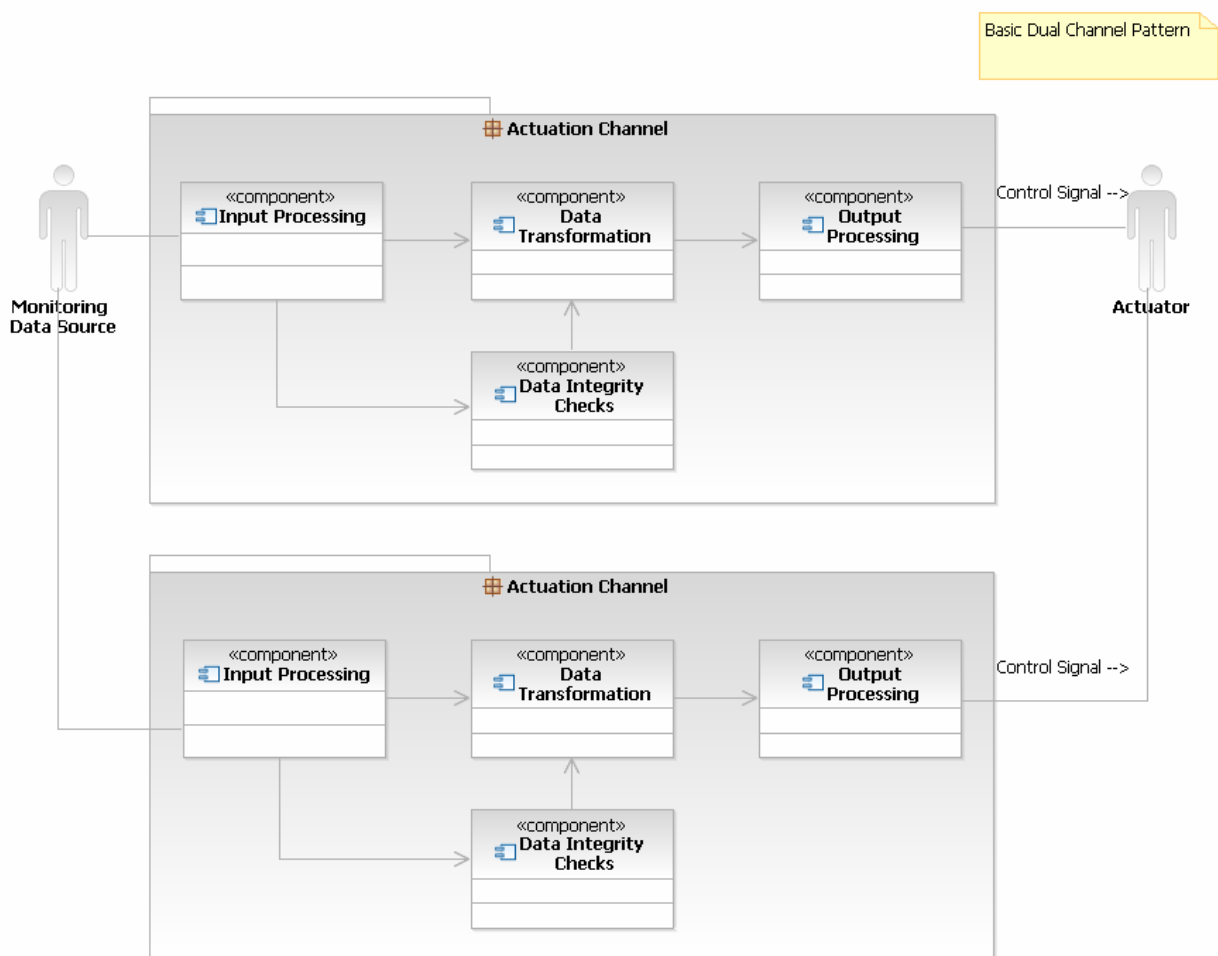


Figure 65: Basic dual channel architecture

The number of channels can be increased further in order to construct a multichannel voting structure where all channels calculate the same control function and finally a voter unit selects the value that was calculated by the majority of paths. (The number of channels is an odd number.) Figure 66 presents a TMR (triple modular redundancy) structure built up of three parallel control paths.

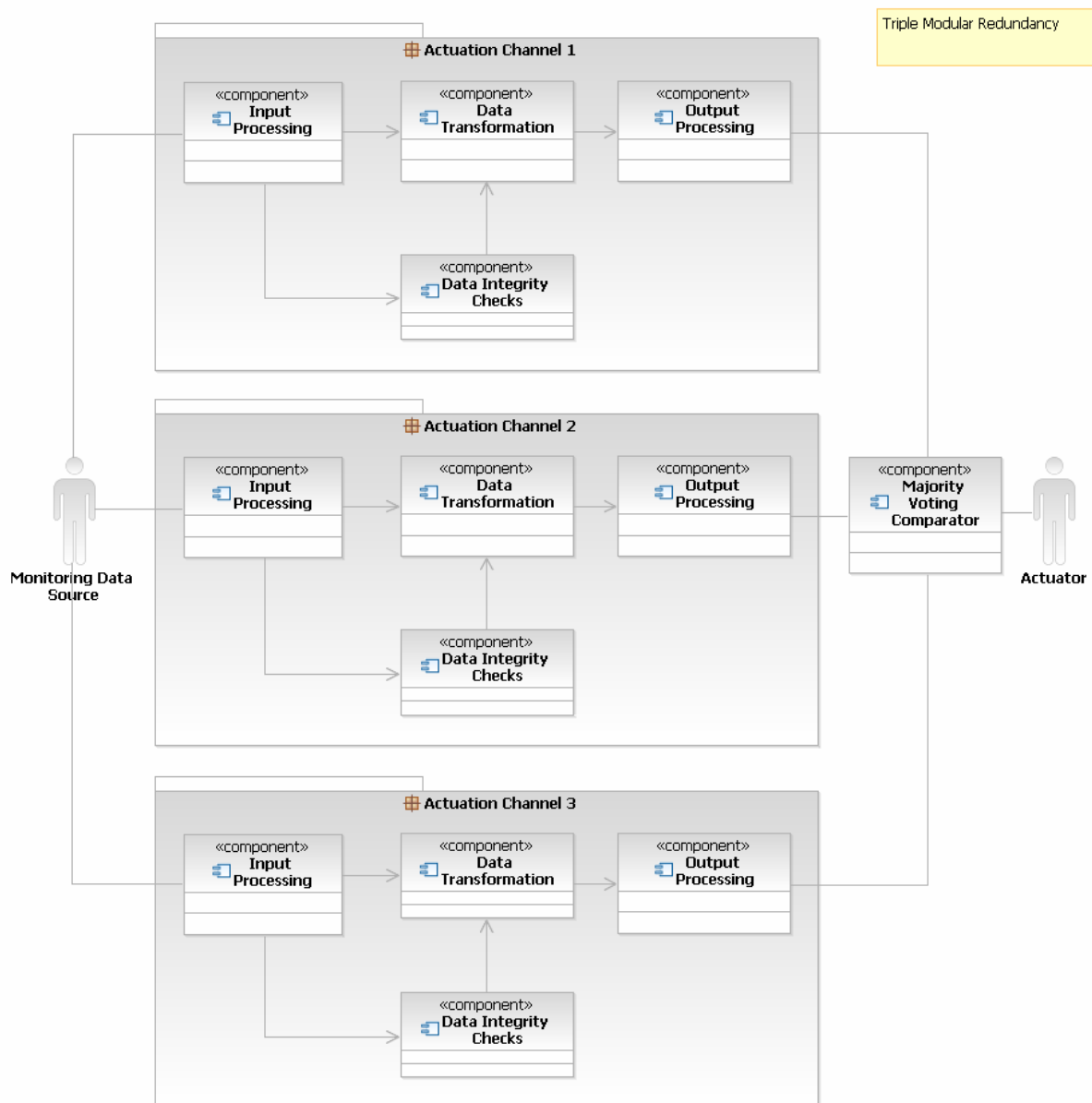


Figure 66: Triple modular redundancy

4.2.3 Messenger

Mobile and wireless communication is usually less reliable than the wired one, thus it is especially important to design a reliable communication mechanism in the applications. The following patterns aim to solve this problem.

4.2.3.1 Careful write

The careful write pattern can be used when there is enough time and bandwidth to verify the written data. The idea of the pattern is that after the application has written a data block, it shall read it from the destination and compare it with its version. The pattern can be easily implemented while it provides protection against communication errors and failures on the destination's side.

4.2.3.2 Mobile to fixed infrastructure

The mobile to infrastructure communication scenario could contain various fault types. The communication channels are less reliable; they are more vulnerable to disturbances in the environment. The mobility of the communication parties introduces further failure possibilities, e.g. nodes could move out of each other's range. However, a mobile host usually could utilize more than one communication adapter, for example a typical laptop has a WiFi, Bluetooth, and infra adapter.

Thus the pattern recommends that if an error is detected in a communication, instead of simply just retransmitting the message on the same channel, another one shall be used. In this way through the diversity of the communication channels the fault tolerance of the communication could be increased.

The pattern includes the following tasks:

- Switching to another adapter in case of a failure on the sender side,
- Detecting the duplicate messages on the destination side which arrive on different channels.

The second situation could occur if the failure detection in the sender returns a false positive, i.e. it reports a failure despite the fact that the destination received it, and the sender retransmits the message on a different channel.

4.3 Integration of tools

The meta-model and the UML profile provide the base for model driven development. The concepts of the tool-chain that will be developed on this basis are depicted in Figure 67.

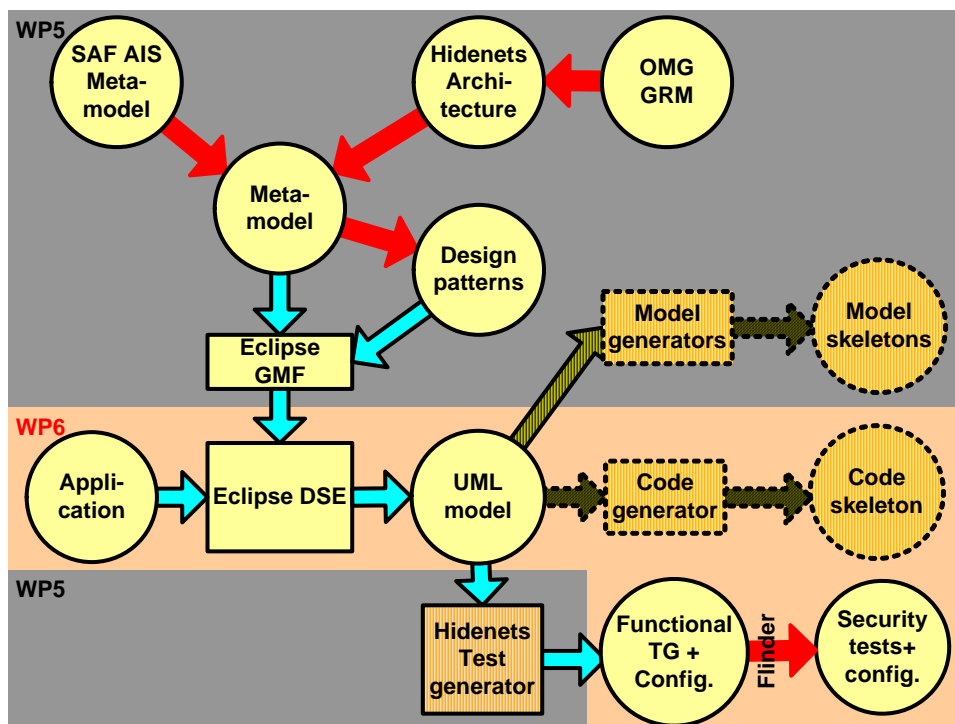


Figure 67: Concepts of the HIDENES tool-chain.

Meta-model and the design patterns. The HIDENETS metamodel is a synthesis of the HIDENETS architecture and the meta-model of the Service Availability Forum Application Interface Specification. Using the notions introduced by the meta-model, design patterns are defined

to facilitate the application of best practices. The meta-model and the design patterns serve as the input for the creation of the domain specific editor.

Domain specific editor in Eclipse GMF. HIDENETS application models are created by using the domain specific editor which is generated from the meta-model and enriched by design patterns. When an application model is ready it can be exported into a UML model.

Model generators. Functional requirements can be checked on the application's UML model, but UML tools usually do not provide a mechanism for the analysis of the non-functional properties (e.g. performance, dependability). Thus the UML model could be translated to other model formalisms, where tool support exists for these kinds of analysis tasks. Possible output formats could be input of model checkers or Petri nets.

Code generator. If the model is detailed enough a code skeleton can be generated that creates the general part of the code. Typical examples include generating the class skeletons of Java or C++ application. However, if the behaviour is also defined with state machines, more complex logic can be automatically generated.

Test generators. The UML model of the application can be a basis for generating test environment descriptions and test cases. The UML model should be annotated representing what the role of each component is in the testing scenarios. Given appropriate testing goals, e.g. coverage criteria and a description of the dynamic behaviour test cases could be created. Using the functional test suites and a description of the typical fault and attack types a security test suite could also be generated.

5 Conclusions

Even the initial evaluation of the metamodel clearly indicates the following expected benefits.

The metamodel helps identifying a high level of modularization and componentization possibilities. Due to these features, individual functionalities can be clearly separated and HIDENETS middleware services can be implemented in a highly heterogeneous and distributed form (even possibly by using remote procedure call or service styled interfaces between the interdependent modules). For instance, fundamental services can be implemented on resource (and power) constrained simple nodes controlled by local intelligent sub-centers complementing them to a full HIDENETS platform or the entire middleware can be implemented as a single moderate performance node.

The metamodel has to be extended with the parameters necessary to the *qualitative* and *quantitative verification* of designated applications in order to support formal and simulation based validation and verification. The general aspects for validation and verification are already described in the deliverable D4.1; the weaving of these aspects can be done either manually or by model transformations after the final consolidation of this deliverable.

The metamodel may serve as an overall framework for *metamodel-based testing* of applications. On the one hand, the definition of interfaces can be used as a specification of the testing stubs, on the other hand, the existence of the model of the application in a form compliant with the recent metamodel support the reuse and extensions of the concepts of model-based testing.

The metamodel serves as a basic document in the HIDENETS approach for software technology fitting into the mainstream of model-driven design and implementation. Subsequent steps will integrate the metamodel into a domain-specific development and verification environment. The designated technology environment for development is the Eclipse platform which supports the creation of development environments by providing standard services out of the box (e.g., version control, documentation support, development workflow, automation and tracing). In addition, recently adapted technologies, like the Eclipse General Modeling Framework, enable the highly automated creation of development environments like *domain specific editors* based on the metamodel (ensuring this way the compliance of the application models under development to the HIDENETS metamodel).

The metamodel supports the creation of the pilot demonstrators in two ways:

- On the one hand, the modularization and componentization features of the metamodel enable the module-wise development of the target middleware. In this case, the individual service groups can be developed separately and concurrently. In the testing phase the underlying services can be substituted by stubs or functionally similar skeletons. For instance, reliable communication can be substituted in other modules by standard services sharing an identical interface as the designated full functionality of HIDENETS, potentially using default or dummy parameters for some reliability interface parameters. Similarly, during the component development self-aware clocks can be substituted by the standard time services of the underlying operating system.
- On the other hand, automated implementation mechanisms like the statechart based code generator developed at BUTE allow for a highly productive implementation of the demonstrator in WP 6.

6 Appendix

6.1 Modelling concepts borrowed from the SPT profile

We will use the *time-related concepts* defined by the UML Profile for Performance, Schedulability and Time (SPT) for unambiguous notation and well-established reasoning in the context of HIDENETS concepts. Below we present the corresponding diagram from the SPT metamodel and a terse description of classes copied from the standard.

6.1.1 Concepts of the General Resource Modelling sub-profile

Below we introduce those concepts of the General Resource Model (GRM) profile that are used for *resource modelling* in our HIDENETS metamodel.

6.1.1.1 Abstract syntax

The basic package structure of the GRM profile is presented in Figure 68. The causality model is shown in Figure 69 and Figure 70, the core resource model is presented in Figure 71, the realization model is outlined in Figure 72.

Foundations of resource management are introduced in Figure 73. Metaclasses corresponding to exclusive resources and resource types are indicated in Figure 74 and Figure 75 respectively. The resource usage framework (Figure 76) supports both dynamic (Figure 77) and static (Figure 78) resource usage.

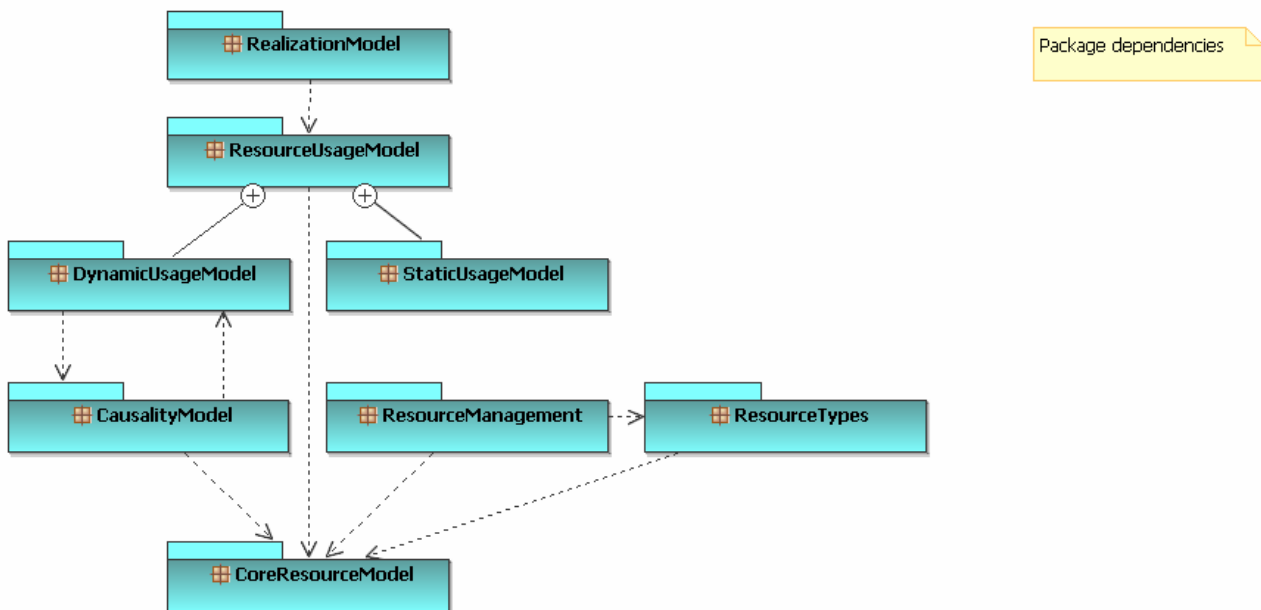
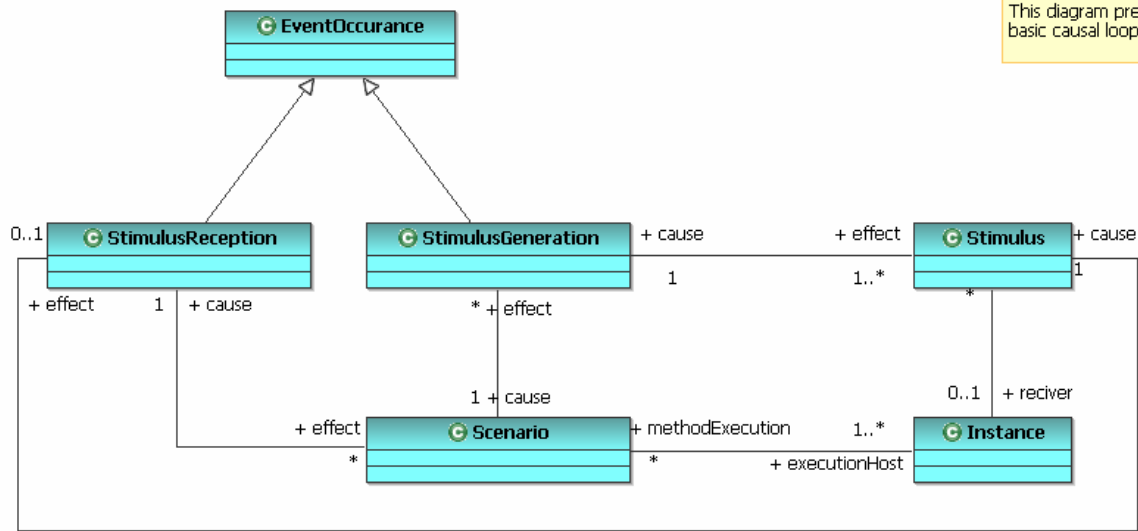
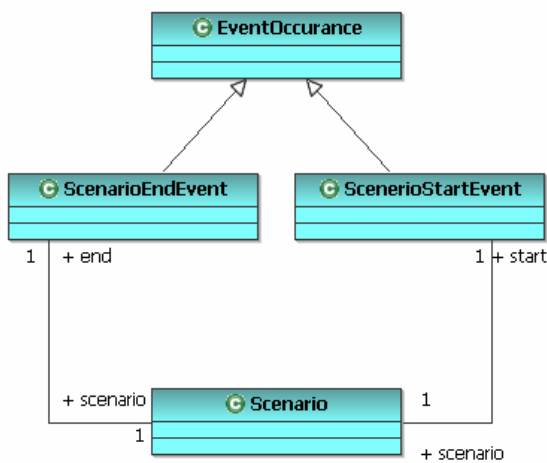


Figure 68: Package dependencies



This diagram presents the basic causal loop model.

Figure 69: Basic causal loop model



This diagram presents scenario starts and event occurrences.

Figure 70: Scenario start and end event occurrences

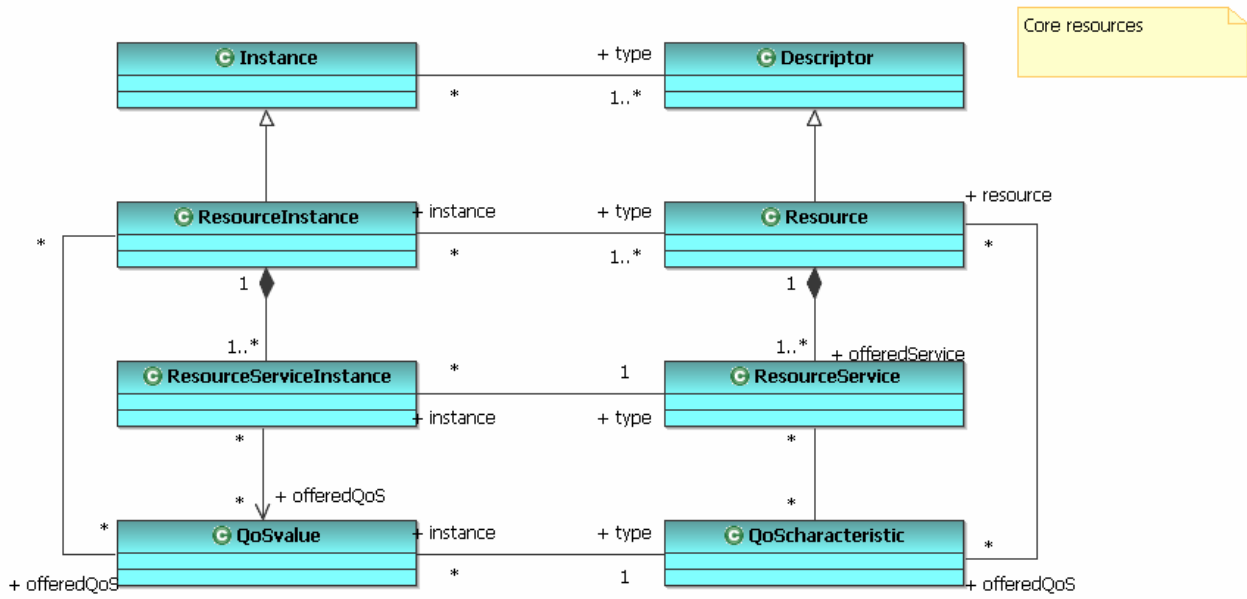


Figure 71: Core resources

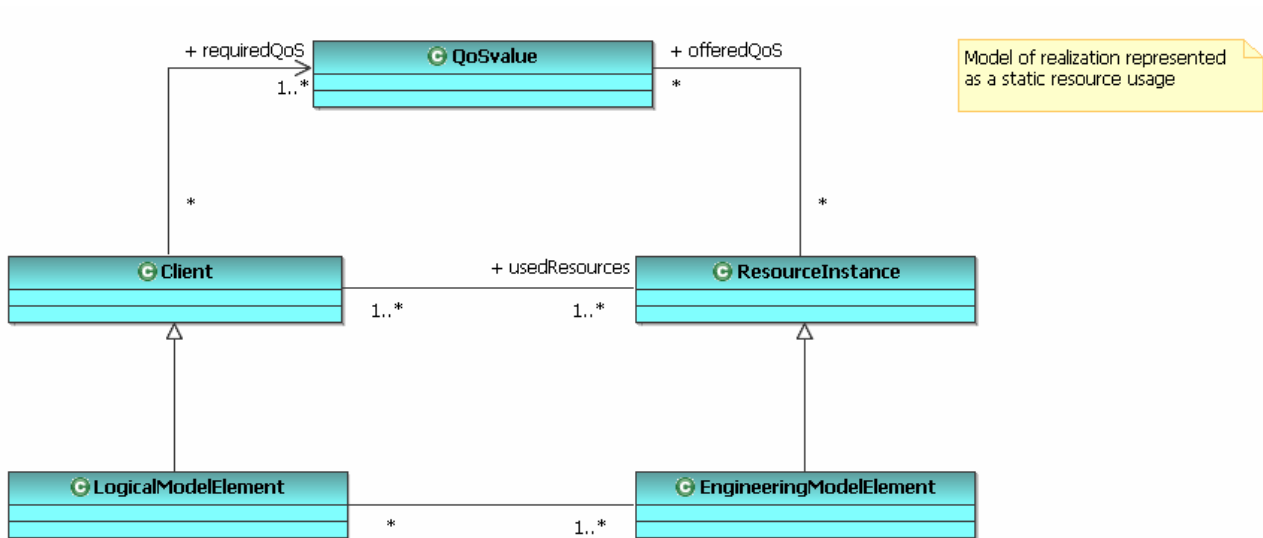


Figure 72: Model of realization represented as a static resource usage

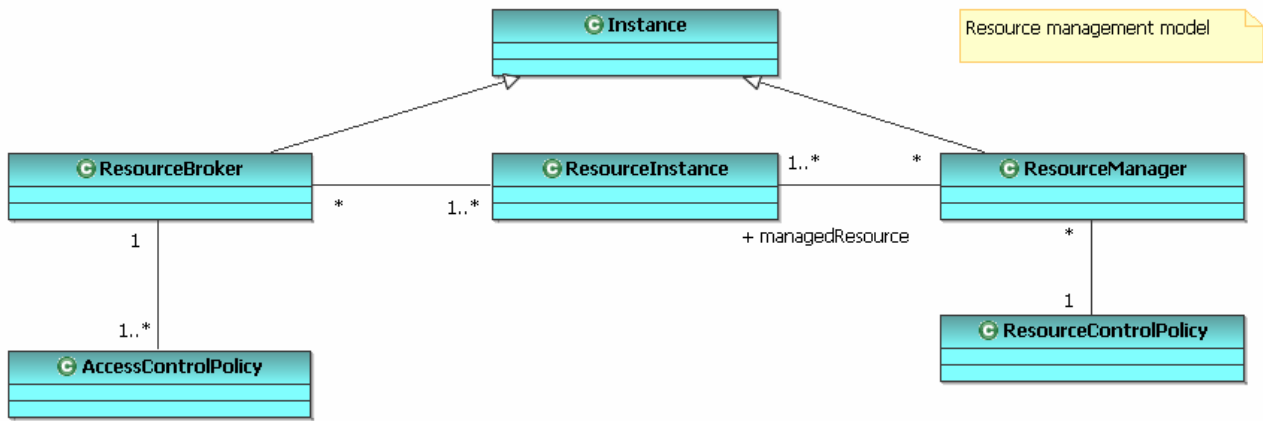


Figure 73: Resource management model

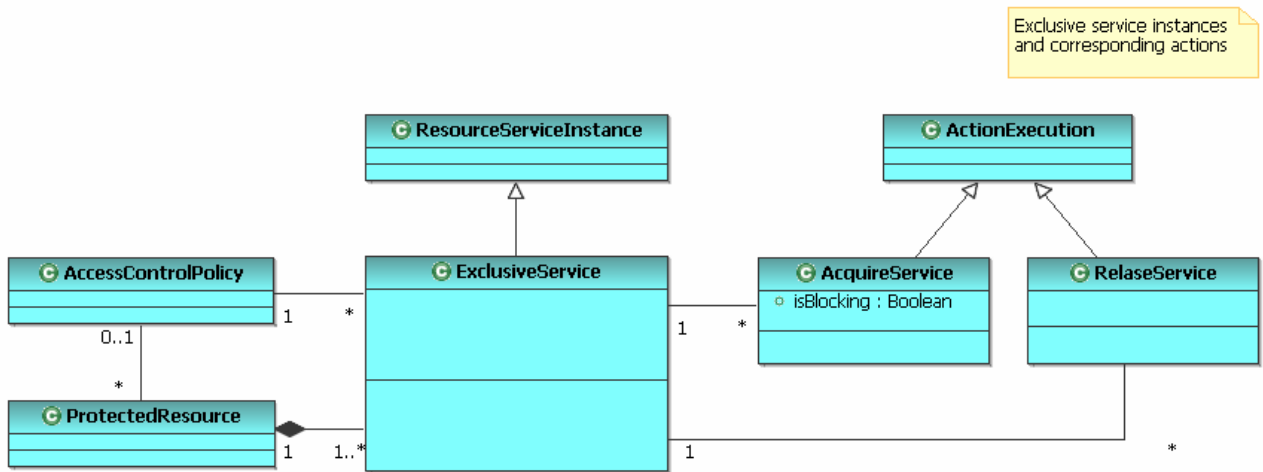


Figure 74: Exclusive service instances and corresponding actions

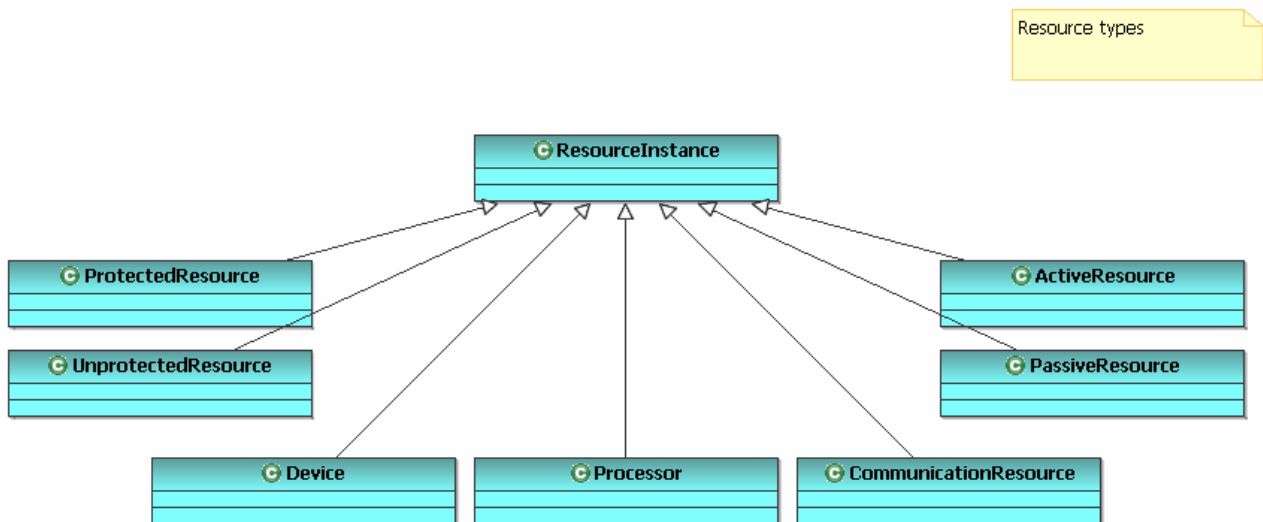
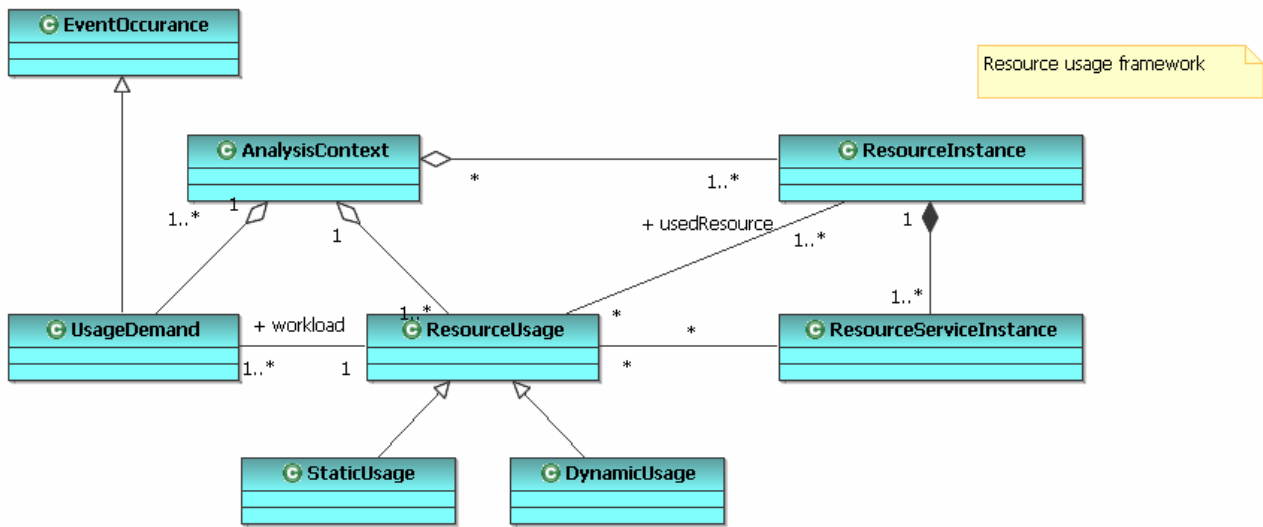
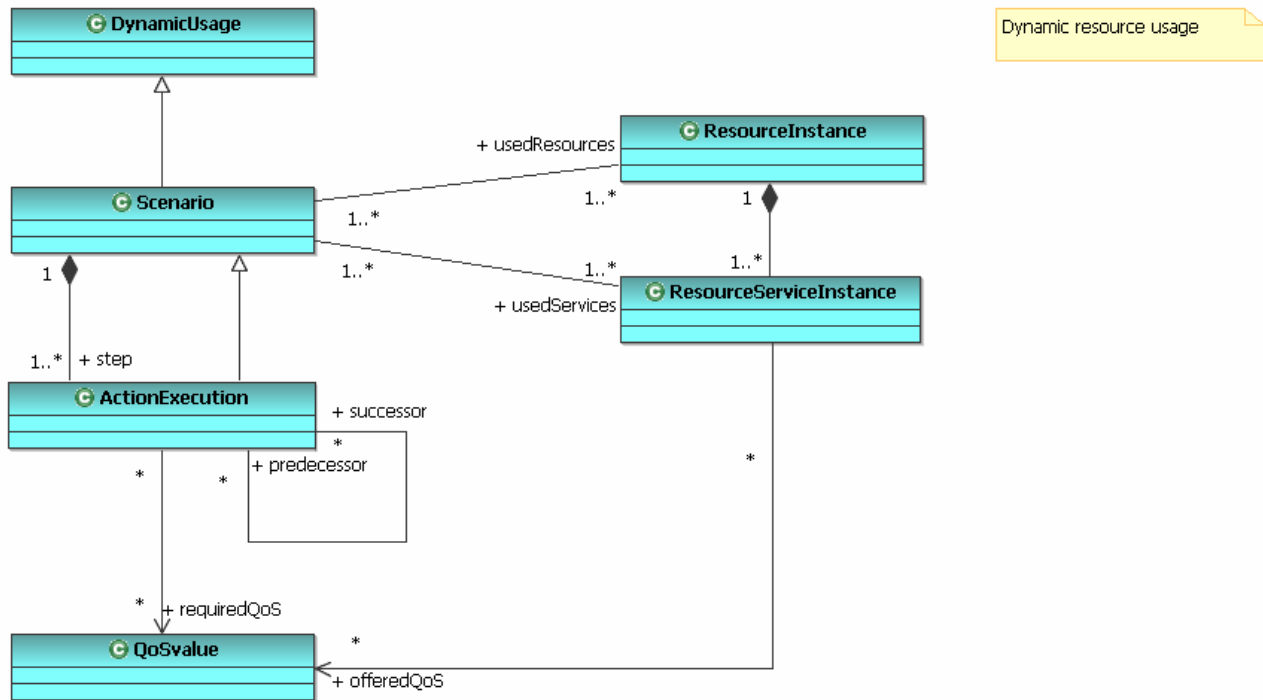


Figure 75: Resource types



Resource usage framework

Figure 76: Resource usage framework



Dynamic resource usage

Figure 77: Dynamic resource usage

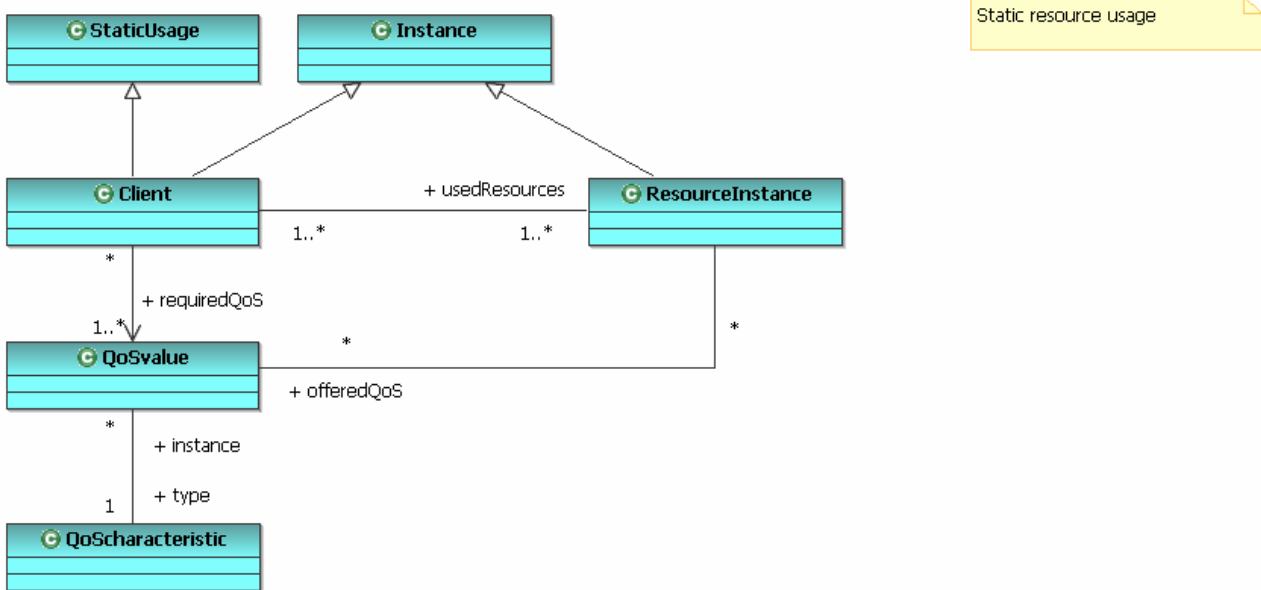


Figure 78: Static resource usage

6.1.1.2 Class descriptions

6.1.1.2.1 Metaclass Access Control Policy

This is an instance of the policy by which access to an instance of an exclusive service is controlled. The responsibility for administering the policy rests with a resource broker.

6.1.1.2.2 Metaclass Acquire Service

This is an instance of an action execution that is used to gain access to an instance of an exclusive service on some protected resource. An exclusive service of a protected resource cannot be accessed unless this action is executed successfully. The operation may fail, in which case attempts to use the protected resource will also fail. Once the resource is no longer needed, it can be released with a “releaseService” action. Until the latter action is executed, the resource remains in “possession” of the entity that executed the acquire action.

6.1.1.2.3 Metaclass Action Execution

This represents a single execution of some action specification. Each execution of the same action will result in a different action execution (i.e., each action execution has a distinct identity). Action execution is a subclass of Scenario and inherits its associations. This allows action executions to be decomposed into finer-grained steps, if so required.

6.1.1.2.4 Metaclass Active Resource

This is a resource that is capable of generating its own stimuli concurrently (i.e., asynchronously to other activities) without necessarily being prompted by an explicit scenario. This is a subclass of Resource Instance.

6.1.1.2.5 Metaclass Analysis Context

This is a context that is the root for an analysis of a model (there can be more than one context for a given model).

6.1.1.2.6 *Metaclass Client*

This is an explicit run-time instance that uses resources. It is used in static model analysis schemes. In dynamic analyses, it is implied by a scenario.

6.1.1.2.7 *Metaclass Communication Resource*

This is a resource whose primary purpose is to connect two or more other types of devices in order to enable them to communicate. This is used to model various kinds of networks, channels, etc. This is a subclass of Resource Instance.

6.1.1.2.8 *Metaclass Descriptor (abstract)*

An abstract concept representing some kind of design-time specification. This concept includes all kinds of descriptors such as classifiers, collaborations, data types, etc. It is generally assumed that every instance element in the domain model may have an implicit or explicit descriptor.

6.1.1.2.9 *Metaclass Device*

This is a resource instance that is neither a processor kind of device nor a communication device. In real-time systems, it is used for modelling various kinds of specialized devices such as sensors, effectors, secondary storage devices, etc. This is a subclass of Resource Instance.

6.1.1.2.10 *Metaclass Dynamic Usage*

A kind of resource usage instance that involves some kind of dynamic scenario with an ordered usage of resources involved in the scenario. This is typically used for more precise types of analyses.

6.1.1.2.11 *Metaclass Engineering Model Element*

This is an instance of a model element that realizes one or more logical model elements in the layered interpretation of resource usage. In this role, the model element acts as a resource instance with specified offered QoS values.

6.1.1.2.12 *Metaclass Event Occurrence [abstract]*

An instance of the occurrence of an event (change of state) of some type. Event occurrences are assumed to be instantaneous.

6.1.1.2.13 *Metaclass Exclusive Service*

A resource service instance associated with a protected resource that can only be accessed if the appropriate resource broker approves. The resource broker makes these decisions on the basis of an access control policy set for each exclusive service. This service can only be used after an acquire service action has been executed successfully and up to the time that the release service action has been executed.

6.1.1.2.14 *Metaclass Instance*

An abstract concept representing some kind of run-time instance that is created based on one or more type specifications (descriptors). This concept includes all kinds of instances, including objects, data values, etc.

6.1.1.2.15 Metaclass Logical Model Element

An instance of a model element that is realized by one or more engineering model elements in the layered interpretation of resource usage. In this role, the model element acts as a client with specified required QoS values. In general, a logical model element may be realized by a set of engineering model elements.

6.1.1.2.16 Metaclass QoS Characteristic (abstract)

This concept represents the descriptor of some kind of resource service characteristic that specifies either how well the service needs to be or can be performed. It is an abstract concept that is used to describe the overall GRM framework. In case of specific domain concepts, however, QoS characteristics are usually modelled by attributes, since this is both simpler and easier to understand.

6.1.1.2.17 Metaclass QoS Value (abstract)

The value of a QoS characteristic for a specific resource or resource service instance. The value can range from a single number to a complex structured value such as a probability distribution.

6.1.1.2.18 Metaclass Passive Resource

This is a simple resource that is incapable of generating stimuli unless it is prompted by a scenario. This is a subclass of Resource Instance.

6.1.1.2.19 Metaclass Processor

This is a resource instance that is capable of storing and executing a computer program and its associated data. It is used for modelling physical CPUs, computers, etc. as well as virtual machines (“logical” processors). This is a subclass of Resource Instance.

6.1.1.2.20 Metaclass Protected Resource

This is an instance of a resource that provides one or more exclusive services — services that can only be accessed according to an access control policy administered by an associated resource broker. (The resource broker is a role that may be played by the resource itself.).

6.1.1.2.21 Metaclass Release Service

This is an instance of an action that is used to release an instance of an exclusive service that was acquired by a previously executed acquire service action. This action execution is not bound by access control restrictions and can be executed at any time, even if the resource service has not been acquired before (in which case it has no effect).

6.1.1.2.22 Metaclass Resource

A design-time descriptor that specifies a kind of resource. A resource is an element that has resource services whose effectiveness is represented by one or more QoS characteristics.

6.1.1.2.23 Metaclass Resource Broker

An instance of a run-time entity responsible for controlling access to the exclusive services of a resource. Note that this role may be played by the resource itself. The broker processes acquisition requests from clients of the service and, based on the appropriate access control policy for that service, it dispenses access to the service. If a service instance is busy, then the reply may remain pending until access is possible.

6.1.1.2.24 Metaclass Resource Control Policy

An instance of a policy that is used to manage one or more resource instances. This policy is used by a resource manager. Note that the role of a resource manager may be played by the resource itself.

6.1.1.2.25 Metaclass Resource Instance

An instance of some resource. A resource is a run-time entity whose services can be characterized by QoS values. This is a generic superclass that can be specialized in a number of ways.

6.1.1.2.26 Metaclass Resource Manager

The run-time instance responsible for managing one or more resource instances. This is primarily a role that can be played even by resource instances. The intent behind this concept is to allow modelling of resource managers that typically appear in run-time system infrastructures (memory managers, device controllers, etc.). In general, management includes the creation, maintenance, and possible destruction of resources according to some resource control policy. A resource manager may also play the role of a resource broker.

6.1.1.2.27 Metaclass Resource Service

A design-time descriptor of a service offered by some resource. This service may have QoS characteristics that are used to describe how well instances of the service perform their functional responsibilities.

6.1.1.2.28 Metaclass Resource Service Instance

This is a specific instance of a resource service attached to a specific resource instance. While it is somewhat unusual to talk about “instances of a service,” it makes sense in cases where the service may have QoS characteristic values. Thus, even though two service instances may have the same descriptor, the individual instances may have different QoS values.

6.1.1.2.29 Metaclass Resource Usage (abstract)

An abstraction of an instance of a pattern of usage of one or more resources. A resource usage instance may be static or dynamic. The pattern itself is kept separate from a specification of the external demand which induces the usage. Although this represents a specific instance, in case of periodically occurring usages, a single instance is used to represent all repeated incarnations.

6.1.1.2.30 Metaclass Scenario

An instance of a dynamic usage of a set of resources and resource services that consists of an ordered collection of individual steps (action executions). This concept is used for modelling complex dynamic situations for more refined types of analyses. The concept is defined recursively to allow the modelling of complex hierarchies of scenarios. Thus, the components of a scenario may themselves be scenarios and so on. The model of scenario instances supports both concurrent and sequential scenarios.

6.1.1.2.31 Metaclass Scenario End Event

This event occurrence takes place at the instant when a scenario fully completes execution.

6.1.1.2.32 Metaclass Scenario Start Event

This event occurrence takes place at the instant when a scenario commences execution.

6.1.1.2.33 Metaclass Static Usage

A kind of resource usage instance that only identifies a set of clients and resources. This is typically used for very high-level types of analyses.

6.1.1.2.34 Metaclass Stimulus

A concept imported directly from the UML dynamic semantics model. It represents an instance of a communication that is in transit between a sender instance and a receiver instance (or a set of receiver instances). A stimulus is generated by the occurrence of a stimulus generation event, which is, in turn, induced by the execution of an action that generates an interaction between run-time instances. This concept is primarily used to model various dynamic situations.

6.1.1.2.35 Metaclass Stimulus Generation

An occurrence of an event that results in the generation of a stimulus (see above). This occurs as a direct consequence of a scenario that includes an action execution step (such as a call to an operation or a signal send).

6.1.1.2.36 Metaclass Stimulus Reception

An occurrence of an event that represents the acceptance of a stimulus by a receiver instance. This is normally the result of some instance executing an implicit or explicit receive action. The details of this are outside of the scope of the GRM and may be modelled in detail in specific cases where that is important for model analysis.

6.1.1.2.37 Metaclass Unprotected Resource

An instance of a resource which does not offer exclusive services and whose access, therefore, is not protected by an access control policy.

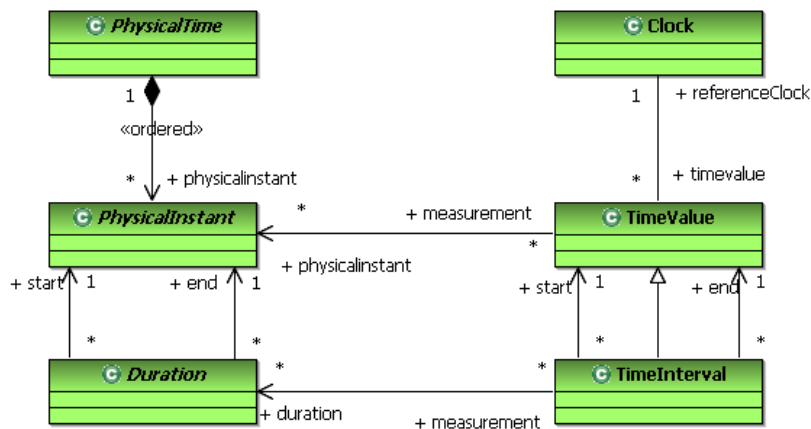
6.1.1.2.38 Metaclass Usage Demand

A kind of event occurrence that leads to a usage demand. Set of values (possibly complex) that characterize the external load intensity imposed by a resource usage in a given analysis context.

6.1.2 Concepts of the Time sub-profile

6.1.2.1 Abstract syntax

Figure 79 presents the *key time-related concepts* introduced by the SPT profile.



This diagram presents the fundamental concepts introduced by the SPT profile for time modeling.

Figure 79: Time-related concepts of the SPT profile

6.1.2.2 Class descriptions

Below we present an abbreviated version of class descriptions copied from the standard; for an in-depth discussion refer to the original source [SPT].

6.1.2.2.1 Metaclass *Physical Time*

This is the *notion of physical time*. A crisp definition of physical time has forever eluded scientists and philosophers alike, so we are not going to be presumptuous and attempt to do so here. Our fundamental model of physical time is that it is a continuous and unbounded progression of instants. Formally, time is viewed merely as a means of imposing a partial order on events by associating events with time instants. We make no assumptions about whether time is global, which permits the modelling of relative time. It is an abstract concept since we do not anticipate that real-time system modellers will need to represent physical time itself in their models. Instead, it is assumed that they will represent it indirectly through time values, time-measuring mechanisms, etc.

6.1.2.2.2 Metaclass *Physical Instant*

This is the concept of a *physical time instant* i.e., a point in time. Like geometrical points, instants do not have any extent (duration). Given any two different physical time instants, one will always precede the other. As with physical time, we do not anticipate that system modellers will need to represent physical time instants directly, but only through associated concepts such as time measurements or events.

6.1.2.2.3 Metaclass *Duration*

This represents the *interval between two physical time instants*. As with physical time, we do not anticipate that system modellers will need to represent physical durations directly, but only through associated concepts that can measure it. Hence, we do not provide direct support for modelling physical durations, but only for modelling durations as measured by timing mechanisms.

6.1.2.2.4 Metaclass *Clock*

This is a kind of *timing mechanism* that generates a clock interrupt periodically. This concept inherits most of its features from TimingMechanism.

6.1.2.2.5 Metaclass Time Value

A value that corresponds to a particular physical instant in time as measured by some reference clock in some inertial frame of reference.

6.1.2.2.6 Metaclass Time Interval

A kind of time value corresponding to a duration. There are two kinds of durations: absolute and relative. The former start and end at specific points in time whereas the latter are not rooted to any particular time.

6.2 Modelling concepts borrowed from the QoS & FT profile

We will use the QoS-related concepts defined by the UML Profile for Modelling Quality of Service and Fault Tolerance Characteristics and Mechanisms (QoS & FT) for unambiguous notation and well-established reasoning in the context of similar HIDENETS concepts. Below we present the corresponding diagrams from the QoS & FT metamodel and a terse description of classes copied from the standard.

6.2.1 Abstract syntax

Fundamental QoS-related metaclasses (QoS context, characteristics, dimensions, etc.) are introduced in Figure 80; the metaclass QoS Constraint is further refined in Figure 81.

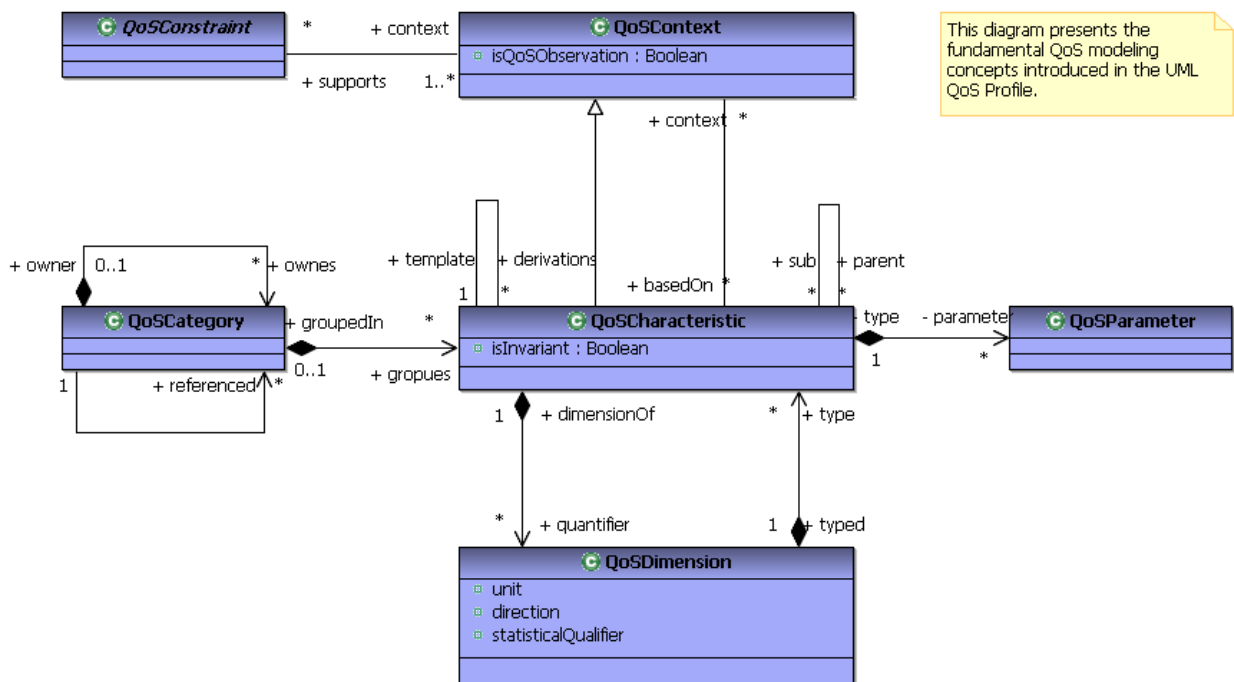


Figure 80: QoS context, characteristics and dimensions in the QoS & FT profile

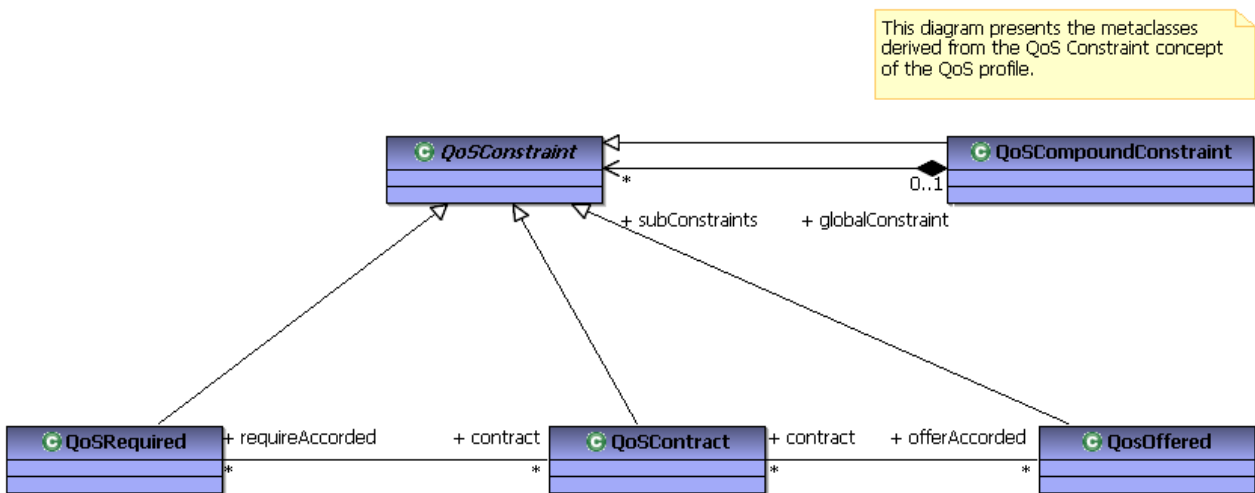


Figure 81: QoS constraints in the QoS & FT profile

6.2.2 Class descriptions

Below we present an abbreviated version of class descriptions copied from the standard; for in-depth discussion refer to the original source [QoSFT].

6.2.2.1 Metaclass QoS Characteristics

QoS Characteristics represent *quantifiable characteristics of services*. The QoS Characteristics are specified independently of the elements that they qualify. QoS Characteristic is the constructor for the description of non-functional aspects like: latency, throughput, capacity, scalability, availability, reliability, safety, confidentiality, integrity, error probability, accuracy and loading. These are some general characteristics, but specific domains have associated specific characteristics. The relation Sub-Parent provides support for the extensions-specialization. This association supports the reuse of characteristics. Sometimes the QoS Characteristic definition requires some parameters. The description of generic characteristics may require, for example, the parameterization of the units and types for the description of value definitions, or some specific methods for the quantification of the values; QoS Parameter supports these parameters. The attribute *isInvariant* specifies when the QoS Characteristic can or cannot update the value of dimensions dynamically.

6.2.2.2 Metaclass QoS Dimension

QoS Dimensions are *dimensions for the quantification* of QoS Characteristics. We can quantify a QoS Characteristic in different ways (e.g., absolute values, maximum and minimum values, statistical values). For example, we can quantify the latency of a system function as the end-to-end delay of that function, the mean time of all executions, or the variance of time delay. A QoS Characteristic can require more than one type of value for its quantification. Examples of dimensions of reliability are: time to repair, time to failure, failure masking that server exposes to their clients (failure, omission, response, value, timing, late, or early), service failure i.e., the way in which a service can fail (halt, roll back, or initial state), semantic of services (exactly once, at least once, or at most once) and number of failures supported.

6.2.2.3 Metaclass QoS Category

When the number of QoS Characteristics is large, or they are especially complex, some mechanisms for *grouping* are required. Some examples of general groupings of quality attributes are: (i) Performance: Performance makes reference to the timeliness aspects of how software systems behave. (ii) Dependability: Dependability is the property of computer systems such that reliance can justifiably be placed on the service it delivers. (iii) Security: this capability covers different subjects such as the protection of entities and access to resources. The main difference between a QoS Category and a QoS Characteristic is that QoS Characteristics are directly quantifiable, and QoS Categories do not provide a direct framework for the evaluation of non-functional attributes; it requires a more detailed level of specification to establish constraints or comparisons. The definition of QoS Characteristics included in a QoS Category can make reference to characteristics defined in other categories.

6.2.2.4 Metaclass QoS Parameter

The description of generic characteristics may require, for example, the *parameterization of the units and types* for the description of value definitions, or some specific methods for the quantification of the values; QoS Parameter supports these parameters.

6.2.2.5 Metaclass QoS Constraint

This is an abstract metaclass. A QoS Constraint limits the *allowed values* of one or more QoS Characteristics. The QoS Constraints define the constraints of the QoS Characteristics of modelling elements. Application requirements or architectural decisions limit the allowed values of quality and the QoS Constraints describe these limitations. QoS Context defines the QoS Characteristics and functional elements involved in a QoS Constraint. The QoS Context establishes the vocabulary of the constraint, and the QoS Constraint the allowed values.

6.2.2.6 Metaclass QoS Required

When a client defines its QoS Required constraint, the provider (software element or a resource) that supports the service *must provide* services with some quality levels to achieve its clients' requirements. The service provider must support not only the service required but also must provide its services with certain quality constraints. When the provider defines its QoS Required constraint, the client must achieve some quality requirements to get the quality offered. An example of QoS Required constraint for providers is the maximum frequency of invocation from its client. When a client uses a service with some quality requirements, it specifies the non-functional requirements with a QoS Required constraint. This constraint specifies the quality that the server must achieve. This constraint limits the space of valid values for the QoS Characteristics involved in the service. The QoS Characteristics are the dimensions of the quality space, and the QoS Required defines the valid values of this space. The expressions of QoS Required constraints appear because the system must fulfil some user requirements, or because a user-provider (a software element that provides services and uses other services) component or subsystem must support other qualities required. The user-provider requires some qualities for its providers to achieve the required quality. For example, a reliable component can support its quality requirements, when its service providers are reliable. Often, an end-to-end quality requirement is decomposed into a set of sub-quality requirements, and the software architects define a set of QoS Required constraints to achieve the quality required.

6.2.2.7 Metaclass QoS Contract

The quality provider specifies the quality values it can *support* (provider-offered QoS) and the requirements that *must achieve* its clients (provider-required QoS). The user, the quality it requires (client-required QoS) and the quality that it ensures (client-offered QoS). Finally, in an assembly process, we must establish an agreement between all constraints. In general, the allowed values specified by a client-required QoS must be a subset of values supported in provider-offered QoS, and the allowed values that provider- required QoS specifies must be a subset of values supported in client-offered QoS. If the provider does not support the required QoS, we must negotiate the contract and the final quality provided. Sometimes, we cannot compute the Contract QoS statically, because it depends on the resources available or quality attributes fixed dynamically. However, we can identify the qualities supported in the contract and some limit values. To compare two QoS Values, we can use the = operator for QoS Dimension Slot, or we can use the attribute direction in QoS Dimension. In the last case the data type of QoS Dimension associated with QoS Dimension Slot must support the relational operators < and >.

6.2.2.8 Metaclass QoS Offered

The specification of software components includes the description of their interfaces. The interfaces include the set of services provided. However, their architectures and implementations are designed to support some specific qualities. Architects design a component or a subsystem to support some levels of scalability, to have a limited response time, or to make the component reliable based on persistence techniques, and these decisions create impacts on the types of data structures and algorithms. These quality properties have a special impact on the architecture. QoS Offered has associated the set of QoS Characteristics that the component takes into account (QoS Characteristics are part of the specification of QoS Context). QoS Offered establishes the *limits of values* that support the software elements. This is the space of quality that can support the software element. When a quality does not appear in the context of the Offered QoS, the software element does not take it into account, and the component does not guarantee this quality. Often, the Offered QoS depends on the QoS provided by the resources and service providers that the software element uses. When the provider defines an Offered QoS constraint, it is the provider who must achieve the constraint. When a client defines an Offered QoS constraint, the client must achieve the constraint invoking the service.

6.2.2.9 Metaclass QoS Compound Constraint

QoS Compound Constraints can represent global constraints decomposed in a set of subconstraints. The sub constraint can have a precedence order relation. These relations can represent, for example, how to decompose a latency constraint in a set of subconstraints.