Università degli Studi di Napoli Federico II

Dottorato di Ricerca in Ingegneria Informatica ed Automatica
XXV Cycle

A Model-Driven Approach to Quantitative Analysis of Critical Systems

Author:
Roberto Nardone

Supervisor:
Prof. Antonino Mazzeo
Prof. Nicola Mazzocca
Ing. Immacolata Lamberti
Coordinator:
Prof. Franco Garofalo

SecLab Group
Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione

April 2013

“Essentially, all models are wrong, but some are useful.”
“Remember that all models are wrong; the practical question is how wrong do they have to be to not be useful.”

George E. P. Box

Universita' degli Studi di Napoli Federico II

Abstract

Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione

Doctor of Philosophy

A Model-Driven Approach to Quantitative Analysis of Critical Systems

by Roberto Nardone

Critical systems are present in our daily life and affect many aspects providing essential support to several human activities; they are employed in several application domains, providing support to economic activities, transportation, communication, health-care. Such systems must meet strict non-functional requirements, including dependability requirements, and should be able to cope with competitive market needs. Their criticality implies the necessity to meet several requirements, often dictated by international standards, whose fulfillment must be demonstrated in order to achieve necessary certifications. Quantitative evaluations are hence needed to assess and demonstrate compliance with target requirements, since design phases of the lifecycle.

For this purpose modeling approaches have been investigated during the years; their aim is to have a model of the overall system which allows to assess and demonstrate these properties. It is commonly known that the adoption of formal models in industry, although if strongly recommended when not mandatory, is slowed down by difficulties dictated both by the complexity of the models (which reflects the complexity of the systems) and by the need to have skilled staff in the usage of formal languages. For this reason, also if the academic research trends have focused on complex modeling approaches and techniques, formal models are not so used in industries: in particular only combinatorial models are widely adopted, thanks to their intuitive representation. The great simplifications introduced by combinatorial models lead to approximated results, hence system designers do not trust in them. In the last years the need of a "model engineering" is noticed: there is the necessity to define appropriate methodologies and processes to support development activities of complex models, aiming at improving the model quality/cost ratio. In this way three main research trends have been introduced: formal models generation, multiformalism, and compositional approaches.

This thesis aims at defining a methodology for quantitative analysis of critical system that can integrate Model-Based evaluations, conducted by using formal models, with

Model-Driven Engineering (MDE) paradigms, taking advantage of the actual research trends. More in detail, according to such approach, critical systems properties can be assessed starting from high level models expressed in proper languages (usually called Domain Specific Modeling Languages - DSMLs), and defining automatic transformation chains, able to generate formal models. The methodology is implementable into cost-effective processes in order to reduce time-to-market specifically during verification stages of critical system development lifecycles. Such objective will be achieved by both improving usability and re-usability of typical Model-Driven artifacts. The adoption of this methodology for railway system dependability analysis may lead to an estimated costs reduction of approximately 30%. The usage of a high-level central language significantly decreases also the training time of new analysts, as they only need to learn the DSML usage (properly developed for the application domain).

This thesis defines also two high-level modeling languages: the first, CIP_VAM, has been defined to model vulnerability aspects of physical infrastructures, while the second, DAM-Rail, allows the dependability modeling in railway domain.

Another important contribution of this thesis is given in the definition of new methods and formal operators: in detail a reference model architecture for performability and Quality of Service evaluation is defined, as well as the formal concept of Model Template is introduced (to enable model reuse). At last two techniques which deals with scalability are addressed: stub and reduced models.

Even if the focus is on quantitative evaluation, the proposed methodology is useful also for the complete design (based on centralized informations), hence it is possible to extend it for automatic generation of documents and, more generally, of all those artifacts which support system development.

The proposed approach has been applied with different goals to case studies coming from three different domains: railway systems, conferencing systems, and sensor network. These systems have different and complementary properties in terms of number of involved components, scalability, etc. Produced models are described, and obtained results are shown and commented.

The thesis is structured in three Parts, each one composed of three Chapters. The first Part introduces the problems, explores related works and points out open issues. In detail, Chapter 1 introduces critical systems and outlines the state of the art, in Chapter 2 Model-Driven approaches are described in order to introduce their key concepts, while Chapter 3 describes two languages found in literature (i.e. MARTE and DAM). The second Part describes the proposed approach, defines the two high-level modeling languages and introduces the above mentioned methods, formal operators and techniques.

In detail Chapter 4 describes the proposed approach and shows a simple example of its application, Chapter 5 defines the above mentioned high-level modeling languages (i.e. CIP_VAM and DAM-Rail), Chapter 6 addresses defined methods and formal operators. The last Part shows the application of the proposed approach to real world systems. In detail Chapter 7 applies the methodology to railway systems, Chapter 8 applies it to conferencing systems while Chapter 9 adopt the methodology for performance modeling of wireless sensor networks. Chapter 10 end the thesis, giving some conclusive remarks.

This thesis includes materials from the following research papers, already published in peer-reviewed conferences and journals:

- A. Mazzeo, N. Mazzocca, **R. Nardone**, L. DAcierno, B. Montella, V. Punzo, E. Quaglietta, I. Lamberti, P. Marmo, An integrated approach for availability and QoS evaluation in railway systems, Proceedings of the 30th International Conference on Computer Safety, Reliability and Security (SAFECOMP 2011) pp 171-184.
- E. Quaglietta, L. D'Acierno, V. Punzo, **R. Nardone**, N. Mazzocca, A simulation framework for supporting design and real-time decisional phases in railway systems, 14th International IEEE Conference on Intelligent Transportation Systems (ITSC 2011), pp 846-851.
- **R. Nardone**, V. Casola, A complete methodology to evaluate Metro Systems Performability, InfQ workshop, Lucca 5th-6th July 2012
- S. Marrone, N. Mazzocca, **R. Nardone**, V. Vittorini, Combining heterogeneity, compositionality and automatic generation in formal modeling, The Workshop on Research and Use of Multiformalism Modeling Methods (WRUMMM 2012), Sept 2012, London.
- S. Marrone, N. Mazzocca, **R. Nardone**, R. Presta, S. P. Romano, V. Vittorini, A SAN based modeling approach to performance evaluation of an IMS-Compliant Conferencing Framework, Transactions on Petri Nets and Other Models of Concurrency, Lecture Notes in Computer Science (ToPNoC 2012), pp 308-333.
- S. Bernardi, F. Flammini, S. Marrone, N. Mazzocca, J. Merseguer, **R. Nardone**, V. Vittorini, Enabling the Usage of UML in the Verification of Railway Systems: the DAM-Rail Approach, Reliability Engineering & System Safety (RESS 2012).
- S. Marrone, **R. Nardone**, A. Tedesco, P. DAmore, V. Vittorini, R. Setola, F. De Cillis, N. Mazzocca, Vulnerability Analysis and Modeling for Critical Infrastructure Protection, 7th Annual IFIP Working Group, International Conference on Critical Infrastructure Protection, George Washington University Washington, DC, USA.

The following research papers are related to this thesis but were not included:

- E. Quaglietta, V. Punzo, B. Montella, **R. Nardone**, N. Mazzocca, Towards a hybrid mesoscopic-microscopic railway simulation model, 2nd International Conference on Models and Technologies for ITS, 22nd-24th June 2011, Leuven, Belgium.
- F. Flammini, S. Marrone, N. Mazzocca, **R. Nardone**, V. Vittorini, Model-Driven V&V Processes for Computer Based Control Systems: A Unifying Perspective, Leveraging Applications of Formal Methods, Verification and Validation. Applications and Case Studies, Lecture Notes in Computer Science, (ISoLA (2) 2012), pp 190-204.
- S. Marrone, **R. Nardone**, A. Orazzo, I. Petrone, L. Velardi, Improving Verification Process in Driverless Metro Systems: The MBAT Project, Leveraging Applications of Formal Methods, Verification and Validation. Applications and Case Studies, Lecture Notes in Computer Science, (ISoLA (2) 2012), pp 231-245.
- M. DArienzo, M. Iacono, S. Marrone, **R. Nardone**, Estimation of the energy consumption of mobile sensors in WSN environmental monitoring applications, The 27th IEEE International Conference on Advanced Information Networking and Applications (AINA-2013), Barcelona, Spain, March 25-28, 2013

Keywords: Critical Systems, Formal Model-Based, Model-Driven, Domain Specific Languages, Model Transformation, Model Template, Model Composition, Railway Systems, Conferencing Systems, Sensor Networks.

Contents

List of Figures

Acronyms and Abbreviations

ATL	**A**TLAS **T**ransformation **L**anguage
BN	**B**ayesian **N**etwork
CI	**C**ritical **I**nfrastructure
CIP	**C**ritical **I**nfrastructure **P**rotection
CPT	**C**onditional **P**robability **T**able
DAM	**D**ependability **A**nalysis and **M**odeling
DSL	**D**omain **S**pecific **L**anguage
DSM	**D**omain **S**pecific **M**odeling
DSML	**D**omain **S**pecific **M**odeling **L**anguage
FCC	**F**ailure **C**onsequence **C**lass
FMEA	**F**ailure **M**ode and **E**ffects **A**analysis
FMECA	**F**ailure **M**ode, **E**ffects and **C**riticality **A**analysis
FT	**F**ault **T**ree
GSPN	**G**eneralized **S**tochastic **P**etri **N**et
IMS	**I**P **M**ultimedia **S**ubsystem
M2M	**M**odel to **M**odel
M2T	**M**odel to **T**ext
MDE	**M**odel **D**riven **E**ngineering
NFP	**N**on-**F**unctional **P**roperties
OMF	**O**sMoSys **M**ultisolution **F**ramework
OMM	**O**sMoSys **M**ultiformalism **M**ethodology
PN	**P**etri **N**et
QoS	**Q**uality **o**f **S**ervice

RAM	**R**eliability, **A**vailability and **M**aintainability
RB	**R**epair **B**ox
RFT	**R**epairable, **F**ault and **T**ree
SM	**S**ervice **M**ode
SN	**S**ensor **N**etwork
SPN	**S**tochastic **P**etri **N**et
SRN	**S**tochastic **R**eward **N**et
UML	**U**nified **M**odeling **L**anguage
VAM	**V**ulnerability **A**nalysis and **M**odeling
VSL	**V**alue **S**pecification **L**anguage
WSN	**W**ireless **S**ensor **N**etwork

To my grandmother Elisa. . .

Part I

Modelling Critical Systems: State of the Art and Open Issues

Chapter 1

Introduction to Critical Systems

Critical systems are present in our daily life and affect many aspects providing essential support to several human activities, from transportation to industrial automation systems. Often a critical system is also complex since these systems are typically a multifaceted amalgamation of technical, information, organization, software and human (users, administrators and management) resources. Complexity of such systems comes not only from its involved technical and organizational structure but mainly from complexity of information processes that must be implemented in the operational environment (data processing, monitoring, management, etc.). critical systems, often, need to meet several requirements, often dictated by international standards, whose fulfillment must be demonstrated in order to achieve necessary certifications. The application of formal methods in industry (highly recommended, if not mandatory) is slowed down by models complexity and heterogeneity, and by the need to have high skilled personnel involved in model development. In other words, there is a request for modeling methodologies and supporting tools which can hide the complexity of the modeling process, without losing expressive power and solving efficiency.

In this Chapter we introduce critical systems, their characteristics, the issues related to their implementation and the need for quantitative evaluations of their properties.

1.1 Critical Systems

There is a surprising amount of disagreement concerning the definition of a critical system that involves a diversity in development methodologies. In [1] critical systems are classified according to the development approach, and four different view are described: the first, which can be called the *dependability* approach, is oriented to the development of ultra-reliable and fault-tolerant systems; another, which we can refer to as the *safety* approach, is oriented to the development following the principles of system safety engineering; the third is the *security* approach aimed at developing systems that prevent unauthorized disclosure of informations; at last the *real-time* approach has the aim to develop system strongly constrained to deadlines of performed activities. With this in mind, the same author gives a definition of critical systems as the one that derives from the system safety engineering tradition: "a critical (computer) system is one whose malfunction could lead to unacceptable consequences". The same definition of critical systems is shared by J. C. Laprie, as reported in [2].

The typology of "unacceptable consequences" depends on the application context and could include substantial financial losses, environmental damages, injuries or death of human beings. One of the criteria by which critical systems can be classified is therefore dictated by the kind of consequences that result from their failure:

- *safety critical*: systems whose failures can result in physical damage to human lives and the surrounding environment which, in turn, can result in personal injury and/or loss of lives;
- *security critical*: systems whose failures does not ensure the integrity and confidentiality of managed data;
- *mission critical*: systems whose failures do not ensure the mission accomplishment;
- *business critical*: systems whose failures causes significant losses in economic terms or a great damage in terms of commercial and business appearance.

Examples of *safety critical* systems are those controlling airplanes, medical devices or nuclear control centre; between *security critical* systems can be found banking systems, which allows you to make online transactions: any alteration of sensitive information relating to customers' accounts due, for example, to hackers intrusion, involving a breach of confidentiality and integrity, can lead to disastrous consequences. Navigation systems of a spacecrafts without crew can be classified as *mission critical* systems, since its loss will not damage in terms of human lives, but certainly causes serious damage for the company appearance; web-hosting systems can be classified as *business critical* since a failure can cause the impossibility to reach stored websites and the payment of penalties to clients.

Obviously this classification, also if diffused, it is not universally accepted: some variations to this classification are adopted, but all of them are based on the analysis of the failure consequences. Furthermore a system cannot be classified in a mutually exclusive way in one of this category, since it can belong contemporary to more than one category:

an example is a railway system that is *mission critical* from the service point of view but it is also *safety critical* since a failure can cause several damages to environment and people and lastly it can be *business critical* since its dependability attributes are strictly related to economic penalties to pay in case of not reached targets.

Critical systems are expected to satisfy a variety of specific characteristic, that we can address as "critical properties". These characteristics are strictly related one to each other, since an improvement of one of them can have an impact on others (Figure 1.1).

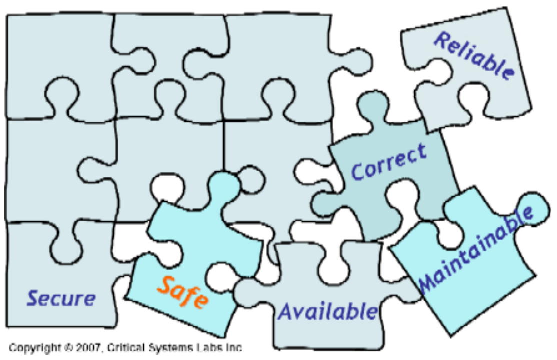


FIGURE 1.1: Attributes of a Critical System

Between critical properties, those requiring analysis and assessment concern both performance characteristics and the dependability attributes. While the performance is strictly related to the service levels provided by the system, the dependability of a system is formally defined as the ability of a system to provide a service that can be considered reasonably trusted [2, 3]. The dependability has several sub-attributes, in part listed below, with respect to the different meanings of service and a system trustworthiness, the most important attributes are those summarized in the acronym RAMS (i.e. reliability, availability, maintainability and safety):

- *availability*: ability to provide the service when required, an index of which is the asymptotic Availability (A);
- *reliability*: ability to provide service continuously over time, whose index is the Mean Time To Failures (MTTF);
- *maintainability*: ability to be subjected to maintenance actions (repairs, reconfigurations), an index of which is the Mean Time To Repair (MTTR);
- *integrity*: ability to deny undesired modifications of stored information;
- *confidentiality*: ability to deny access to unauthorized personnel (including attackers or even accidental intruders) from stored information;

- *security*: absence of catastrophic behavior for the data system environment in which the system works;
- *safety*: absence of catastrophic behavior for people and environmental things, an index of which is the Mean Time Between Hazardous Events (MTBHE).

1.2 Analysis of Critical Systems

Critical systems have not only to satisfy several properties simultaneously, they also must demonstrate their compliance with quantitative targets; the analysis, prediction and assessment are fundamental for their proper operation and effective use, which shall convince about their suitability for the tasks, increasingly crucial and critical, for which we use them. During the last 20 years, a multitude of methods and tools supporting evaluation of critical systems have been proposed. Since qualitative observations are more subjective and can be affected by people interpretations, the focus of these approaches is on quantitative evaluations: the aim is to quantitatively assess a specific characteristic of the system. Quantitative evaluations are performed using scientific tools and measurements while qualitative ones are also harder to reproduce with accuracy: two people qualitatively evaluating the same thing may end up with different, and also conflicting, results, while quantitative evaluations shall lead to the same result, also if performed by different people and in different time. The main benefit of quantitative evaluation is that they help to remove human bias from a statistic, making it more reliable than informations gathered qualitatively. For this reason critical systems needs to rely on quantitative analysis, when it is possible. Furthermore, the use of quantitative methods becomes mandatory for the analysis of properties that are subject of a contract and/or norms: in this cases it is important to formally demonstrate the compliance with target values. Example of properties which need quantitative assessment are dependability attributes, previously described.

The two main approaches for quantitative evaluation are the usage of models and direct measurement (often in test environments) [4]. While the firsts require a model of the system behavior, the latter approaches are based on the development of the system or of its prototypes. With specific focus on the latter, in literature we often talk about "formal Model-Based" when we want to address the use of formal models (where the word "formal" is used in the sense explained in the following). The following subsections will give a brief overview on these methods, offering a wider focus on the firsts.

1.2.1 Direct measurement

The direct measurement allows to use a hand-held instrument to directly measure a feature of the system; this technique gives an answer to your data straight from the experiment rather than calculating values relying on system models. The direct measurement approach faces two difficulties: the first is on the accuracy of the test environment in terms of reproduction of the circumstances that will be encountered during service; the second regards large number of tests required in a great part of situations. If we are looking for very rare failures occurring in specific service circumstances, it will be necessary to subject the system to tests in a highly realistic test environment. Furthermore, it will be clearly necessary to submit the system to very large numbers of tests (proportional to the expected value) and, if we are dealing with a reactive system, then a test input must be a whole trajectory of inputs that drives the system through many states. If we are dealing with a component of a larger system, then it will also be necessary to conduct tests under conditions of single and multiple failures of components that interact with the system under test. Obviously, it is very expensive to set up and run such a test environment, and very time consuming to generate the large and complex sets of test inputs required. As an example, imagine we wish to analyze a failure rate of a component of approximatively 10^{-6} failures per hour: we need to test the component for about 1 million of hours to see one failure, and, after a failure, it is necessary to reset the state of the component to enable another failure occurrence. Failure, at last, can be also caused by a failure of another system component, so the design of test environments is not so trivial.

1.2.2 Formal Models

Formal models are used to provide a rigorous definition of the concept of proof. They are able to combine low-level informations (such as failure rate at component level) with the aim to evaluate the characteristic under analysis. Obviously the system complexity is reflected in model complexity: the greater is the accuracy of the model, the greater is the model complexity (both in their development and in their solution). They must rely on an unambiguous specification of the system both in terms of variables (states) than in those regarding operations (functionalities). A model solution can be performed through analytical and simulative approaches. The characteristics of these two methods are specular: the advantages of the analytical solutions reside essentially in their capability, where it is possible, to furnish an exact solution of the model, while the simulative approach can provide a solution in whatever situation (also where the firsts fail) but they are affected from an error that is in inverse proportionality with the number of simulations, hence the time of the solution process.

Between formal models, **Combinatorial** and **State-based** models are distinguished. **Combinatorial** models adopt a logic connection between events: they allows to obtain quantitative measures specifying the system failure condition from failures of its subcomponents in the most convenient way. In detail, this type of models allow to calculate the probability that occur a certain event, usually called *top event*, from more simple events (*basic events*) connected through logical relationship. In the dependability analysis, events represent the operational state of the system or of its components, as the probability to provide correctly a service or the occurrence probability of catastrophic events. The state of the system can be described from two points of view: it is possible to describe the conditions of properly functioning, or it is possible to describe the conditions that lead to a system failure. It is commonly known that the second choice is more convenient than the first [5]: it is easiest to determine the conditions leading to a system failure than those maintaining the system up and properly running, especially if we consider different service levels. Example of formalisms supporting these methods are Reliability Block Diagrams, Fault Trees and Reliability Graphs.

On the other hand **State-based** models describe the system as a finite state space, transitions between these states and randomly distributed sojourn times in the states. Additionally, a reward function associates a numerical value (e.g. 0 for a failed state and 1 for good state in the simplest case) to each state. State-based models are supported by **Markov Chains**: they are mathematical formalism that undergoes transitions from one state to another, between a finite or countable number of possible states. It is a random process in the simplest version characterized as memoryless: the next state depends only by the current state and not by the sequence of events that preceded it. This formalism helps evaluation of the probability to be in a state j after k step. Markov chains are commonly used to dependability evaluations of systems whose components exhibit strong dependencies. With respect to combinatorial models, they cannot assume the total component independence. Used alone, these methods may lead to optimistic predictions for the system reliability and safety parameters. Some of the advantages are simplistic modeling approach, redundancy management techniques and possibility to insert failure coverage. Alternatives to Markov Chains are **Petri Nets** [6], widely used for modeling and evaluation of dependability, performance and performability of critical systems. This formalism allow to model the structure of a system or its behavior over time, including sequencing, parallelism, competition, distribution, non-determinism, synchronization, cooperation, randomness and time. Petri Nets describe quantitative aspects both in terms of graphical representation (similar to flowcharts or block diagrams) and in terms of mathematical models (for example, by means of state equations or mathematical equations). Some extensions of Petri Nets have been proposed over time: starting from Petri Nets with priority mechanisms, it have been added probabilistic and temporal annotations. Among the most common formalisms we remember: Stochastic Petri Net (SPN) [7], Generalized Stochastic Petri Net (GSPN) [8], Stochastic Reward Net (SRN) [9] and Stochastic Activity Network (SAN) [10]. These formalism can model both exponential and non-exponential temporal delays. Some tools support the analysis of the different formalisms: between them, the Möbius [11] framework support the analysis of SANs and it will be used for most quantitative models evaluation in this thesis.

1.2.3 Multi-formalism

With increasing complexity of the system under analysis, a single modeling formalism is revealed often inadequate: in particular some real-world systems are characterized by the inability of a single formalism to capture all the properties of a system, therefore the application of formal methods focuses on a few characteristics at a time, often maintaining a level of abstraction that is not always satisfactory for the validation of the system. The **multi-formalism** approach allows the realization of models with different independent formal techniques, exploiting advantages of each of them. It offers the opportunity to choose the most appropriate modeling technique to represent different components of a real system in order to build a single model. This obviously requires both a theoretical effort, which provides the foundation for integration and interoperability between different formalisms, and the definition and the development of tools and environments for modeling and solving multi-formal models. It is possible to recognize two approaches to multi-formalism: explicit and implicit. The first approach makes visible multi-formalism to the user level, which can choose to use explicitly different formal models to describe different parts of the model; the latter is achieved by the use of the framework in order to solve the model and it is not visible to the user, which describes the model through a formal language that allows you (implicitly) to define which parts are intended to be expressed in different formalisms.

The SMART project [12] integrates the following formalism: Stochastic Petri Net, Markov Chain in Continuous Time and Discrete in a unique environment modeling. The various models can exchange data during the process of analysis.

SHARPE [13] is a tool for the reliability and performability analysis that integrates different formalisms such as Fault Trees, Generalized Stochastic Petri Nets, some types of Queuing Networks and Markov Processes; also this environment allows the exchange of information and results between the various models during the analysis phase.

The Möbius approach [11] is more general than the other two providing a three-stage approach to model development: the first stage is the development of an atomic model in a specific formalism, such as Stochastic Activity Network (SAN), Bucket and Balls Formalism and Performance Evaluation Process Algebra (PEPA). The second stage is the development of a composed model where sub-models can be both atomic models and other composed models. To this aim the approach uses two operators: the *replica* and the *join* of sub-models. The third stage consists in building a reward model that is the process with which reward variables, defined on atomic models, are evaluated. Each reward variable is evaluated, with a specific confidence level, starting from analysis of the model, and it can refer to a interval-of-time or to a instant-of-time. The solution of the model is performed by different solvers that are available in Möbius framework, translating the overall composed model into a specific Abstract Functional Interface (AFI): this approach give extensibility features to the approach, allowing the definition of new formalisms and solvers which can be added to the framework. Obviously, even if the AFI is sufficiently general to allow the definition of different formalisms, it does not allow the definition of a whatever formalism but they remain confined to a set of "compatible" formalisms.

AToM3 [14] implements a modeling approach based on meta-modeling and processing techniques including graphs. It allows to model different system components using different formalisms and its user interface provides a graphical editor for each defined formalism. The models can be translated from a formalism to another thanks to a special grammar on graphs, which defines the transformation rules. The analysis can be performed either applying simulative techniques but also translating all the models in a common formalism and subsequently applying the solver specific.

The DEDS (Discrete Events Dynamic Systems) approach [15] provides a framework for the construction of performance models of discrete-event systems through a graphical interface. The used formalisms are: Queuing Network, Petri Net, Coloured Petri Net. The analysis is performed by translating all the sub-models in a formalism similar to that of Petri Net: the resulting model is then analyzed through the construction of the state space.

OsMoSys (Object-based multi-formaliSm MOdelling of SYStems) [16] is a multi-formalism and multi-solution framework for the model-based analysis of systems. The modeling methodology, also referred to as OMM (OsMoSys Multiformalism Methodology), is based on the meta-modeling and meta-languages principles (there are three levels of models: models, meta-model, meta-meta-model). This methodology is applicable to graph-based formalisms such as Petri Net, Fault Tree and Bayesian Network, and imposes further aspects of object-orientation. Furthermore OsMoSys is highly extensible since the developer is able to integrate his own formalism, opportunely producing the XML description of the formalisms he needs.

OsMoSys is also a multi-solution framework since it integrates not only different kinds of formalism, but it provides the possibility to choose which solver shall be used for a specific formalism (e.g numerical, analytical, symbolic, etc.). This aspect of OsMoSys is also named OMF (OsMoSys Multisolution Framework). The different solvers are integrated in OsMoSys through the use of adapters, that are ad-hoc software entities which can perform the interaction between the specific solver and OsMoSys through a common interface. The solution process is based on the coordination of these adapters performed by a workflow engine.

1.3 Research trends in quantitative evaluation

Research trends in quantitative evaluation are essentially the following:

- *divide-and-conquer*: this approach focuses on methods and techniques for developing models through the composition of several sub-models, each one modeling a manageable portion of the system;
- *multi-formalism* and *multi-paradigm*: this technique explores the possibility of combining sub-models developed through different formalisms and modeling paradigms towards an overall model of the system;
- *automatic generation*: this technique aims at generating formal models, suitable for the analysis, from high-level specifications.

In the first approach a model consists of several sub-models tied together by appropriate rules and composition operators: in order to provide appropriate methods for solving sub models and aggregating results, composition techniques and operators definition are necessary. In [17] a comprehensive review of the state of the art in such techniques is presented. These methods include techniques to limit the size of the state space (*largeness avoidance*) and techniques to manage the size of the models (*largeness tolerance*).

The second approach has been discussed sufficiently in Section 1.2.3.

The third approach deals with the generation of formal models from high-level specifications. As surveyed in [18], scientific community has also explored such way and several approaches are present in literature: in [19, 20] the generation of Fault Tree and Dynamic Fault Tree models from a set of UML diagrams is described; in [21] UML

Diagrams are used to generate Timed Petri Nets-based models; generation of Stochastic Reward Nets from Statechart and Activity Diagrams are presented respectively in [22] and [23]. In [24] the automatic generation of Repairable Fault Trees is obtained starting from an high-level system description compliant with the DAM profile (Section 3.2). Another example is in [25] where Stochastic Reward Nets are generated starting from a SysML description of a system with rejuvenation.

1.4 Evaluation of Critical Systems in industrial context

The evaluation of critical systems in industrial context is necessary to control the behaviour of the system during all the lifecycle, to predict the impact of different architectural choices, to estimate critical properties that need to be compared with target values defined in contracts or in regulations. Often direct measurement cannot be performed on real critical systems, since their complexity and high development costs do not permit the production of prototype artifacts. It is possible only to apply test cases in test environments, but their accuracy, in terms of reproduction of the operation circumstances, needs to be verified. These tests often regard functional properties of the system and of the sub-systems, while non-functional properties are assessed using analytical or model-based approaches, and follows specific lifecycles defined by international norms (e.g. CENELEC 50126 [26]). In particular, models are mainly adopted during the verification of design constraints and for the prediction of performance and dependability parameters, in relation to structural changes and implementable strategies during degradation situations.

To date analytical models are preferred in industrial field: their usage has the enormous advantage of making unnecessary the presence of skilled personnel on formal languages, but they are based on assumptions that are difficult to prove (e.g. strong independence assumptions among the system components) and excessive simplifications of the consequences of failures on the service performed by the system. These disadvantages instead, represent the advantages of the formal models: their application makes feasible the contemplation of internal dynamics, even if complex, and can lead to more accurate results. For this reason they are used in safety analysis, where a formal proof of the compliances is required.

Between formal models, the combinatorial methods are the most used since they are very user-friendly: the graphical representations of these models are very intuitive, and models can be step-wisely refined and can be easily modified. State-based models, also if more powerful, are not so used leading to impossibility to model partial component failures. As a consequence, models of systems are often too simplified and therefore results are overlay approximated with respect to the real behavior (the modelers tend to produce excessively conservative models, underestimating the interest parameters, and promising values too low than those allowed by technology). If underestimation can be tolerated in safety analysis, it can not be the same for the performability analysis: the underestimation of service availability indexes will primarily cause the proposal of too low values in tender phase, hence a reduced probability to win the contract against competitors. The approximation is a problem also during design stages since it does not allow to discriminate between different solutions, and designers do not trust in these results.

At last, the analysis of the Quality of Service as a measure of the service offered to system users, is almost never considered in industry: the QoS is thought as a direct consequence of the performability level reached by the system. In particular, in [27], the level of QoS expected defines the metrics and target values of performability indexes; the system, during the service, reaches certain levels of performability, measured according to the defined metrics, which are considered as directly connected to QoS levels. This relationship need to be verified: this is very difficult since performability can be quantitatively measured while QoS could depend on qualitative evaluations.

Furthermore the techniques showed in Section 1.3, as stated also in [28], are rarely adopted into a broader industrial development process: the development of business transformations is not engineered enough and it is often related to a specific application domains, resulting in inflexibility and low reusability. There is the lack of a general framework in which Model-Driven approaches support formal Model-Based analysis. This need has been collected from several European project as SATURN (SysML bAsed modeling, architecTUre exploRation, simulation and syNthesis for complex embedded systems [29]) and the ARTEMIS JU-CHESS (Composition with guarantees for High-integrity Embedded Software components aSsembly [30]) whose goal is to create a framework for critical system analysis using model-driven approaches, that support the entire development cycle of systems.

1.5 Evaluation of properties during the lifecycle of the system

The high levels of complexity make the design and the development extremely difficult. Maintenance is necessary to ensure that the predetermined requirements continue to be met during the service. In fact, during the design of a critical system, system behaviour should be coherently analysed: critical properties must be coherently assessed during all the lifecycle of the system, to evaluate the impact of different choices on overall system characteristics. Critical systems are, therefore, a challenge for the design, construction and operation, requiring an engineering approach that covers the entire lifecycle of the system: the System Engineering [31]. This approach consists in:

- identification and quantification of system goals;
- development of alternative conceptual projects;
- selection and implementation of the best solutions;
- verification of the project with respect to its requirements;
- evaluation of the system capacities to reach the defined targets.

The overall aim is to ensure system development and integration to meet the needs and expectations of stakeholders, within acceptable limits of cost (in terms of Total Cost of Ownership, Life Cycle Cost, etc.), time and risk. Among the basic concepts of System Engineering we can remember:

- Holistic view of the system: the understanding of the system can not be based on the understanding of the individual parts, but on a global vision considering the problem as a whole, i.e. the whole lifecycle of the system;
- Emergent behaviors: the behavior of the system is not the sum of the behavior of its parts, but the interaction of these results in the manifestation of behaviors and properties not directly related to the individual parts;
- System boundary: the boundary of the system need to be extended during the analysis to include interactions with other systems that can influence behavior;
- Multiview and multiobjective analysis: the system shall be considered from different points of view with different objectives in order to obtain a better comprehension of the whole system.

The various development stages are often regulated by international standards; these standards impose verification processes on the client and/or certification authorities, called to verify adherence of the system to the standards and verify proof of the reliability degree of the product according to qualitative and, especially, quantitative analysis. For these reasons the lifecycle of a critical system can be considered critical itself. Each organization has its own vision of the concept of System Engineering and proposes a different model of the system lifecycle. There are, then, various models including: EIA-632, EIA/IS-632, Vee proposed by INCOSE, 61508 (safety) and CENELEC EN 50126. These models can be cascaded, iterative, incremental and evolutionary.

The proposed lifecycle are extension of the waterfall model and some of them are based on the V-Model: instead of descending along a straight line, after the step of construction goes back with a typical shape of a V. Typical phases V-Model are outlined below:

1. *Requirements Definition*: this specifies the set of elicited requirements for the system which may include individual actions to be carried out by main components.
2. *Functional Design*: the system is designed in terms of models, diagrams. The logical structure of the system is designed in this phase.
3. *System Partitioning*: the system design is broken up into a number of independent modules which can be developed separately by a number of suppliers.
4. *Hw/Sw Development*: the individual sub-systems are developed and implemented by the various subsystem suppliers. Note that some modules may still be developed in-house by the system integrator.
5. *Testing*: each subsystem is tested in isolation to ensure that it fulfills the requirements laid down.
6. *Functional Integration*: subsystems which work together to provide some function are integrated together and tested to ensure that they fulfill the requirements for that function and that they will operate correctly as an integrated unit.
7. *Integration*: the separate systems are integrated together to produce the complete system architecture. The full architecture is then tested to ensure that all of the systems operate correctly as a complete implemented architecture.

As a practical example, in the following, the model proposed by CENELEC EN 50126 norm is described; this model is applied for the assurance of RAMS properties in European railway systems.

1.5.1 CENELEC 50126 lifecycle

The CENELEC EN 50126 [26] is an European Standard that defines a process, based on the system lifecycle and tasks within it, for managing RAMS: its adoption leads to the development of railway which achieve specific RAMS requirements. The lifecycle defined in this norm facilitates the assessment of RAMS interactions between elements of a complex railway applications. The same norm promotes co-operation between a railway authorities and railway support industry in the achievement of an optimal combination of RAMS and cost for railway applications.

The model of the lifecycle proposed by CENELEC standard EN 50126 is shown inFigure 1.2.

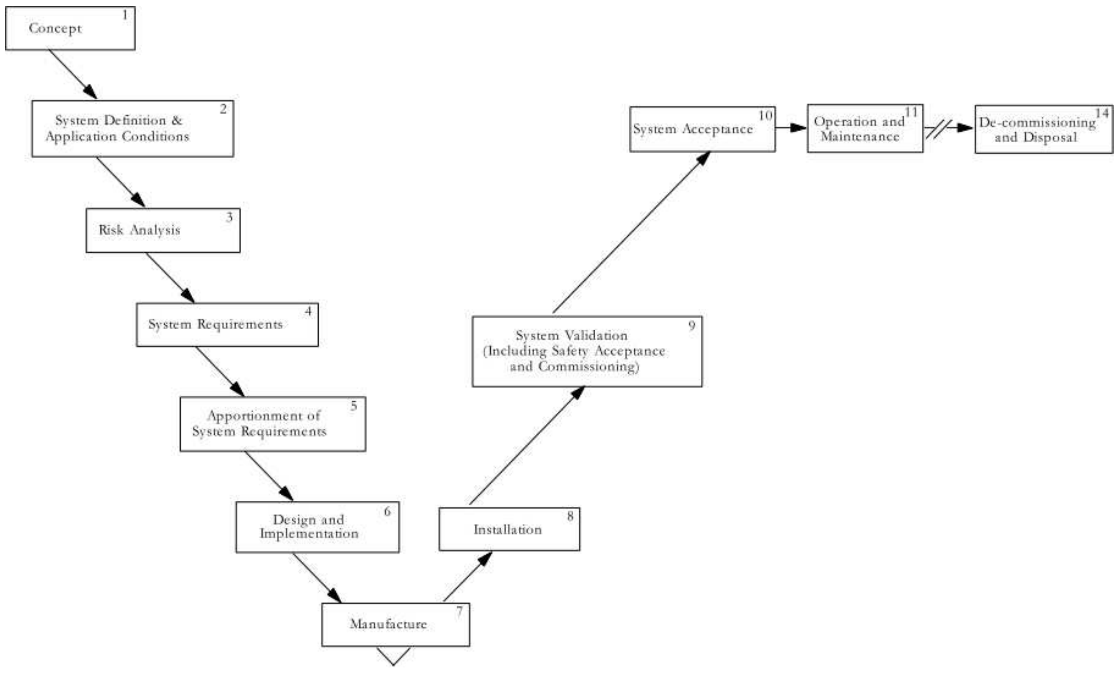

Concept 1
System Definition & Application Conditions 2
Risk Analysis 3
System Requirements 4
Apportionment of System Requirements 5
Design and Implementation 6
Manufacture 7
Installation 8
System Validation (Including Safety Acceptance and Commissioning) 9
System Acceptance 10
Operation and Maintenance 11
De-commissioning and Disposal 14

Figure 1.2: V-model defined in CENELEC EN 50126

The typical structure based on a "V" organizes development phases into levels of complexity with the most complex item on top and least complex item on bottom. This places the design next to verification in the sense that the acceptance activities are intrinsically linked to those of development.

This model defines 14 phases in the lifecycle, starting from the concept to the decommissioning of a railway system, specifying inputs, requirements, deliverables and verification technique of each phase. These phases can be grouped into four macro-phases, which are:

- *Definition*: the objective of this phase shall be to develop a level of understanding of the system sufficient to enable all subsequent RAMS lifecycle tasks to be satisfactorily performed. In this phase are defined: the mission profile, the boundaries, the application conditions influencing the characteristics of the system. At last, also in this phase, the overall RAMS requirements for the system are defined as well as the overall demonstration and acceptance criteria for RAMS for the system.

- *Design*: during this phase the overall RAMS requirements are apportioned to all subsystems and specific acceptance criteria are defined at subsystem level; subsystems conforming to RAMS requirements are hence created and their compliance with RAMS target is demonstrated through the usage of proper formal models (combinatorial as Reliability Block Diagrams, Fault Trees and also state-based models).
- *Testing*: after the realization of the subsystems, the objectives of this phase are to validate that the total combination of sub-systems, components and external risk reduction measures comply with the RAMS requirements for the system; in this phase RAMS specific characteristic are collected in order to assess compliance with the overall RAMS requirements of the complete system, and hence accept the system for entry into service.
- *Operation*: the objective of this phase shall be to operate (within specified limits), maintain and support the total combination of subsystems, components and external risk reduction measures such that compliance with system RAMS requirements is maintained; the RAMS performance of the system shall be monitored in order to maintain confidence in the system.

1.6 Thesis contribution

The original contribution of this thesis is in the definition of a structured approach to quantitative analysis of a critical system. In particular, to cope with the problematic of industrial adoption of formal Model-Based quantitative analysis, this thesis combined formal Model-Based analysis with a Model-Driven approach aiming at developing necessary formal models. More in detail, according to such approach, critical systems properties can be assessed starting from high level models expressed in proper languages (usually called Domain Specific Modeling Languages - DSMLs), and defining automatic transformation chains, able to generate formal models. The methodology is implementable into cost-effective processes in order to reduce time-to-market specifically during verification stages of critical system development lifecycles. Such objective will be achieved by both improving usability and re-usability of typical Model-Driven artifacts.

Another important contribution of this thesis is in the definition of a structured methodology to the development of formal models of the system under analysis: a reference architecture that organizes formal models is defined.

This thesis defines also th new formal operator of *Model Template*, and new techniques are addressed in order to help modelers during the model development phase.

Even if the focus is on quantitative evaluation, the proposed Model-Driven methodology is useful also for the complete design (based on centralized information), hence it is possible to extend it for automatic generation of documents and, more generally, of all those artifacts which support system development.

Chapter 2

Model-Driven approaches

Model-driven engineering (MDE) is a software development methodology which focuses on creating and exploiting domain models (that are representations of knowledge and activities that govern a particular application domain), rather than on the computing (or algorithmic) concepts. Model-driven engineering technologies offer a promising approach to alleviate the complexity of platforms and express domain concepts effectively, supporting the automatic generation of artifacts. This Chapter outlines the history of Model-Driven, starting from software engineering area (where it bore) and arriving to system engineering. Concepts of language definition and transformations are also pointed out with a specific focus on UML profiling technique.

2.1 Model-Driven principles

During the past five decades, software developers have tried to create abstractions that help them to code in terms of what they wanted to design rather than relying on what the developing environment offered them [32]. Since the dawn of software engineering, such abstractions included both languages and technology platforms. For example, the first programming languages, such as assembler or Fortran, abstracted developers from complexities of programming directly in machine language. Similarly, the first operating systems such as OS/360 and Unix, abstracted developers from the complexities of hardware programming. Despite that these languages and platforms increased the level of abstraction, they had always the goal to the elaboration. In particular they provided abstractions in the solution space (i.e. within the domain of the processing technologies) instead that in the direction of the problem space, which expresses the project in terms of concepts belonging to the application domain, which may be for example telecommunications, health care, biology, etc.

The successive efforts were directed towards the creation of new technologies with the aim to increase even more the level of abstraction in software development. An effort in this direction occurred in the '80s with the introduction of the so-called Computer-Aided Software Engineering (CASE), which focused on the development of methods and software tools that would allow developers to express their projects in terms of general-purpose graphical representations, such as state machines, structure diagrams, dataflow diagrams. One of the objectives of CASE was to permit further analysis of graphics programs that involve less complexity than conventional general-purpose programming languages. Another objective was to synthesize implementation artifacts from graphical representations to reduce the effort of coding, debugging and manual porting. Although CASE tools attracted great attention in the literature, it have not been widely adopted in practice. A problem was that the graphic representation of general purpose language for writing programs with CASE tools was not well supported by underlying platforms, which were largely single-node operating systems, such as DOS, OS/2 or Windows (which lacked support for major property management of QoS such as fault tolerance and security). The amount and complexity of the generated code, needed to compensate the scarcity of the underlying platforms, was beyond the understanding of translation technology available at the time, which made it difficult the development, the debug, and the evolution of CASE tools and of the applications created with these tools.

Another problem with CASE was its inability to manage complex mechanisms of large scale production, in a wide range of application domains. In general, CASE tools did not support the design concurrency, so they were limited to programs written by one person or by a team that serialize their access to the files used by these tools. Furthermore, due to the lack of powerful middleware platforms, the CASE tools were targeted to proprietary execution environments, which made it difficult to integrate the code generated by them with other software languages and technologies. The CASE tools did not support effectively either the application domains because their graphic representations, based on one-size-fits-all, were too general and not customizable.

As a result, CASE had a relatively weak impact on the development of commercial software of the '80s and '90s also because focused on a few application domains such as that of call processing in telecommunication systems, which is well represented, for example, as state machines. The usage of CASE were limited to a subset of tools that allow designers to draw diagrams of software architectures, that the programmers used as a reference during code editing. Since there was a direct relationship between diagrams and implementations, developers were not inclined to update diagrams coherently with the code, especially during the latter stages of the projects.

The innovations of the languages and platforms in last two decades have helped the increase of the level of abstraction, removing some of the limitations of previous CASE tools. For example, today's developers typically use the object-oriented programming languages such as C++, Java or C#, instead of Fortran or C. Today's class libraries and application frameworks reuses and minimize the need to reinvent the services that are common to specific domain such as transactions, fault tolerance, security or management of distributed resources. Hence, thanks to the maturation of third-generation programming languages and of the reusable platforms, developers are better equipped to hide the complexities of older technologies. Despite these advances, several problems remain. At the center of these problems there is the complexity growth of the platforms, which have developed faster than the masking capacity of general-purpose languages. For example, the most popular middleware platforms such as J2EE, .NET and CORBA, contain thousands of classes and methods with many intricate dependencies, which require hard planning and development effort. Furthermore, since these platforms are evolving rapidly and new platforms appear with regularity, developers spend a lot of time performing manual porting of application code on different platforms or developing new versions of the same platform.

An important problem is that most of the applications and codes for the platforms are written and maintained using the third-generation programming languages, which still have difficulties to take off in terms of time of deployment, configuration and guaranteed quality. For example it is certainly difficult to write Java or C# code that unfold in a correct and optimal distributed systems on a large scale with hundreds, and also thousands, of interconnected software components.

With these kinds of problems, the software industry is reaching the maximum complexity in the technologies of the next-generation platforms (such as web services) have become so complex that developers spend years to manage these APIs platforms and usage patterns, and often they become familiar only with the subset of the platforms they use regularly. In addition, the third-generation programming languages require developers greater focus on small details of the implementation, instead of letting them focus on issues such as accuracy or performance management.

This fragmented view makes it possible for developers to have a difficulty understanding of what portion of their applications will be susceptible to side effects, coming from changes in user requirements, environments and/or languages. The lack of a global overview, often leads developers to implement sub-optimal solutions that sometimes duplicate unnecessarily code, violate architectural principles and complicate the system evolution and its quality.

2.2 Model-Driven Engineering

A promising approach that is able to cope with the increasing complexity of platforms, and with the inability of third-generation programming languages to easy the complexity and express domain concepts in a light weight fashion, is to develop Model-Driven Engineering technologies. These technologies bring together two important aspects:

- domain-specific modeling languages (DSMLs), which formalize the application structure, behavior, and requirements within a specific domain. The DSMLs are described using meta-models (or domain models), which define the relationships between concepts in a domain, specifying the semantics and constraints that exist between these concepts. Developers use the DSMLs for building applications that use elements captured by meta-models, and express their designs in a declarative way rather than imperative;
- transformation engines and generators that analyze certain aspects of models and synthesize different types of artifacts, such as source code, simulation inputs, XML deployment descriptors, or representation of models. The ability to represent artifacts from models helps to ensure consistency between applications and analysis of information with the functional requirements and quality requirements captured by models.

New MDE technologies try to put into practice all efforts performed to increase the level of abstraction that have occurred so far. For example, instead of using general-purpose notations that rarely express concepts belonging to the application domain, the DSML may be constructed on an ad hoc basis through the meta-modeling exploiting precisely semantics and syntax of the domain of interest. Having graphical elements that are related directly with the familiar concepts of the application domain not only helps to lower learning curve, but it also helps the experts until user requirements are met with greater precision. In addition, MDE tools impose specific constraints for the application domain, so model can be verified, and errors are prevented, since the first phases of the project lifecycle.

2.2.1 Model-Driven terminology

In the last decade different methods based on models, that differ in some small but essential details, has followed one another. To better clarify these issues, we report those that are so-called the “manifest” of such methods, and provide a temporal and contextualization of their meaning [33].

- *Model-Driven Architecture (MDA)*: the first white paper of the OMG refers to MDA appeared in 2000. Later, in 2003, it was published the current version of the standard [34]. All model-driven initiatives follow the principle that "everything is a model". The current definition of MDA is present in [35] which states: "MDA is an OMG initiative that proposes to define a set of non-proprietary standards that will specify interoperable technologies with which to realize model-driven development with automated transformations. Not all of these technologies will directly concern the transformations involved in MDA. MDA does not necessarily rely on the UML, but, as a specialized kind of MDD (Model Driven Development), MDA necessarily involves the use of model(s) in development, which entails that at least one modeling language must be used. Any modeling language used in MDA must be described in terms of the MOF language, to enable the metadata to be understood in a standard manner, which is a precondition for any ability to perform automated transformations".

- *Model-Driven Development (MDD)*: the MDD was defined the first time in 2003 [36]. "Model-Driven Development is simply the notion that we can construct a model of a system that we can then transform into the real thing". An important difference between MDA and MDD that can be seen from the definitions is that MDD does not adhere to any standard OMG, but the main contribution of MDD lies in flexibility in defining development processes.

- *Model-Driven Engineering (MDE)*: three years later, in 2006, in the homonym article [32], MDE is defined as: "Model-Driven Engineering technologies offer a promising approach to address the inability of third-generation languages to alleviate the complexity of platforms and express domain concepts effectively". From this definition we can understand how MDE is the evolution of CASE tool. The MDE initiative proposes a process definition wider and not limited to the development as for MDD; MDE methods and techniques are not only applicable to software systems but also to critical systems. In particular, the effort must be oriented to support the quantitative analysis, since they play a crucial role in the system lifecycle, in order to formalize and automate this activity. There are great benefits: the paradigm of systems design "construct-by-correction", typical of processes based on testing and verification of the late-time properties of a system, can be replaced by paradigms "correct-by-construction" where, the construction (and verification) of an initial model of the system and verification of automatic transformations involves a final model (in other words the system) automatically corrected. The effort of verification can thus be concentrated in the initial stages of the system lifecycle. Within the quantitative evaluation, outlined above, and within the scope of the techniques and methods of the MDE, a first scenario of application of these techniques to critical systems is clearly defined: there is the possibility to increase the spread of formal methods in industrial development processes in real systems. In fact, the ability to define high-level languages closer to the user (for abstraction and ease of use) as well as the ability to automatically generate models (formal models for quantitative analysis) from the first allows the usage of formal methods in a "transparent" way.

The schema of what said up to now is reported in Figure 2.1.

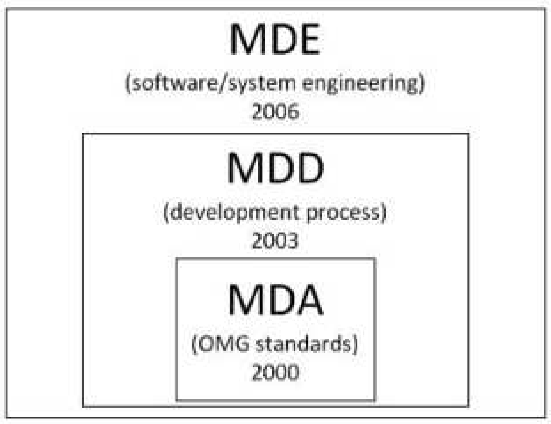


FIGURE 2.1: Model-Driven initiative representation

2.3 Domain Specific Modeling

DSM (Domain Specific Modeling) means a set of procedures and modeling artifacts that are specific to a given application domain [37]. They are different from existing techniques for general purpose since they use in the modeling directly concepts that belong to the application domain. These concepts and methodologies have been introduced to increase the level of abstraction with respect to the current programming languages. With the term application domain is intent both a technical domain such as persistence, user interface, communications, transactions, and a functional domain as a business domain of telecommunications, banking, insurance or retail sales. In particular the formers can be addressed as "horizontal" domains while the latter as "vertical". In practice, each DSM solution focuses on very small domains because in doing so we have a better chance to automate procedures.

In DSM, the models can be used to generate code as well as for the implementation, testing and debugging of applications. The models are the primary source of changes. However, the code does not disappear for all professionals involved, because there is a need of developers able to implement generators, provide framework and create reusable components and libraries.

One of the significant benefits of a DSM approach is that the concepts of the domain are already known and used by those who use the DSM solution: the semantics is considered known and developers do not need further concepts for the application domain which they are interested.

The DSM revisits the concepts of generators and transformations: generators provide the same type of automation given in the past (an example of generators are obviously compilers), but in the context of a specific application domain. This automation allows developers to focus primarily on the most important issues and on the functionality of the application rather than on implementation details.

In DSM large part of the testing is actually carried out prior to the modeling stage, because the language itself contain the rules and constraints of the domain. The rules must be in the modeling language to avoid generating code from incorrect models. Therefore, since the code is generated, the most common programming errors are eliminated.

The difference between a DSM approach and MDA is visible especially in the so-called agile development processes, in particular during the maintenance phase, where changes need to be performed on models previously created. In the MDA approach, main difficulties reside in making changes to models that have been partially created by the generators. The architecture of a DSM is essentially composed of three entities:

- *the language.* A domain-specific language provides an abstraction mechanism which allows to deal with complexity in a specific domain. This is done by providing concepts and rules in a language that represent “something” in the application domain, rather than “concepts” of a given programming language. In general, the main domain concepts are mapped to objects of the modeling language (i.e. elements that can be instantiated at modeling level), while others will be captured as properties of these objects, connections, or links. The language is defined through a meta-model with the relative notation and supporting tools.
- *the code generator.* A generator specifies how the informations that are extracted from the models are transformed into code. In the simplest case, each symbol of a modeling language produces a fixed code, including the values entered in the symbol as arguments. The generator can also produce different code depending on the values of the symbol, the relationships they have with other symbols, or other information in the model. This code will be linked with the framework and compiled into an executable program. During the creation of a working DSM

solution, the aim is that after the generation, the generated code should not be no more manually changed or extended. The generated code is then simply a sub-product intermediate on the way to the finished product, as, for example, the file *.o are during the C compilation.

- *the domain framework.* A domain framework provides the interface between the generated code and the underlying platform. In some cases, no additional coding operation is necessary: the generated code can directly call the components of the existing platform, whose services are sufficient. Often, however, you want to define additional code or utility components to make the generated code simpler. This framework can vary in the format and it can go from components to individual groups of statements of the programming language that can be commonly found in the code in the selected domain. These components may already exist from previous developments.

The generated code is not run by itself, but together with additional code in a given target runtime environment. This objective is provided with the platform code (the code that is already available with the target). This is used regardless of how the implementation is realized: manually or by means of generators. The developed product can use a selected part of a larger platform (for example, J2EE), a whole platform or several platforms.

2.3.1 Language

The language provides an abstraction for the development and it is the most visible artifact for developers. In DSM it is used to define specifications that manual programmers would treat as source code. If language is built correctly, it allows you to apply terms and concepts of a particular domain. This means that a domain-specific language is probably useless in other domains. For domain-specific languages, the same definitions that are adopted to languages in general can be applied. The modeling languages are composed of syntax and semantics. On the syntax, we can further distinguish between abstract and concrete syntax. The first indicates the structure and grammar rules of the language, while the second defines symbols of the notation and form of representation of the language. To increase the abstraction and to generate more complete code, it is usually necessary to extend both syntax and semantics.

The syntax. The syntax specifies the conceptual structure of a language: the constructs of a modeling language, their properties and their mutual connections. The syntax of a modeling language indicates something more than the classic “reserved words”: it also covers the grammar rules that must be followed when specifying models. In DSM these rules belong to the domain and are defined according to domain concepts. The rules are necessary to avoid to generate code from models with errors. The availability of these rules during the modeling phase, makes easier the implementation of the generators which therefore do not need to check the correctness of input models. In DSM rules are checked, as far as possible, during the early stages so as to detect and prevent errors in the moments in which their solution is inexpensive. Typically it is preferred to insert rules in the language rather than generators. This is most evident in the case where there are different generators starting from the same model: it is best to check the pattern once rather than doing it for each generator. The rules of the language typically impose constraints on how models should be created: they define allowed values, relations between concepts and how certain concepts should be used. The rules can vary from strict correctness rules and consistency checks on the models to rules that guide modelers, rather than impose, a particular way to model. Once the rules are defined, the modeling language ensures that all developers follow the same domain rules. Finally, if the spectrum of applications needs to be extended, the modeling language can be extended together.

The semantics. All concepts of the models represent something: they have a semantics. When you add an element in a model or a link between two elements, you create meaning in the model. In DSM, the semantics of the modeling language comes directly from the problem domain. Instead, in general-purpose languages, the semantics do not map specific problem domain, but is left to the developers to make the connections between semantics and concepts of the language of a given application domain, and often this connection is not well documented: as might be expected at this point, different people perform different connections, resulting in different models for the same problem. The semantics of the problem domain is not the only source of semantics for a DSM. As with all modeling languages, you must also recognize the semantic side implementation: how the modeling constructs are mapped in a given solution domain. This is usually called mapping operations. It is important to remember that syntax and semantics defined in a purely abstract way are not enough to define a language: the models need to be managed through a visual formalism. It therefore requires a concrete syntax in addition to the abstract: what is usually done, is to combine a textual and graphical representation.

2.3.2 Code generator

In DSM code generators convert the models into code that can be interpreted or compiled and executed. They provide a wide degree of automation contributing to the productivity and increasing the entire quality of the DSM approach. The generated code is typically complete from the point of view of the modeler. This means that the code is complete, runnable, and of good quality, in other words after generation, the code does not need to be changed or to be completed in any of its parts. This is possible because the generator (and the modeling language) is built to meet the demand of a small application domain. Obviously not all the code needed is generated. For this you need a domain framework and a target environment. The generators can be classified into declarative, operational and mixed. This classification is based on the approach used to specify the generators. Declarative approach describes the mapping between the elements of the source meta-model and the target programming language. The operational approaches, such as graph transformation rules, define the steps needed to produce the target code from a given source model. Basically, a code generator is a robot that accesses models, extracts informations and transforms them into a specific syntax. This whole process is driven by the meta-model, by the language modeling with its concepts, rules and semantics, and by the syntax required by the framework and from the target platform. The main problem of code generators is that they are not able to cope with the stringent requirements of size and execution efficiency which are key issues when developing software for devices with limited memory and limited processing capacity; this is the same problem that you have using compilers with efficiency of the compiled code: when the code produced by a generator is written by an experienced developer, this will produce higher quality code.

Different uses of code generators. Code generators can be used in alternative ways than the pure creation of executable code. In DSM, the generators can be also used to control the consistency and completeness of projects. This is necessary because it typically does not make sense and is often not even possible to put all the rules in a meta-model and to check them during each modeling activities. An example is the control of a model to verify if it is incomplete and to propose possible actions necessary to make it complete. The generators can also be used to produce prototypes instead for the production of code. The prototypes are usually used to give initial feedback and generators can produce prototypes for a target platform and in a programming language totally different than the final one. At this stage the generators do not require optimized code but the generated prototype needs only to show the functionality, usability and the main characteristics of the system under development. The generators can also be used to produce documentation from the same underlying models from which they produce code. The undisputed advantage is that the documentation does not need to be updated manually during the development phase.

2.3.3 Domain framework

A domain framework provides the interface between the generated code and the underlying target environment. It is not always necessary, in fact, for example, the generated code can invoke directly an interface of the target environment. First of all it is necessary to describe how the target environment is seen: it appears as a set of layers, some of which are optional (Figure 2.2).

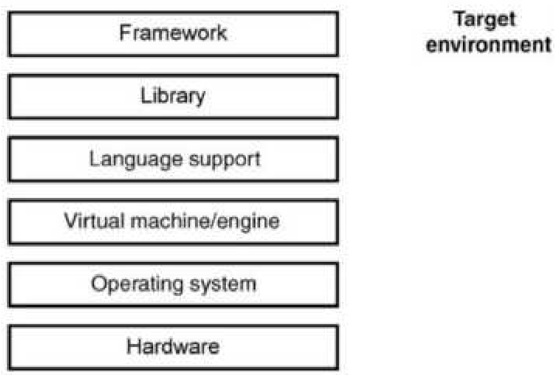


FIGURE 2.2: Target environment architecture

Starting from the bottom it is possible to appreciate as the abstraction level is gradually raised up. A framework can be seen as something that goes beyond the collection of libraries. It is an abstract specification of a type of application and it relies on a certain programming model that the developer must follow. Frameworks are typically classified into white-box and black-box. In a white-box framework user adds functionalities to the existing code: this requires that some implementation details are known and well understood. On the other hand, in a black-box framework, its internal implementation is not visible and new functionalities are developed separately.

2.3.4 DSM development process

The DSM process can be seen as composed of four main steps: initial development, deployment, operation, and maintenance. All these steps require, in order, the realization of the following acts or artifacts:

- *concept proof.* The concept proof is necessary if the application feasibility of the DSM methodology towards a specific application domain must be proved to third-party (the employer, for example). For this demonstration you choose a well-defined application domain where the boundaries are well defined in order to develop quickly a concept proof. The DSM solution should not be necessarily wide but still large enough to give a concrete demonstration for the selected domain;

- *the pilot project.* When the feasibility study of the DSM approach has been verified, it follows a pilot project. In the pilot project a first version of DSM is defined, implemented and tested. At this stage DSM users are more involved and some of them are using your solution in an real world example. The pilot project helps in estimating the effects of an DSM and prepares the third part to an integration with existing design patterns through the creation of tutorials and sample projects, training materials and coaching to people who already have experience in the DSM field;

- *usage of the DSM solution.* The deployment phase is when the DSM solution is adopted in production conditions. From now on the third party start getting the benefits of having raised the level of abstraction and of having automated the production process;

- *maintenance.* The DSM solution typically remains fixed; it needs to be upgraded when requirements change or when the organization finds a better way to use the DSM solution itself. Typical examples are the generation of code for other output as metrics or simulation tools. The developers of the DSM solution update and share languages, generators and domain frameworks to modelers as long as they do not remain static. The DSM solution reaches the end of its development cycle when applications are no longer developed but there is only bugs correction, or when there is a new target environment and the applications require radical changes that go beyond the original DSM solution.

In literature other techniques supporting DSM development are addressed: two of them are language inheritance and formalism hierarchies. In [38] a very deep analysis of inheritance mechanisms has been done with particular reference to programming languages while in [39] the focus shifts to models and metamodels. In [40] authors define several composition operators for metamodels and in [41] a focus on embedded systems is done: here a series of techniques in DSMLs families creation have been addressed such as importation or extension.

2.4 UML and UML Profiles

The Unified Modeling Language (UML) [42] is a well known general purpose standardized modelling language for software system specification. The system structure is typically specified by a Component Diagram and/or a Class Diagram. The behavioural view of the system is instead specified using Use Cases, Activity Diagrams, Sequence

Diagrams and State Machines, or a combination of them. UML is a semi-formal specification language: the semantics of UML diagrams is expressed in natural language while the abstract syntax is provided in terms of UML meta-models. A UML meta-model is actually a Class Diagram that represents the UML concepts and their relationships as meta-classes and meta-associations, respectively. Constraints on the meta-classes and meta-associations are expressed with OCL [43].

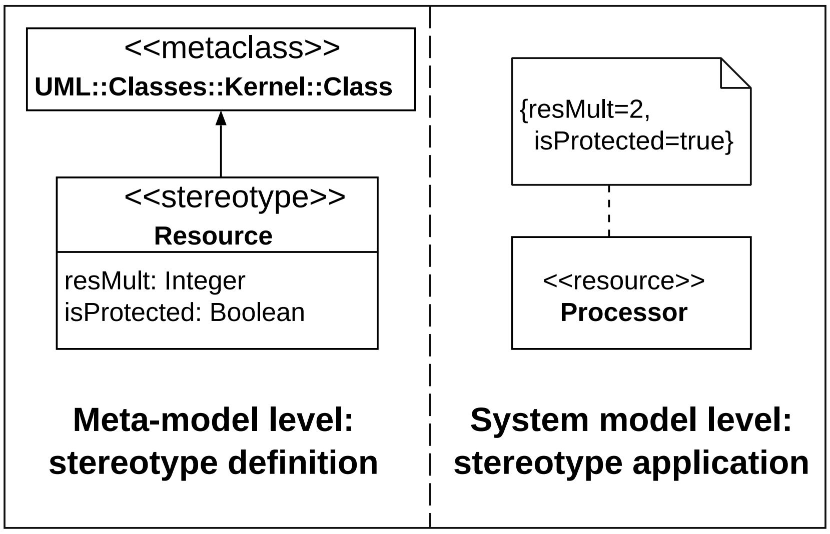
UML can be extended through the *profiling* mechanism that allows to customize UML for a particular domain or platform; this mechanism is defined in the UML Infrastructure. The UML profiling is actually a *lightweight* meta-modelling technique to extend UML, since the standard semantics of UML model elements can be refined in a strictly additive manner. Since this extension mechanism is part of the standard UML, it can be supported by the tool for UML. This feature is one of the main advantages of UML profiles compared to the other mechanisms of customization of UML that are not part of the standard UML and therefore are not supported by the UML tool [44]. Another important advantage of the profiling mechanism UML is that it avoids to redefine concepts already defined in UML. A UML profile is represented by a UML package stereotyped by tag "profile". There are three extension mechanisms used to define a UML profile: stereotypes, tagged value and OCL rules:

- the *stereotype* is the main construct for the specification of a UML profile. It is a particular type of UML class (actually it is a specialization of the *class* metaclass from the UML metamodel). Semantics and notation of a stereotype are therefore very similar to those of a UML class. Stereotypes are identified by a unique name, and represent a set of extensions that are applied to the classes of the extended metamodel. The extended metaclasses are identified by means of extension relations which connect the stereotypes to that extend metaclasses. The steretypes can be seen as semantic specialization, to define the extensions made by the stereotypes tagged value and OCL rules are used;

- a *tagged value* is a property (specialization of the *property* metaclass) that belongs to a stereotype. A tagged value is a new property that is added to the extended metaclass from the stereotype that owns the tagged value. According to the latest UML specification, the type of tagged value can be specified from other classes or stereotypes. So the tagged value can be used to define new attributes, but also for the definition of new associations between classes in the extended metamodel.;

- OCL rules are defined by the Objec Constraint Language [43]. Each OCL rule is applied to a specific stereotype and it is used to control the interaction between different conceptual constructs (extended metaclasses). Although the name OCL refers only to the constraint definition, the latest OCL specifications stipulate that it can also be used as a query language and as language for functions and operations specification.

For example, in Figure 2.3, the *Resource* stereotype extends the *Class* meta-class, then the former can be applied to a class in a Class Diagram. In this example *resMult* and *isProtected* are tags.



<<metaclass>>
UML::Classes::Kernel::Class
<<stereotype>>
Resource
resMult: Integer
isProtected: Boolean
Meta-model level: stereotype definition
{resMult=2, isProtected=true}
<<resource>>
Processor
System model level: stereotype application

FIGURE 2.3: UML stereotypes, tagged values and their application

UML 2 has introduced significant changes to profiles, eliminating the ambiguities (intentionally or not) that have introduced in previous versions, to avoid the restriction of developers [45]. Through these changes and clarifications, the extension mechanisms through the profiles have reached a better definition for the benefit of a uniform usability. Among them, the most important new features include:

- availability of a more extensive and precise definition of profiles and stereotypes. Hence a stereotype in UML 2 is semantically very close to the concept of metaclass; several ambiguity in the original definition has been specified;
- introduction of formal rules for writing OCL constraints linked to stereotypes;
- adding a new notation for stereotypes that is more scalable than the previous one;
- extension and clarification of the semantics of the application and removal of profiles to UML models;
- defining rules for the profile and contents representation through XMI formalism;

- during the definition of a profile it is possible that it is based only on a subset of UML metamodel (rather than the complete metamodel), resulting in a DSML much more compact and simple;
- ability to create new associations between stereotypes and other elements of meta-models. This allows stereotypes to have associations that do not exist for their reference metaclasses;
- introduction of a default mechanism that avoids modelers from stereotyping each element modeling;
- generalization of the profiles mechanism as well as the UML context so that it can be used with any modeling language that is based on MOF.

2.4.1 A structured approach to the definition of UML Profiles

Although the number of available profiles is increasing, in literature, there are a discreet number of articles that guide the developer in the production of UML profiles. The profile realization is a difficult activity, without following a structured design process, the result may be inaccurate and can cause uncertainty in model analysis and the transformation processes. Among these we refer with greater interest to [45, 46]. The definition of a UML profile involves the construction of two separate but related artifacts: a domain model (or meta-model) and the profile itself. A deep knowledge of the domain and a strong familiarity with the UML meta-model are of vital importance for the definition of a profile.

[45] identifies a process which begins with the definition of a domain model which is then translated into a profile. Sometimes it may be necessary iteration between the domain model and profile due to the high complexity of the model or because it is far enough from the UML specification. The main purpose of a domain model is to specify what shall be represented in the DSML and how. Experiences in the definition of profiles indicate that it is better if the first domain model defines just the concepts that are closely to the application domain itself, without taking care of adaptations to the UML, which will be mapping at a later stage. This allows for proper separation of concerns and ensures that the design of the original DSML is not contaminated by contingencies relating to the mapping of the profile. A meta-model is a specification of DSML cleaned from the information that the user profile can provide. In practice it may sometimes not be possible to map precisely a model to its UML meta-model as the latter may have some constraints, attributes, or relationships that are in conflict with the domain model. When this happens, you may want to adjust the domain model with a small

loss of expressive power. Of course, if the profile is far from UML meta-model, it may be that the approach based on profiles is not the correct one for that particular DSML and therefore can be advisable to follow other paths such as the definition of a DSML from scratch. The domain model is a meta-model of the DSML which should include the following key elements:

- a set of basic language constructs that represent the essential concepts of the domain. For example a DSML used to model a multi-tasking operating system should contain linguistic concepts as task, processor, scheduler, queues of processes, priority, and so on;
- a set of valid relationships that exist between the aforementioned domain concepts. In the operating system example, a scheduler may have multiple queues of processes, each with its own priority level;
- a set of constraints which determine how the language constructs can be combined to produce valid models. For example, a constraint may specify that at most one process at a time can be running on each processor. The combination of relationships and constraints makes the fairness rules (well-formedness) of a DSML. Together with language constructs they represent the abstract syntax of the DSML;
- a concrete syntax or notation for the DSML. This is the graphic and/or textual abstract notation. Note that it need not necessarily to be derived from formalisms similar to UML
- the semantics that is the meaning of the language. This is instead a mapping between the elements of the abstract syntax and the related domain entities that syntactic elements represent;

The abstract syntax of a DSML is usually expressed using the OMG MOF language. In principle it is possible to use other languages for meta-modeling, but MOF has the certain advantage of being easily translated into UML profiles. The language that is typically adopted to express domain constraints is OCL because it was specifically designed to be used with MOF and because it is supported by many tools that deal with UML. Once the domain model is complete, the mapping process to the UML meta-model can start. This can be done through a meticulous analysis of all domain concepts and a precise identification of the correct UML constructs that are best suited to represent them. Generally we follow some guidelines:

1. select a UML meta-class whose semantics are as close as possible to the semantics of the domain concept to which it refers. This is very important since the semantics

of UML concepts is often constructed through tool for UML and also because the UML modelers expect a stereotype inherits the semantics of its origin meta-class. It is also to remember that a mapping purely on the syntactic level can lead to confusion or misinterpretation from the tool;

2. check that all the constraints that are applied to the source meta-class are not in contrast one with each other. Note that it may not be sufficient to check only the origin meta-class, but also super-classes must be checked, where present;
3. check if any of the meta-class attributes needs to be refined. This is a way to add to basic concepts the specific semantics coming from the application domain;
4. check if the specific meta-class has associations that cause conflicts with other meta-classes. These could be, for example, the case of conceptual associations that are inherited from UML that can contradict the specific semantics of the domain.

[46] retrace the same process and defines four phases, depicted in Figure 2.4, that are:

- rough description of the domain. It identifies the domain entities, the relationships that exist between them, the rules that connect the structure and behavior of those entities. The result of this step is a conceptual model domain, often expressed with a UML class diagram.
- a reduction of the problem space. The model defined in the previous step is refined and only the useful concepts to the solution are retained. The result of this step is a complete conceptual model of solution.
- a first version of the profile, the "Profile Skeleton". Each entity identified in the previous step is transformed into a stereotype that extends one or more UML meta-class. Tagged-values and constraints are defined and initialized, the latter using OCL.
- analyze the consistency of the produced profile and to perform a profile optimization. This optimization is actuated reviewing the association and reducing the number of stereotypes (e.g mapping some concepts to complex datatypes).

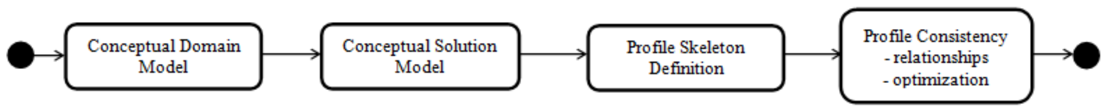


Figure 2.4: Profile definition process

2.5 Model transformations

The transformation is a key concept in the MDE context. They allow to obtain a model automatically from another, for example they are useful to change the formalism and to generate a formal model starting from a UML model. The standard of OMG, in his paper [34], specifies the need for change to move from platform-independent models to platform-specific models, raising the level of abstraction during the modeling phase, then reducing it for a specific platforms, during the development stages.

The transformations can be grouped into two categories:

- Model-to-Model (M2M) transformations;
- Model-to-Text (M2T) transformations.

M2M transformations aim at transforming the model in other model, expressed for example in a different formalism. The main motivation of their need is that the new model enables to perform analysis that are not feasible in the previous formalism. In the approach proposed in thesis these transformations will be widely adopted.

M2T transformations are typically performed by queries. The query mechanism allows users to obtain from the model some textual informations. The needs for M2T transformations are identified in the need to produce, for example, automatic reports extracting relevant informations from the models, or to provide structured data to perform the processing with other software tools. This thesis will use M2T transformations in order to produce textual input files for the modeling tools needed for the analysis.

An example of language used to write M2M and M2T transformation is the ATLAS Transformation Language (ATL) defined in the ATLAS Model Management Architecture (AMMA) [47] platform. ATL is a hybrid language, i.e. it is both a declarative language and imperative one. ATL is applied in a pattern of transformation shown in Figure 2.5.

In the pattern depicted by the figure, a source model *Ma* is transformed into a target model *Mb*, according to the rules defined in the transformation *mma2mmb.atl*, written in ATL. The transformation can be seen as model since it is software. Source and target models, as well as the transformation definition are conform to their respective meta-models: *MMa*, *MMB* and *ATL*. All meta-models in this example are conform to MOF meta-meta-model (obviously this relationship is not strictly necessary, other meta-meta-models are of course usable). This schema, even though it is designed for ATL, it is generally enough to be adopted by all other transformation languages.

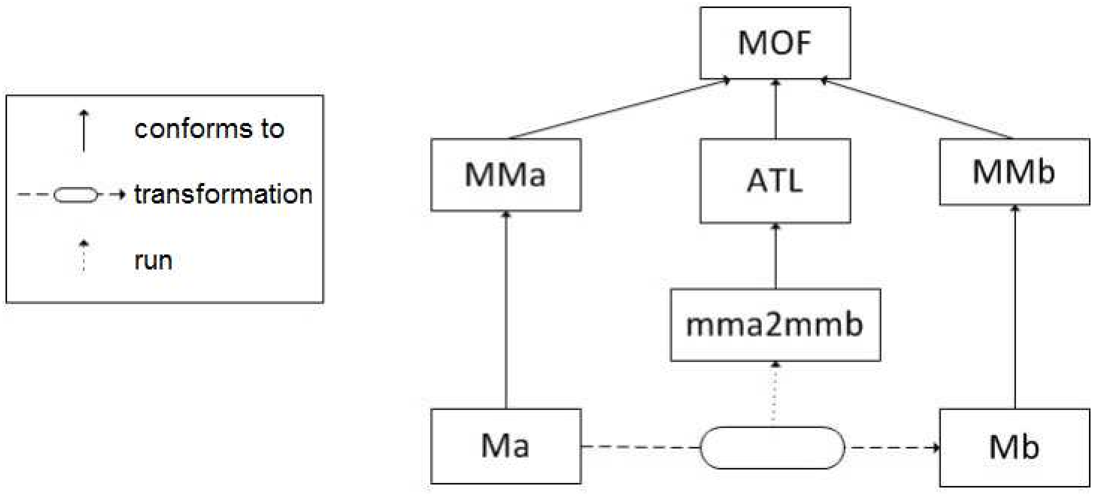


FIGURE 2.5: ATL transformation schema

ATL, as mentioned before, is a mixed language which contains declarative and imperative parts, nevertheless the ATL philosophy encourages the use of the declarative style in specifying transformations. However, sometimes it is difficult to provide a solution completely declarative in a transformation problem. In this case it is possible to use characteristics of the imperative language. ATL transformations are unidirectional, source models are read-only and the transformations produce write-only destination models. The implementation of a bidirectional transformation makes necessary to realize a pair of transformations, one for each direction.

2.6 Formal Model-Based and Model-Driven

Both in formal Model-Based and in Model-Driven, models play a fundamental role in the creation and management of a system. Model-Based is the formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and, later, life cycle phases. With respect to evaluation of critical systems, formal Model-Based uses a model of a proposed system to obtain prediction about usability measures, by calculation or simulation. In addition, the content of the model itself conveys useful information about the relationship between user's task and system design. The goal of formal Model-Based evaluation is to get some results before the implementation of the system or of its prototype. The model itself summarizes the design and the characteristic of the system that will be implemented.

The Model-Based has a wider application range than the Model-Driven; the first narrow to the latter when the concept of transformations between models is used. The Model-Driven approach consists in starting from a certain type of initial model and generate from it, in an automatic manner, other models which describe the system from various points of view. In the evaluation of critical systems, transformations between models can help in carrying out particular types of analysis. Starting, for example, from a structural model of the system which shows its main components, and considering their interactions to realize a functionality, it is possible to obtain by means of a transformation documentation of the system itself or even a useful model for the analysis of system failures (e.g a Fault Tree).

Chapter 3

Languages for modeling critical systems: MARTE and DAM

Recently, the literature has proposed several languages oriented to the modeling of critical systems, in particular focusing on their non-functional properties. In this context, a central role is played by the UML language since it is a modeling de-facto standard and, secondly, since it is the language that better fit the definition of new languages through the already described profiles mechanism (Section 2.4). As mentioned above, in order to address the need to represent requirements and all the information for quantitative analysis, it is therefore necessary to increase the expressive power of UML by introducing new notations.

The type of the specific domain of analysis determines the particular informations to represent with stereotypes. Depending on this, schedulability quantitative analysis may be carried out, rather than the performance or dependability. As an example, in the case of performance analysis, it is necessary to describe *resources* and *services* (operation offered by the resources) and to characterize them with those informations such as the usage rate of a resource, the response time and the throughput of a service. Obviously those information can belong to two types: quantitative, when they describe quantities (such as energy, size or time) that can be measured (through enumeration, comparison or in an absolute way) and expressed using numerical values, and qualitative, when these features can not be measured numerically. Sometimes, in the firsts, numerical value has a significance if they are also specified whit the measurement unit, while the latter can be described by their typology (e.g. a frequency can be qualitatively characterized as sporadic or periodic).

Different profiles presented in literature address information necessary for quantitative evaluation of critical systems. Between them, those that have been widely analyzed are those regarding the security and the protection of critical infrastructures and those aimed to the performance and dependability of critical systems. The main profile in those areas, is MARTE (Modeling and Analysis of Real-Time Embedded System) [48]: this is a OMG standard profile created as the evolution of the SPT profile, dedicated to the Scheduling, Performance and Time. [18] presents the Dependability Analysis and Modeling profile for the analysis of quantitative dependability of software systems modeled in UML; this profile inherit some concepts of the MARTE profile in fact the name of the profile is MARTE-DAM (or simply DAM). [49] shows a UML profile (called UML-CI) for the management of critical infrastructures; the management of a urban wireless network infrastructure is presented as case study. In [50] a DSML is created for the purpose of modeling for risk analysis and the tools that adopt this language are presented. There are also mechanisms for the consistency control of likelihood.

Since MARTE and DAM are the basis of the new languages proposed in this thesis, in the following sections additional details about them will be added.

3.1 MARTE

The UML Profile for *Modeling and Analysis of Real-Time and Embedded systems (MARTE)* [48] is an OMG-standard profile that customizes UML for the modelling and analysis of Real Time and Embedded Systems; it is able to capture non-functional properties (NFP) of a system under analysis, such as timing or performance-related properties; it is useful in all the development cycle, from specification to verification and validation phase. MARTE adopts a component-based paradigms since component architectures are increasingly used in RTE execution platforms and this paradigm is applicable to different stages of the project. Component concept is also used to structure both system and software engineering processes.

The MARTE domain model is reported in Figure 3.1: there are three packages which describe all the concepts required for modelling and analysis of real time and embedded systems. In particular the *MARTE Foundation* contain all the concepts needed to model a generic resource or a generic component. This package includes the main concepts of non-functional properties modelling. The *MARTE Design Model* package specializes the first for modelling purpose, adding a distinction between the modelling of hardware and software resources. The *MARTE Analysis Model* adds those concept useful for analysis purpose, such as schedulability and performance analysis, but allows also for a generic quantitative analysis of any other property.

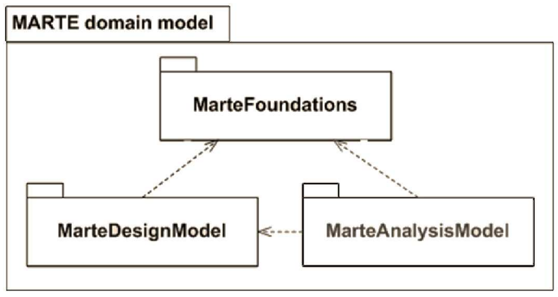


FIGURE 3.1: MARTE domain model

The MARTE Profile has been obtained extending UML with new modelling capabilities (Figure 3.2). It is divided into four packages: three of them corresponds to the three packages defined in the domain model while the fourth, *MARTE annexes*, provides the specification language and the library of the MARTE model.

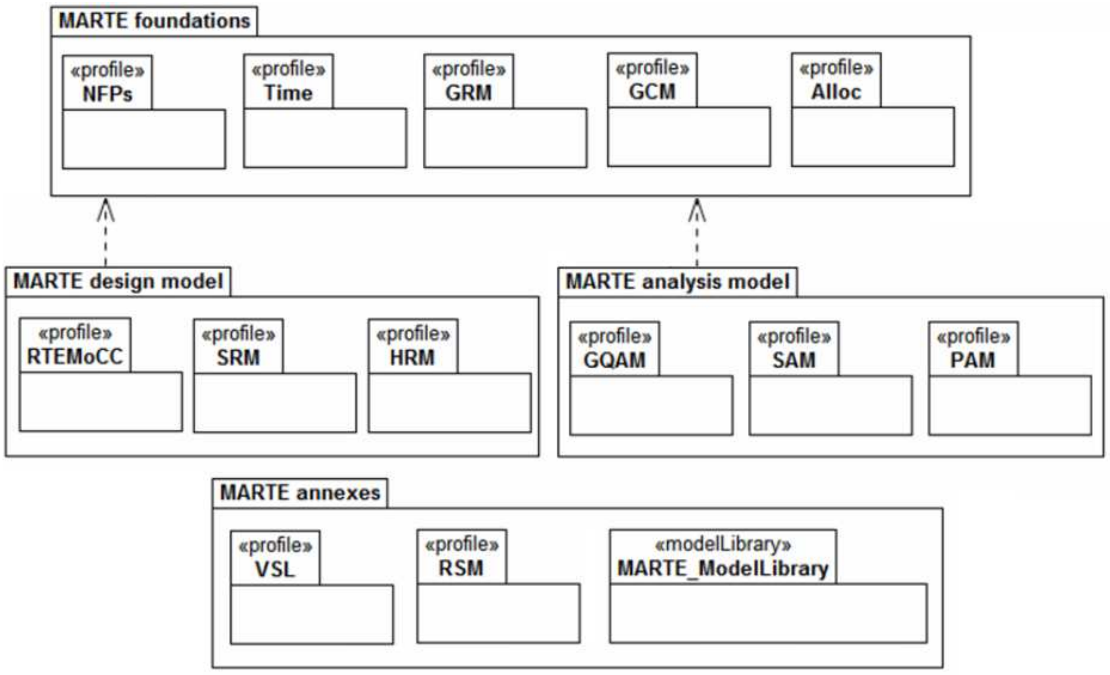


FIGURE 3.2: MARTE profile

The MARTE profile consists of several sub-profiles: in particular, the Time package is very important for real-timing modelling purpose. MARTE defines three basic models of time: Chronometric, Logical and Synchronous. The first concerns mainly the absolute time (for example the delay, the duration and the time clock), the second ordered events (for example, it defines that the event 1 occurs before the event 2) and the third introduces the concept of simultaneity (for example, it defines that the first event to happen at the same time of the event 2).

Another important package is the GRM (General Resource Modeling) that provides the basic abstractions for high-level modeling through generic concepts of real-time applications: resources, services, etc.

The Alloc (Allocation) package is useful to model the allocation of the application elements on the available resources. In MARTE the allocation concept is the association between an application and execution platform (both defined in MARTE). The application elements can be UML elements that are used to model an application, both structural and behavioral aspects. An execution platform instead is represented as a set of connected resources where each resource provides services to support the execution of the application. So resources are basic structural elements, while the services are behavioral elements.

Specifically for the analysis aim, MARTE contains a general package called General Quantitative Analysis Model (GQAM): the object of this package is to represent some general analysis concepts and describe how the system behaviour uses resources; although different domains have different terminology, concepts, and semantic, they also share some foundational concepts which are expressed in GQAM. A specific analysis domain can be defined through specialization of GQAM; concretely, MARTE addresses the schedulability analysis and the performance analysis by defining two separated subprofiles specializing the GQAM package.

A key feature of MARTE is the NFP framework and its Value Specification Language (VSL). The former provides a general framework to annotate UML models with non functional properties, defining the necessary data-types for the definition of a specific analysis domain. In particular, a data-type is characterized by several properties, such as the origin (which allows the modeler to specify whether a non-functional property is a requirement or a metric to be estimated), the type of statistical measure associated to an NFP (e.g., mean), the type of the quality order relation in the value domain of an NFP, for comparative analysis purposes. Instead the VSL has been defined in MARTE in order to specify expressions for constraints, properties, and stereotype attributes. In fact, this expression language enables the values specification, at model level, in tagged values, body of constraints, and in any UML element according to a well-defined syntax. The VSL allows to express values using a scientific notation, to specify the upper and lower bounds of an interval, to declare variables, etc.

3.2 DAM

The *Dependability Analysis and Modeling* [18] (DAM) profile was created with the aim of providing a standard support for the dependability analysis of software systems: this profile is a specialization of general concepts for quantitative analysis provided by MARTE-GQAM. The DAM profile is presented with an accurate level of detail for two reasons: the first is that the definition of this profile follows what explained in the previous Chapter and, secondly, since this profile will be extended with new concepts in the following Chapters.

DAM is conceived by following the systematic approach already described: first, a dependability domain model was built; later, the domain model is mapped onto UML extensions (i.e., stereotypes, tags and OCL constraints).

In the initial phase a very accurate study of the dependability concepts and of existing profiles have been carried out. The result of this phase was a list of requirements to satisfy in the profile DAM. These requirements include the identification of the dependability context, of the upper and lower bounds for dependability requirements, of the metrics to estimate and properties to verify. In addition, two fundamental aspects are taken into account: faults and their relationships, and for repairable systems, the repair/recovery processes. Finally, the last requirement provides for the description of the redundant structures which can be present in the system. In the following phase the DAM domain model has been realized, this diagram describes the basic concepts inserted in DAM. After the complete assessment of the domain model, its concepts have been mapped the UML extensions were a set of stereotypes that extend UML language constructs and a specific library for the dependability analysis have been defined. The last phase involve the assessment of the DAM profile.

3.2.1 The DAM domain model

The DAM domain model, that is the class diagram representing concepts that are the basis of the dependability analysis, is structured into packages (Figure 3.3). The system modelling view adopted by DAM is the component-based approach, similarly to MARTE, from which DAM depends.

The main package is the *System* which describes the model under analysis; this package is decomposed in turn in two sub-packages: the *Core* and the *Redundancy* packages which describe respectively the system and, where present, the internal structure of fault tolerant components. The *Threats* model add threats to the system. The *Maintenance* package includes also maintenance concepts.

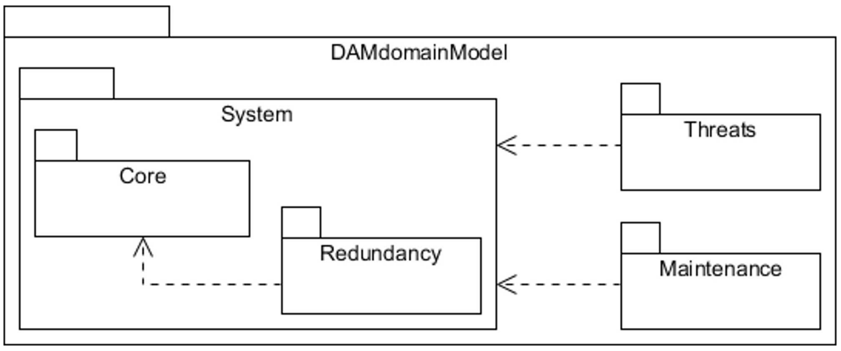


FIGURE 3.3: DAM domain model

The *Core* model is reported in Figure 3.4: it introduces the general concepts that allow to fully describe a reliable system from the structural and behavioral points of view. Each system is considered as a set of *components* (hardware or software): each component can be composed of other components, and different components are bounded together by *connectors* in order to interact. The system delivers a set of high-level *services*. Such services, which may be required from any component, are provided by one or more components of the system. A component provides or requires at least one service (as indicated by the constraint). A service can be composed as the aggregation of basic services and can be also detailed as a sequence of steps. The core domain model depends from two packages defined in MARTE and, specifically, from the General Resource Modeling (GRM) and from the General Quantitative Analysis and Modeling (GQAM). In fact some conceptual classes specialize some conceptual classes defined in MARTE. Some elements of the core model have attributes that represent requirements, metrics, properties or input parameters of a dependability analysis.

The redundancy package contains a domain model to represents redundant structure as a set of fault tolerant components. A hardware/software *redundant structure*, shown in Figure 3.5, increases the system fault tolerance and it is made of components, named generically *FT components*. In particular, a fault tolerant component can be a *spare*, a *variant*, that provides the same services of a main component but with a different design, a *controller*, responsible for the co-ordination of the variants, or an *adjudicator*, that manage the consensus of two or more outputs among the N-variants. .

The system can be affected by threats, i.e. *faults, errors, impairments* according to the definition given in [51]. A *fault* is the original cause of *errors* and it affects system *components*. *Errors* are related to steps and they can be propagated from the faulty component to other components it interacts with. *Errors* may cause *impairments* (*hazards* affecting safety, *failures* otherwise) at different levels: at step level, when the service provided by the component becomes incorrect; at component level, when the component is unable to provide service; at service level, when the failure is perceived by external

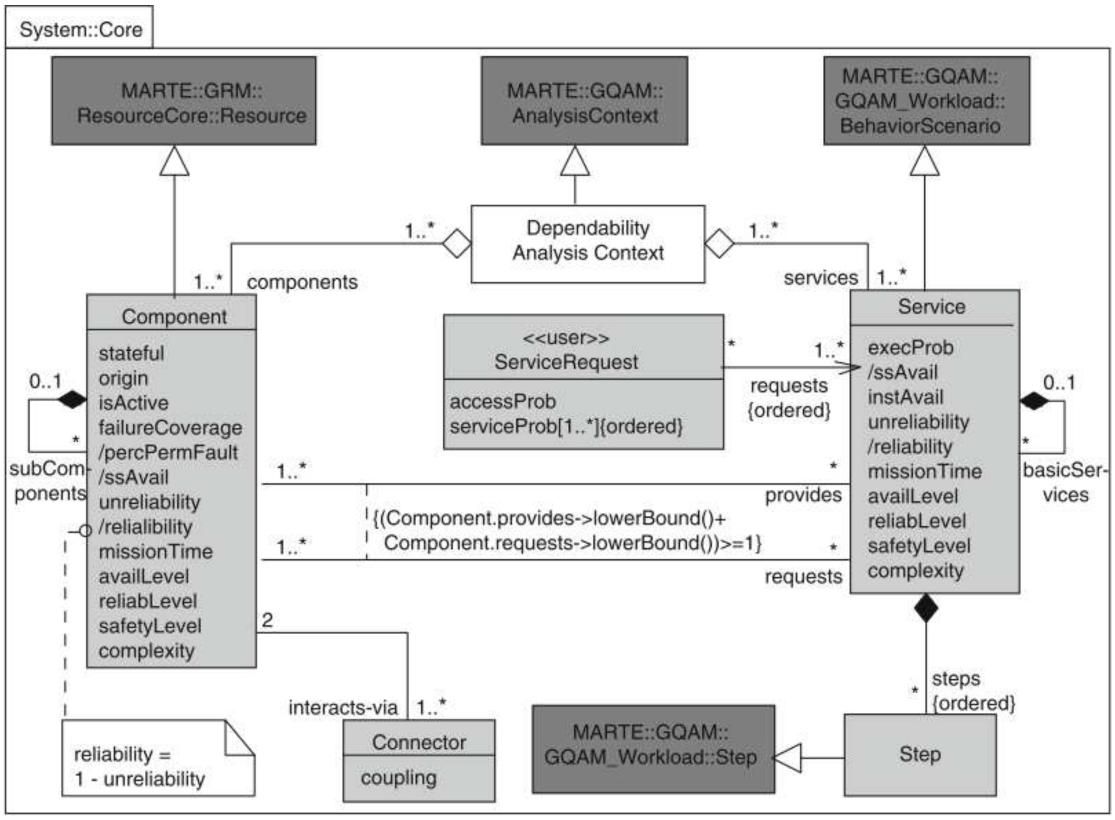


FIGURE 3.4: System::Core domain model

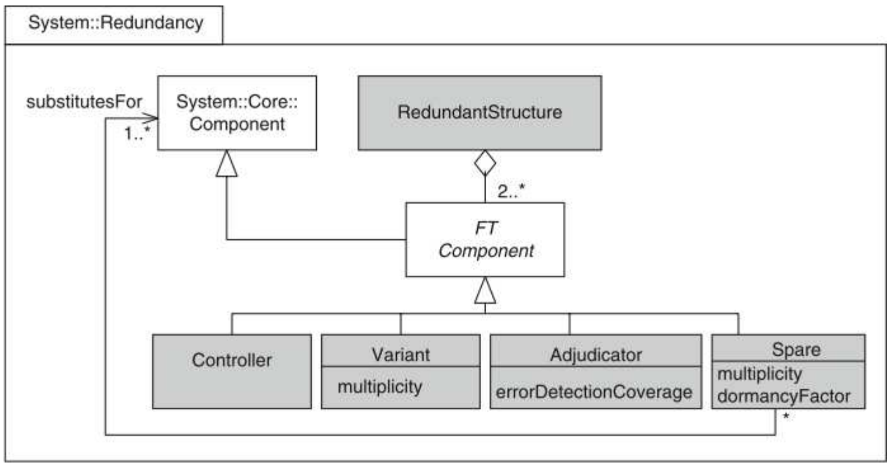


FIGURE 3.5: System::Redundancy domain model

users. The Threats model is shown in Figure 3.6. In this model there is also a *fault generator*, useful when you want to model the fault injectors.

The Maintenance domain model (Figure 3.7) includes concepts that are necessary to support the evaluation of system dependability. Given that during the servicing operation the system is not available or is only partially available to users, the time required to carry out such operations has a strong impact on the system availability. In particular the *maintenance actions* restore the system affected by threats. According to [51],

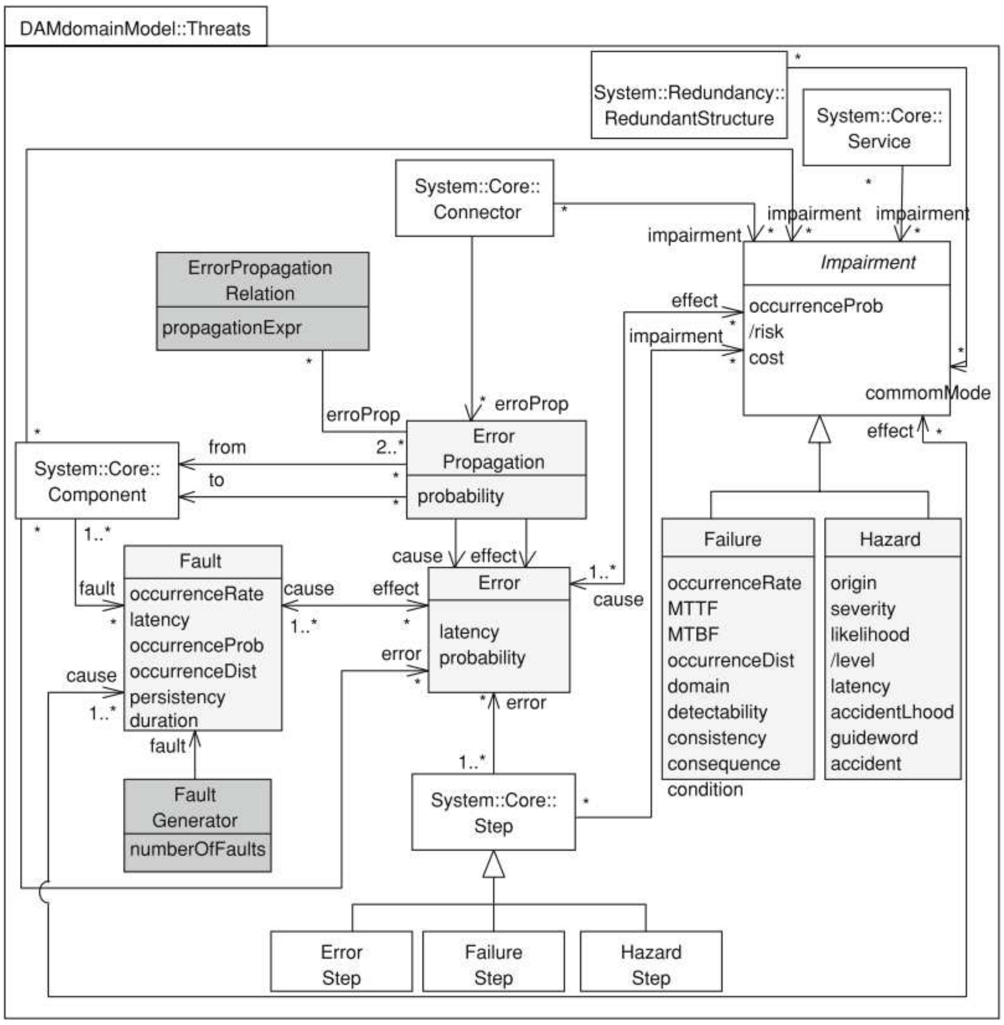


FIGURE 3.6: Threats domain model

repairs of system components involve the participation of *external agents* (e.g., repairman, test equipment, etc.), while *recovery* of services does not require the intervention of the latter. The model also includes the *replacement steps* of components and spares and the *reallocation steps* of services on spares.

3.2.2 The DAM Profile

The DAM profile has been defined by mapping, systematically, the concepts of the DAM domain model onto UML extensions and onto MARTE concepts. It includes a set of DAM extensions (stereotypes and tags) and a DAM library containing dependability specific types. To maintain a small (and usable) set of stereotypes, only a subset of conceptual classes of the domain model were mapped to stereotypes but, when it had been possible, complex datatypes have been defined. The DAM profile is formed by two packages (Figure 3.8 (a)): the *DAM_UML_Extensions* and the *DAM_Library*. The

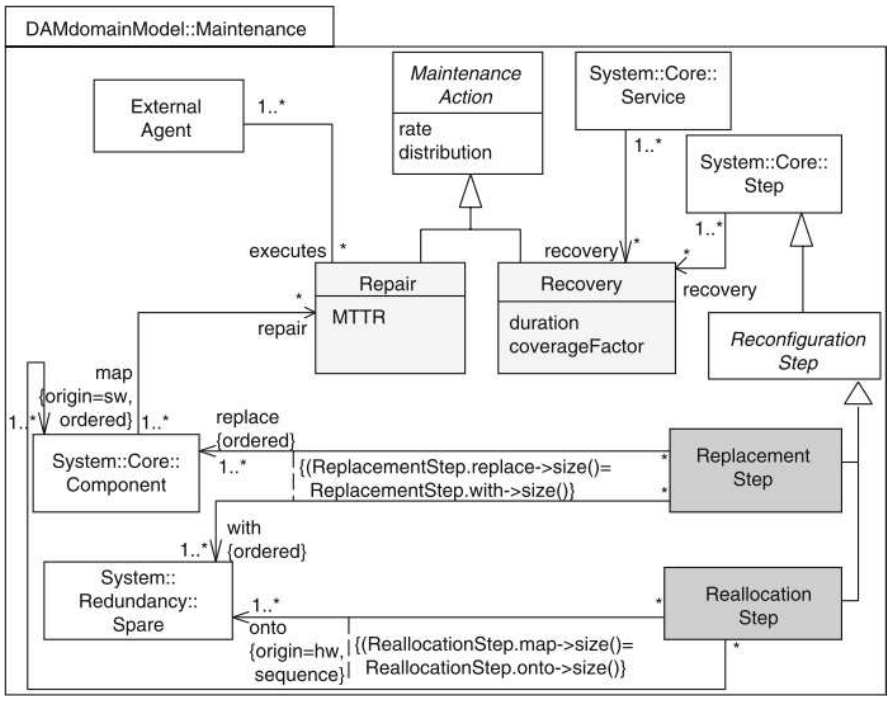


FIGURE 3.7: Maintenance domain model

first contains all the stereotypes, obtained as UML extensions and MARTE inheritances, defined by DAM; the latter is the model library in which datatypes are defined. The *DAM_Library* is structured in turn as two packages(Figure 3.8 (b)): the *Basic_DA_Types* and the *Complex_DA_Types*.

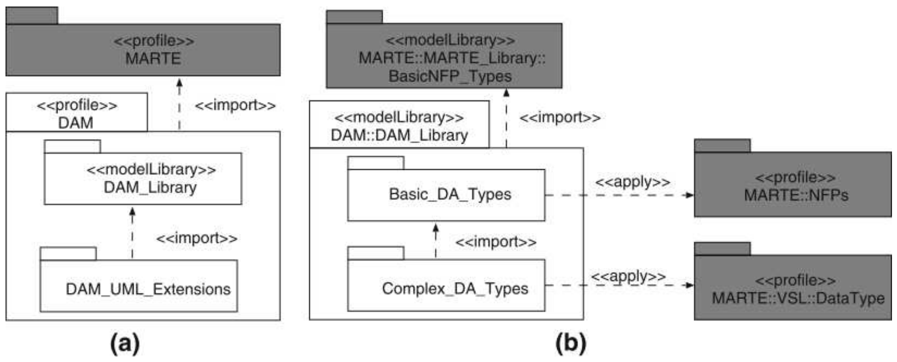


FIGURE 3.8: DAM profile

To maintain the traceability of the DAM stereotypes and datatypes, it was decided to use the same name of the conceptual classes defined in the DAM domain model, adding the prefix *Da*. As an example the Figure 3.9 shows the set of stereotypes obtained by mapping conceptual classes belonging to the Core package of the DAM domain model. In the figure, five different stereotypes are showed: *DaComponent*, *DaService*, *DaStep*, *DaConnector* and *DaServiceRequest*. In particular, while the first three inherit from MARTE stereotypes, the last two extend UML metaclasses since the concepts they represent are not present in MARTE. Each stereotype defines its own tagged-values and, where present, inherits those defined in the MARTE stereotype from which it inherits. The same approach has been used to define other stereotypes belonging to other packages.

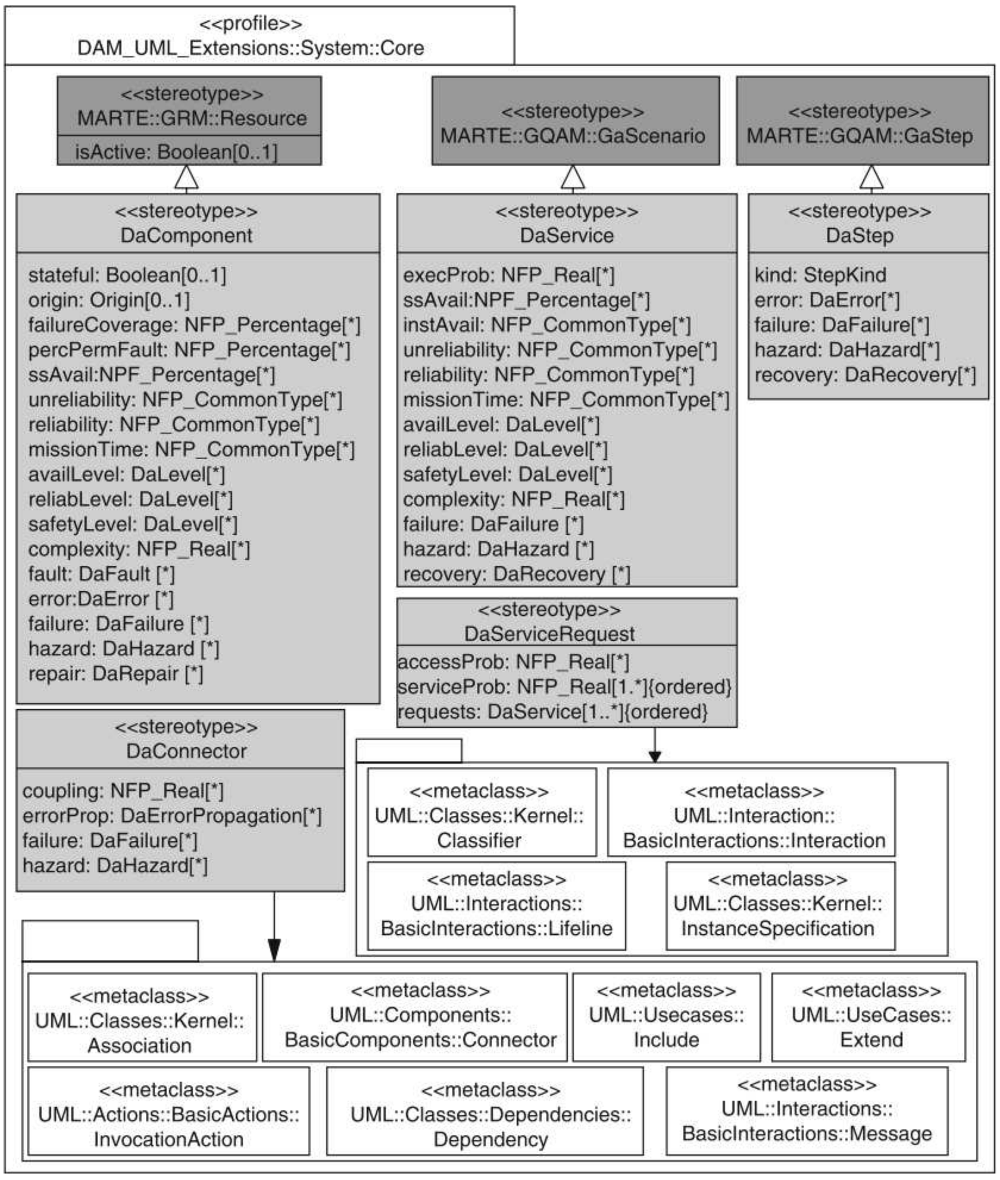


FIGURE 3.9: Core package of the DAM profile

The *DAM_Library* imports NFPs and reuse the VSL defined in the MARTE profile. The complex datatypes represent those conceptual classes that have been not mapped to stereotypes: these datatypes come from the Threat and Maintenance packages of the DAM Domain Model. Example of complex datatypes are the *DaFault*, *DaError* and the *DaFailure* that map respectively the Fault, Error and Failure conceptual classes. Each datatype shows, as properties, the attributes of the conceptual classes. On the other hand, the basic dataypes are simply enumerative types (defining frequencies, failure detectability, etc.) or valued types (Figure 3.10)

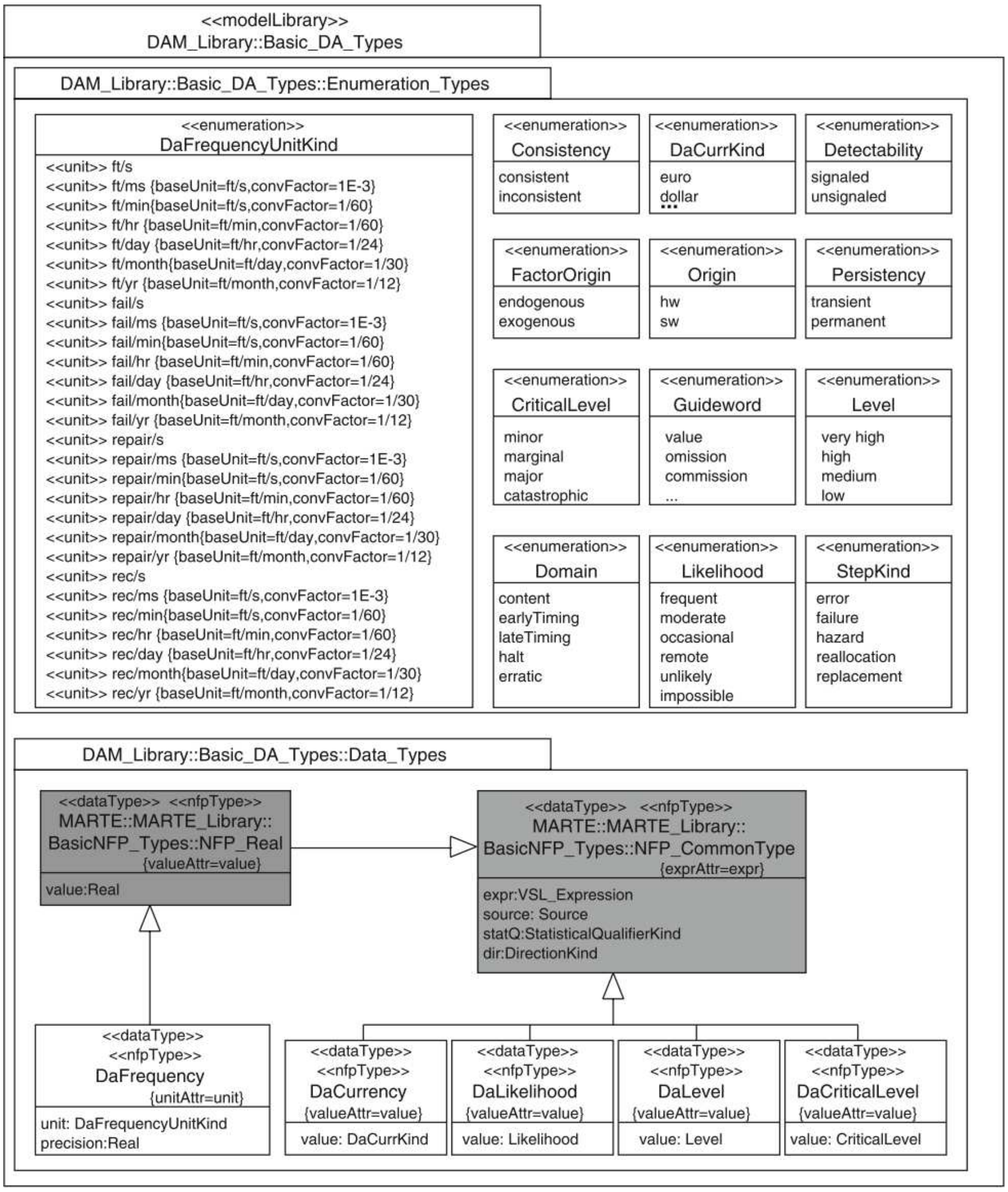

<<modelLibrary>>
DAM_Library::Basic_DA_Types
DAM_Library::Basic_DA_Types::Enumeration_Types
<<enumeration>>
DaFrequencyUnitKind
<<unit>> ft/s
<<unit>> ft/ms {baseUnit=ft/s,convFactor=1E-3}
<<unit>> ft/min{baseUnit=ft/s,convFactor=1/60}
<<unit>> ft/hr {baseUnit=ft/min,convFactor=1/60}
<<unit>> ft/day {baseUnit=ft/hr,convFactor=1/24}
<<unit>> ft/month{baseUnit=ft/day,convFactor=1/30}
<<unit>> ft/yr {baseUnit=ft/month,convFactor=1/12}
<<unit>> fail/s
<<unit>> fail/ms {baseUnit=ft/s,convFactor=1E-3}
<<unit>> fail/min{baseUnit=ft/s,convFactor=1/60}
<<unit>> fail/hr {baseUnit=ft/min,convFactor=1/60}
<<unit>> fail/day {baseUnit=ft/hr,convFactor=1/24}
<<unit>> fail/month{baseUnit=ft/day,convFactor=1/30}
<<unit>> fail/yr {baseUnit=ft/month,convFactor=1/12}
<<unit>> repair/s
<<unit>> repair/ms {baseUnit=ft/s,convFactor=1E-3}
<<unit>> repair/min{baseUnit=ft/s,convFactor=1/60}
<<unit>> repair/hr {baseUnit=ft/min,convFactor=1/60}
<<unit>> repair/day {baseUnit=ft/hr,convFactor=1/24}
<<unit>> repair/month{baseUnit=ft/day,convFactor=1/30}
<<unit>> repair/yr {baseUnit=ft/month,convFactor=1/12}
<<unit>> rec/s
<<unit>> rec/ms {baseUnit=ft/s,convFactor=1E-3}
<<unit>> rec/min{baseUnit=ft/s,convFactor=1/60}
<<unit>> rec/hr {baseUnit=ft/min,convFactor=1/60}
<<unit>> rec/day {baseUnit=ft/hr,convFactor=1/24}
<<unit>> rec/month{baseUnit=ft/day,convFactor=1/30}
<<unit>> rec/yr {baseUnit=ft/month,convFactor=1/12}
<<enumeration>> Consistency
consistent
inconsistent
<<enumeration>> DaCurrKind
euro
dollar
...
<<enumeration>> Detectability
signaled
unsignaled
<<enumeration>> FactorOrigin
endogenous
exogenous
<<enumeration>> Origin
hw
sw
<<enumeration>> Persistency
transient
permanent
<<enumeration>> CriticalLevel
minor
marginal
major
catastrophic
<<enumeration>> Guideword
value
omission
commission
...
<<enumeration>> Level
very high
high
medium
low
<<enumeration>> Domain
content
earlyTiming
lateTiming
halt
erratic
<<enumeration>> Likelihood
frequent
moderate
occasional
remote
unlikely
impossible
<<enumeration>> StepKind
error
failure
hazard
reallocation
replacement
DAM_Library::Basic_DA_Types::Data_Types
<<dataType>> <<nfpType>>
MARTE::MARTE_Library::
BasicNFP_Types::NFP_Real
{valueAttr=value}
value:Real
<<dataType>> <<nfpType>>
MARTE::MARTE_Library::
BasicNFP_Types::NFP_CommonType
{exprAttr=expr}
expr:VSL_Expression
source: Source
statQ:StatisticalQualifierKind
dir:DirectionKind
<<dataType>> <<nfpType>> DaFrequency {unitAttr=unit}
unit: DaFrequencyUnitKind
precision:Real
<<dataType>> <<nfpType>> DaCurrency {valueAttr=value}
value: DaCurrKind
<<dataType>> <<nfpType>> DaLikelihood {valueAttr=value}
value: Likelihood
<<dataType>> <<nfpType>> DaLevel {valueAttr=value}
value: Level
<<dataType>> <<nfpType>> DaCriticalLevel {valueAttr=value}
value: CriticalLevel

FIGURE 3.10: Basic datatypes in *DAM_Library*

Part II

An integrated methodology for quantitative analysis of Critical Systems

Chapter 4

A Model-Driven Approach for Quantitative Analysis

This chapter has the objective to describe the steps necessary to move towards a Model-Driven approach for quantitative analysis of critical systems. The proposed approach integrate the formal Model-Based analysis (i.e. quantitative analysis performed with a formal modeling approach) with the benefits given by Model-Driven which support automatic generation of formal models starting from an high-level description of the system. In detail the approach combines the three different techniques described in Section 1.3, in order to exploit the benefits brought by each of them: *divide-and-conquer*, *multi-formalism/multi-paradigm* and *automatic generation*. According to Model-Driven paradigm (Chapter 2), the system under analysis can be described by using high-level specifications expressed in Domain Specific Modeling Languages (DSMLs), automatic transformation chains are able to generate formal models. Note that, also if used sometimes as interchangeable, there is a slight difference between DSL and DSML: the second specifically addresses modeling languages, while the first is a more general concept and can be applied, for example, also to programming languages.

The proposed methodology is implementable into cost-effective processes in order to reduce time-to-market, specifically during the verification stages of development lifecycles. The definition of DSMLs and model transformations can not be a handcrafted discipline, but must be ruled by mechanisms and techniques with the aim of improving reuse. The originality of the proposed approach is in combining the three main research trends, previously listed, in a comprehensive methodology based on Model-Driven principles and that are able to generate single or multi-formalism, and component-based formal models by means of transformational techniques. This approach follows and deepens what described in [52], adding new details and shades, coming from the application of the methodology to real-world systems of different applicative domain, as it will be described in Part III. The main innovation resides in the extension of horizontal languages with vertical one; furthermore new high-level modeling languages have been defined (Chapter 5) as well as new formal operators and techniques (Chapter 6). Next sections show the proposed approach and its application to a first simple application domain.

4.1 Description of the approach

In this Section the proposed approach is described. Figure 4.1 depicts the conceptual schema behind the approach; it is a three stages approach in order to combine heterogeneity, compositionality and automatic generation of formal modeling.

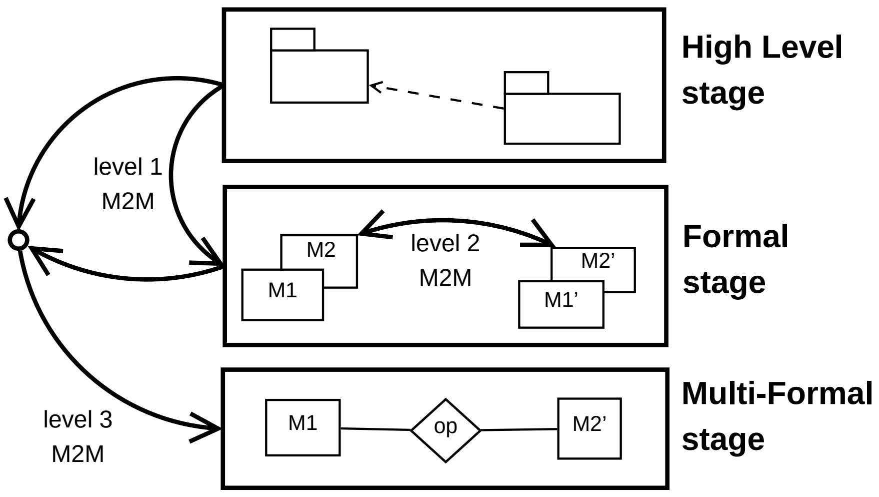


FIGURE 4.1: Conceptual schema of the methodology

The stages are described in a top-down manner. The first stage (*high-level stage*) covers all phases of high-level modeling of the system and its requirements. In order to establish an automated method, it is necessary that all models comply with languages whose syntax has been formally defined. At this stage, the creation of high-level models requires the use of a DSML that can be eventually defined from existing ones and/or using the UML profiling technique. The definition of the DSML requires that language engineers carefully analyze considered domain aspects in order to define and realize a proper metamodel; system modelers shall use the defined DSML and create a model of the system under analysis by specifying peculiar characteristics (properly integrated into the DSML itself).

At the second stage (*formal stage*) several formal models can be automatically generated on the basis of the high-level one: each of the generated models can be expressed in a different formal language. In this context “formal” means that these languages are based on strong mathematical bases (both combinatorial and state-based formalisms).

Examples of formalisms that can be used at this stage range from combinatorial to state-based: Fault Tree, Queuing Networks, Process Algebras, Petri Nets and their extensions. The choice of the language depends on the specific characteristics of the system, and on the type of the requirements whose fulfillment has to be verified through formal modeling.

At the third stage (*multi-formal stage*) single-formalism models are composed in order to create a multi-formal model of the entire system. If quantitative analysis can be performed using a single-formalism model, then the third stage is not necessary. The multi-formal stage is instead necessary for quantitative analysis that require an overall and multi-aspect model of the system, in this case a single-formalism model can not be sufficient to allow the required analysis. The importance of having a multi-formal approach makes possible to have an unique model of the overall system, which expresses its peculiar aspects by choosing the best fitting formalism. This level allows not only a multi-formal description of the system, but also the definition of a model solution process that is able to analyze efficiently each sub-model with the most appropriate solver and aggregates the results in an appropriate manner (as performed by the Os-MoSys framework). The relationships among high-level model subparts are mapped into compositionality operators, which define the rules of passing parameters and/or results between models.

Model-to-model (M2M) transformations allow the generation of models from others. In this approach the following kinds of M2M transformations are defined:

- *Level 1 M2M transformations*: they are used to generate formal models from high-level specifications, allowing to pass from the first to the second modeling stage.
- *Level 2 M2M transformations*: they are used to perform translations between formal models inside the second stage. Such passage is often desirable in order to simplify the final multi-formal model and therefore the solution process (e.g. FTs can be expressed through SANs). It is important to clarify that this transformation passage is optional, not strictly required by the approach.
- *Level 3 M2M transformations*: they generate a multi-formal model by adding compositional operators among the models produced at the second stage. In order to accomplish to this purpose, these transformations have as inputs both formal models and the high-level model developed at the first stage, since some required information may not have been translated into the second stage models.

At the end additional Model-to-Text (M2T) transformations can be used to transform both single and multi-formal models (those belonging to second and third stage) into the specific syntax of the analysis solver: as an example SAN models can be transformed into the concrete syntax of the Möbius tool, i.e. into a set of XML files, written in the concrete syntax understood by the tool.

4.2 Enabling techniques

In order to be feasible and cost-effective, the proposed methodology should be supported by proper techniques which make possible a high-level of automation. In this Section these techniques are addressed.

4.2.1 Extending languages

The first technique here applied is the DSML construction. The DSML must be suitable to describe all the aspects of the system, needed during the following analysis activities. If, for example, an availability analysis is needed, the high-level model must contain all the information regarding the context (e.g. mission time, environmental condition, etc.) as well as components failure rates and their failure modes. Another example could be the performability analysis: in this case the high-level model shall contain not only the information needed by the availability analysis, but also information regarding services performed by the system. In the last case, service information strictly depend on the application domain: the performability analysis of railway systems require the annotation of travel times between stations, dwell times at a station, acceleration and braking characteristics, and each failure must express its effects on the service in term of consequences on the movement capacity and/or caused delay.

For these reasons it is important to highlight that horizontal and vertical DSMLs [53] can be useful but with different characteristic: the horizontal DSMLs can allow the modeling of those concepts that are common to several application domains such as data modeling, testing, performance; they are characterized by high usability and low re-usability; vertical DSMLs focus on specific business contexts (telecommunication, railway, automotive, etc.) and can support the annotation of characteristics of the specific application domain. It is possible to use new language able to join the advantages of both the two kinds of languages by means of language composition techniques: this is obtained starting from a more general horizontal DSML, adding those vertical concepts and features needed during the analysis phase.

In the example described in Section 4.5, in fact, the horizontal DSML DAM (described in Section 3.2) is extended with elements of a vertical domain (the *production cell* domain).

At UML profile level, stereotypes belonging to vertical DSML own the tagged value of the inherited stereotypes of the horizontal DSML, producing a more usable profile for the domain modelers.

4.2.2 Inheriting transformations

In the previous Subsection it has been pointed out that new languages can be obtained specializing horizontal DSML with new vertical concepts. This conduct to problems in the transformation development: new domain specific concepts require the implementation of new transformations, which cannot be reused in other application domains. Hence it is necessary to understand how extension of an horizontal DSML can be exploited in order to achieve a greater level of reuse of model transformations. It is possible to refer to a mechanism that tries to exploit the relationship between a base language and a derived language in order to define relationship between rules in base and derived transformations: the transformation inheritance. Such methodology is fully explained in [54] and can be summarized in Table 4.1 where four cases of *language elements inheritance / transformation rules inheritance* are described. The description refers to the following case: assuming A' as a language extension of A, we want to understand the inheritance relationships between $A - to - B$ and $A' - to - B$ transformations.

TABLE 4.1: Summary of transformation inheritance methodology

Name	**Language Elements**	**Transformation Rules**
addition	A' extends A by adding a new element	a new rule, that maps the new source element onto the target one, must be defined
redefinition (new context)	an element in A is inherited without changes but it may be used within a new context in A'	the rules in $A-to-B$, related to the translation of inherited elements, are extended and/or partially redefined in $A'-to-B$
redefinition (old context)	an element defined by A assumes a new meaning within A', but the context in which it is used does not change	the rules in $A-to-B$, related to the translation of inherited elements, must be redefined
full reuse	an element defined in A is inherited by an element of A' without modifications	the rules in $A-to-B$, related to the translation of inherited elements, are fully reused by $A' - to - B$

The *addition* case requires to add, to an existing transformation, all those rules that maps new stereotypes into target elements. The *redefinition* situations require just to refine or redefine all those rules where source elements muted from their initial meaning.

The *full reuse* does not need implementation of new rules since the new language inherits entirely from the previous one without any meaning mutation.

From a more technical point of view, two main mechanisms have been defined in literature for transformation composition that we can exploit for implementing transformation inheritance: *module superimposition* [55] (that allows for overlaying several transformation definitions and executing them as they were a single transformation, well supported by the most important transformation languages, including Atlas Transformation Language (ATL) [55]) and *rule overriding* (that substitutes an existing rule by a new one with the same name).

4.3 A reference architecture

In Figure 4.2 an overview of a possible architecture supporting the proposed methodology is reported. This architecture can be characterized as three-layers, three-tiers and three-users.

The three tiers (*Editing Tier*, *Transformation Tier* and *Analysis Tier*) represent the three phases of the generative approach. The first tier concerns to all those phase of model editing: in the proposed approach, the system high-level model must be produced. Since formal and multi-formal models are automatically obtained through transformations, the second tier regards these models. The third tier include all phases of the analysis (including the quantitative evaluation with analysis tools) that are the management of the concrete formal sub-models (those written in the specific syntax of the analyzer tool), the solution description and the evaluation of results.

The three users are the languages and transformations developers, respectively *Language Engineer*, *Software Engineer*, and *System Modeler*. The role of the first is to develop, manage and maintain the DSML and the language editor: he needs to individuate domain specific concepts, insert them in the DSML and check the coherence of syntax and semantics. The role of the second user is to define transformations: he starts from the DSML and develops all the transformation needed by the approach (those generating formal and multi-formal models, backward transformations aimed at refining high-level models with results obtained at the analysis tier).

Finally there are three horizontal layers (*Artifact Layer*, *Tool Layer* and *Repository Layer*) that represent the objects involved in each tier. The first layer is related to all those artifact that shall be produced during the application of the approach: different model are required at each stage of the methodology, starting from high-level model and formal models, reaching concrete sub-models (and input files) for the analysis tools.

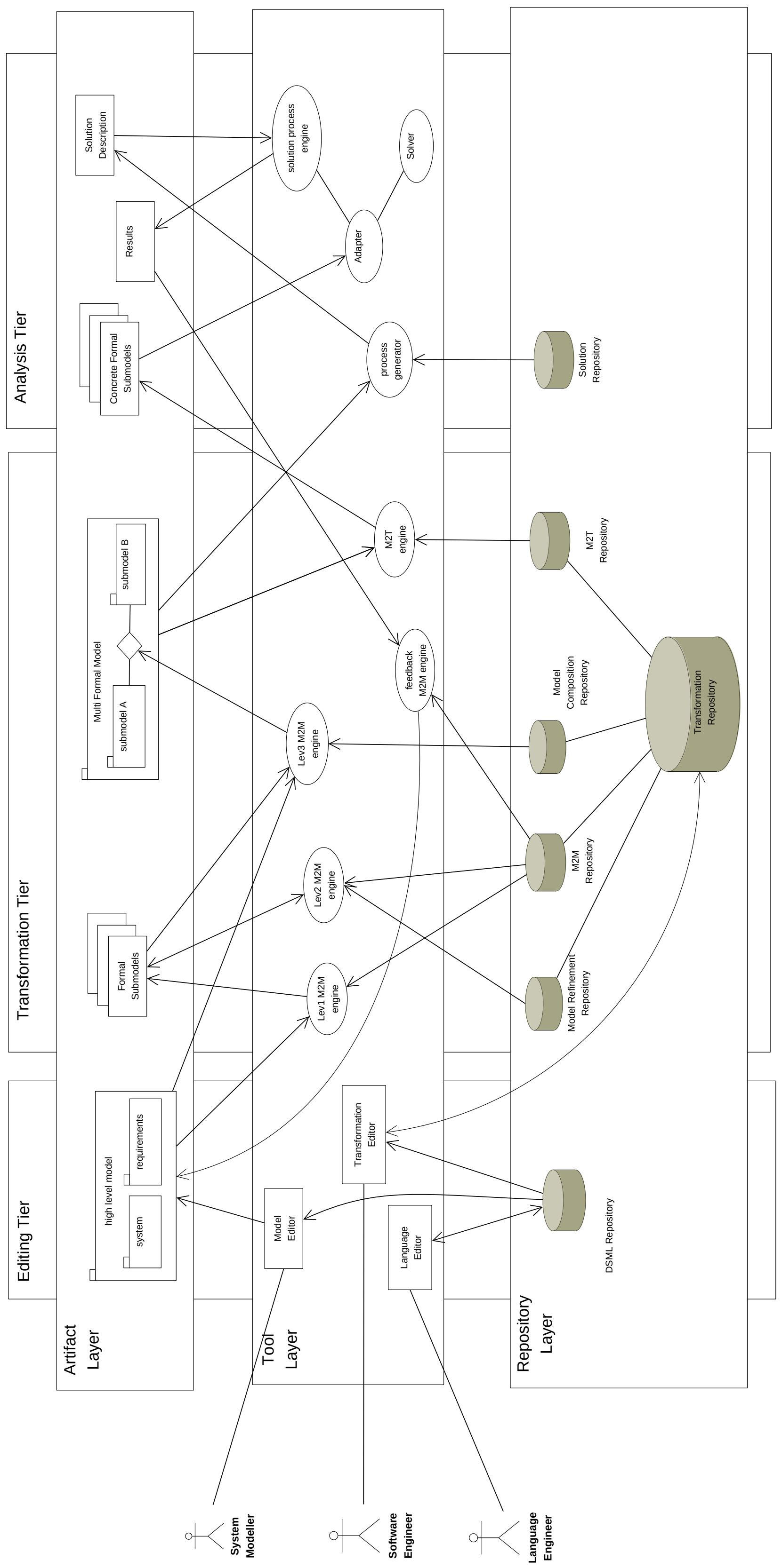


Figure 4.2: Reference architecture

At the second layer all the necessary tools are enclosed: there is the one supporting the DSML that can be any tool supporting the UML profiling technique, when this has been the adopted technique to develop the language; there is the transformation environment supporting their execution; there are all those tools required for the model analysis and described in the solution process engine. At the third layer there are all the repositories which can be used to store the information used/generated by the approach: these information can be also used to store some historical information (e.g. different versions of the system model following the design refinement and optimization as well as the different solution obtained during the system lifecycle).

Some existing tools (open-source and/or commercial) can be used in some points of the architecture: e.g. Papyrus, EMF/GMF, can be used as effective model editors; ATL for Eclipse can be used as transformation environment; Möbius can be used as tool supporting Fault Tree and SAN analysis; OsMoSys Multisolution Framework (OMF) elements can exploit the roles of tool and repository layers for multi-formal models.

4.4 Model transformations: from DAM to RFTs and BNs

To allow the approach adoption for the quantitative analysis of real-world systems, it is necessary to implement transformations from high-level DSMLs to destination formalisms. The work presented in this thesis offers two different transformations, which allow the automatic generation of formal models from UML dependability models, properly realized applying the DAM profile (Section 3.2).

DAM models may be automatically translated into formal models: this translation is performed by means of M2M transformations; in addition, according to the process described in Section 4.1, some M2T transformations are needed in order to serialize the obtained formal models into a concrete syntax, specific of the analysis tools. M2M transformations are defined on source and target meta-models. Here the source DSML is DAM, whereas we need Repairable Fault Trees (RFTs) and Bayesian Networks (BNs) as destination formalisms. Let us define the meta-models for both RFTs and BNs.

RFTs have been introduced to allow complex repair policies modeling and evaluation [56]. RFTs integrate Generalized Stochastic Petri Nets (GSPNs) and Fault Trees (FTs): repair policies are represented by nodes, called Repair Boxes (RBs), which encapsulate a GSPN model. The RBs are connected to a FT which describes the faults that may happen and their contribution to the occurrence of top event. A RB is applied to an event, and models a repair action that can be performed on the related system component. A simplified RFT meta-model is shown in Figure 4.3.

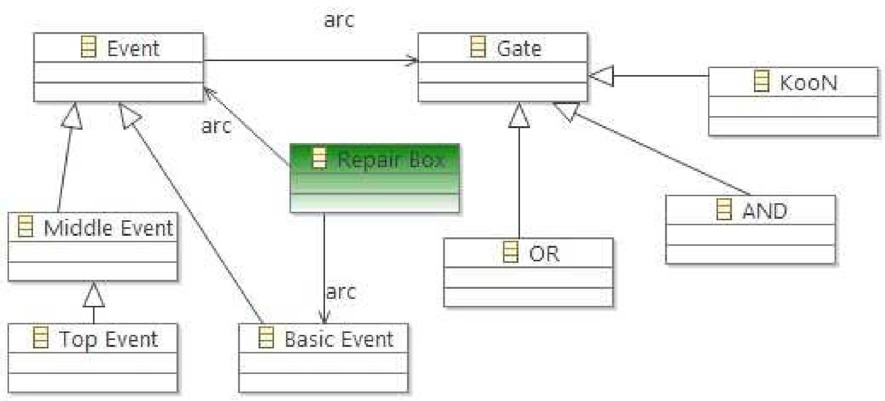


FIGURE 4.3: The RFT simplified meta-model

BNs allow to easily model common modes of failure and multi-state systems [57]. A BN is a probabilistic graphical model which is used to represent a set of random variables and their conditional dependencies through a Directed Acyclic Graph. Each node in the graph represents a random variable, while the edges between nodes represent probabilistic dependencies among the corresponding random variables. The conditional dependencies in the graph may be estimated by well known statistical and computational methods. A simplified BN meta-model is shown in Figure 4.4.

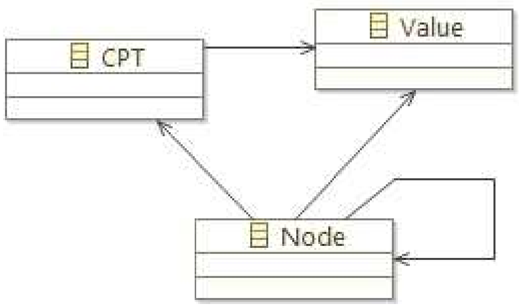


FIGURE 4.4: The BN simplified meta-model

One of the most critical issue in the application of transformational processes to critical systems is the correctness assessment of the transformations themselves. Different approaches have been proposed in scientific literature spanning from testing [58] to formal verification [59] but the problem still remains an open issue. This thesis does not focus on transformation verification: the transformations here introduced have been tested on some significant examples and then applied to the case studies. Future research effort will be spent on transformation assessment.

4.4.1 *DAM2RFT* transformation

DAM2RFT transformation generates RFTs from UML models, properly annotated with DAM stereotypes and tags. It exploits the strict relationship existing between FTs and

RFTs and the dependency relations existing among the DAM packages, as depicted in Figure 4.5. Hence, we adopt a *divide-et-impera* approach which exploits transformation composition and reuse, by applying module superimposition. In practice, superimposition allows for defining a new transformation from existing ones by the union of their set of rules. Hence, first a transformation from DAM to FT (*dam2ft* in Figure 4.5) is realized with the aim to generate the Fault Tree structure of the RFT model from the System and Threats domain models; then the proper rule set for the generation of Repair Boxes and related arcs from the Maintenance domain model is defined and *dam2rft* is obtained by superimposition.

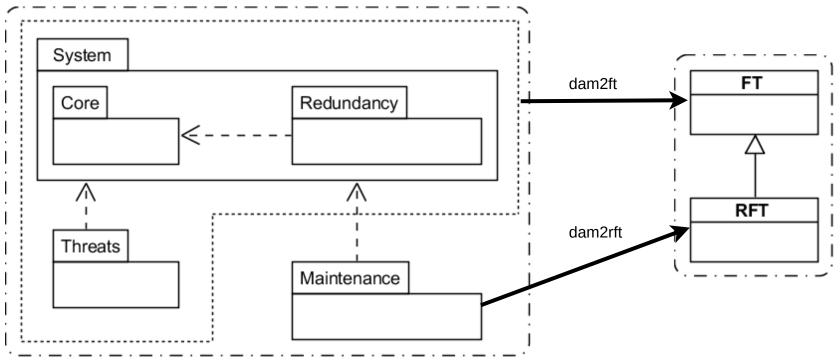


FIGURE 4.5: DAM-to-RFT transformation schema

In details, *dam2ft* works as follows: the Top Event is associated with the failure of the system which provides the service (specified by the Use Case Diagram). Events and gates are created from the Component Diagram by recursively applying the following rules:

1. for each *DaComponent*: if *fault* is null, one Middle Event is created; however, if *fault* is not null, one Middle Event and m Basic Events are created where m is the resource multiplicity (*resMult*);

2. for each *DaSpare*: if *fault* is not null, m Basic Events are created where m is the resource multiplicity (*resMult*);

3. for each Middle Event (created from a *DaComponent*) an Gate is created and arc to the Middle Event is set:

    - an OR gate if *DaComponent* does not belong to a *DaRedundantStructure*;
    - an AND gate if *DaComponent* belongs to a *DaRedundantStructure* and $FTlevel = 1$;
    - a KooN gate if *DaComponent* belongs to a *DaRedundantStructure* and $FTlevel = k$, $k > 1$;

4. when a sub-component relationship exists between a *DaComponent* X and a contained *DaComponent* Y and Y contains other elements, an arc is created from the Middle Event generated by Y to the Gate that is input of the Middle Event generated by X;

5. when a sub-component relationship exists between a *DaComponent* X and a contained *DaComponent*/*DaSpare* Y and Y contains no other element, an arc is created from the Basic Event generated by Y to the Gate that is input of the Middle Event generated by X.

The transformation implemented by *dam2rft* works under the hypothesis that the DAM specification includes a model of the repairing process of a sub-component and information about its steps (this can be expressed by means of a State Machine Diagram, an Activity Diagram, or a Sequence Diagram). The existence of a repair model associated to a *DaComponent* is annotated by a *DaActivationStep* stereotyped element. First a RB is generated for each replica of *DaComponent* and of its *DaSpare*, if any. The RB is filled with information about the MTTR and the resources (the repairmen) needed to accomplish the activity. This information is retrieved by the diagram used to describe the repair dynamics and by navigating the Component Diagram.

Finally, each RB is connected to the Fault Tree generated by *dam2ft* through repair arcs between: (1) RB and its triggering event, i.e. the Middle Event or Basic Event, from which the RB has been generated (the sub-system to be repaired); (2) RB and all the Basic Events that are present in the sub-tree whose root is the RB triggering event. Once the RFT model has been generated, a GSPN model is derived by applying another M2M transformation in order to allow easy analysis of this formal model. The description of this last transformation is reported in [54].

4.4.2 *DAM2BN* transformation

DAM2BN transformation generates BNs from UML models, properly annotated with DAM stereotypes and tags. This transformation woks under the hypothesis that the DAM specification includes, beyond a Use Case and a Component Diagram, also a behavioral model of the service, and specifically a State Machine Diagram which models the state transition caused by classes of failures. In details, *dam2bn* works according to the following rules:

1. for each *DaComponent* one BN node is created; the values of the random variable associated with the node are the elements of the *DaComponent*'s *failure* array plus

one (the nominal service mode); the Conditional Probability Table (CPT) is built according to its tagged values:

- if a *failure* element has a not null *occurrenceRate*, this rate is used to determine the probability associated with the value of the node;
- if a *failure* element has a not null *condition*, this condition defines the truth table used to determine the CPT;

2. for each *DaRedundantStructure* one BN node is created; the associated random variable has just two values corresponding to *nominal* and *failure* service modes and its CPT is built according to the rules defined in [60]. An arc is added from the node corresponding to the *DaRedundantStructure* and the node corresponding to the *DaComponent* which contains the *DaRedundantStructure*;
3. for each *DaService* one BN node is generated; the values of the random variable values and/or the CPT associated with the node are calculated from the states in the State Machine Diagram associated with the *DaService*;
4. if a *usedResources* tagged value of *DaService* is assigned, it is used to determine the BN nodes that must be connected to the one BN node translating the *DaService*; then proper arcs between such nodes are added;
5. if a UML dependency between two *DaService*s is found, it is used to connect their associated BN nodes;
6. for each *DaRedundantStructure*, the CPT of the related BN node is determined (in analogy to what specified at point 1.b) from the *condition* tagged-value in the *DaStep*s contained in the State Machine.

4.4.3 M2T transformations

M2T transformations are necessary to finalize the M2M transformation chains, in order to allow the usage of existing solver tools. After the two M2M transformations previously introduced, two M2T are necessary: the former transforms GSPN models into GreatSPN [61] input file format, the latter generates JavaBayes [62] compatible files from a BN model. The two transformations are implemented by means of ATL queries.

4.5 An example from automation domain

To give greater a practical impact of the proposed approach, this Section introduces an example taken from Flexible Manufacturing Systems (FMS) domain. In Figure 4.6, a simple FMS production cell is depicted.

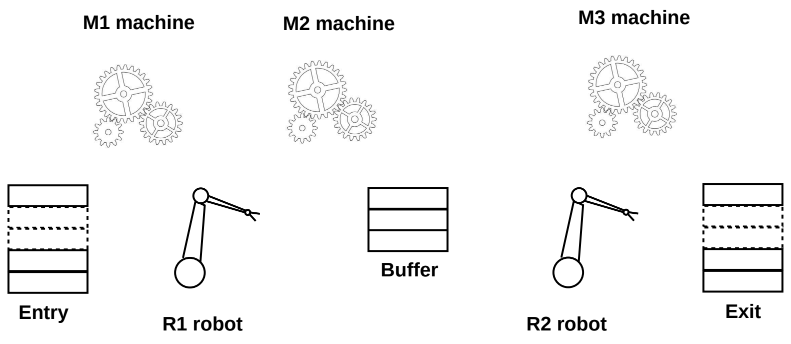


FIGURE 4.6: Simple flexible production cell

The production cell, depicted in the figure, is composed of entry and exit areas, buffer for temporary storage of unfinished parts, machines for their manufacture and robots for their handling. Raw materials and unfinished parts are loaded/unloaded into the cell through entry and exit areas; in this example we suppose that these two elements have infinite capacity to produce/consume parts. Materials are worked by machines that are able to accomplish some kinds of transformation on initial material. Materials can be moved only by armed robots that transfer parts from/to machines and other places inside the cell. At last semi-finished parts, that need to be further worked by machines, can be temporary stored into buffers (with a finite capacity). In particular we suppose that the robot R1 is fault-prone and can be repaired. We are interested in generating a model that can give us quantitative informations on the performance of the production cell in presence of failure of the only robot R1 (i.e. a performability model). At this purpose we need to create a multi-formal model, composed of a performance GSPN submodel of the entire cell and a reliability FT model of the R1 robot.

The quantitative evaluation of the performability of the system follows the stage of the proposed approach, that are: (1) definition of a proper domain model, (2) implementation of a DSML, (3) creation of an high-level model of the system to analyze. Then the model is transformed by (4) Level 1 M2M transformations, in order to generate formal models belonging to the the second stage. Finally the generated formal models are transformed into a multi-formal model by the (5) Level 3 M2M transformations. In the proposed example no Level 2 M2M transformations are required since there is not the necessity to change model formalism; for example, using them we could translate the FT model into a GSPN, in order to compose the two models generating a single GSPN global model.

4.5.1 Definition of the Domain Model

In Figure 4.7 the domain model is depicted: this class diagram reports all the concepts needed to describe the application domain. Focusing on *production cell* package, the five elements of the FMS description are represented by means of five conceptual classes and two abstract ones. The abstract classes are *Worker*, that represents the general characteristic of performing objects as robots and machines, and *Stage*, that is related to material transformation and storing entities. *Stage*s are connected between them in order to define a workflow of the material: the workflow is modeled by the *flow* self-association of the *Stage* class. The *serves* association, between the *Robot* conceptual class and the *flow* association, represents the movement of materials: in fact since *Stage* elements are unable to move material by themselves, *Robot*s are needed in order to "serve" material flows. Two properties are present in the domain model: the *rate* of *Worker*, which defines the production rate of machines and robots, and the *capacity*, which represents the maximum amount of material units that can be stored in buffers.

*Worker*s are fault-prone, thus it is natural to exploit existing languages that already capture dependability aspects. For this reason the domain model depends from the DAM domain model. In this way, *Worker* inherits from the *Component* conceptual class of the DAM domain model: this concept of DAM represents objects characterized by mean-time-to-fail (MTTF) and mean-time-to-repair (MTTR).

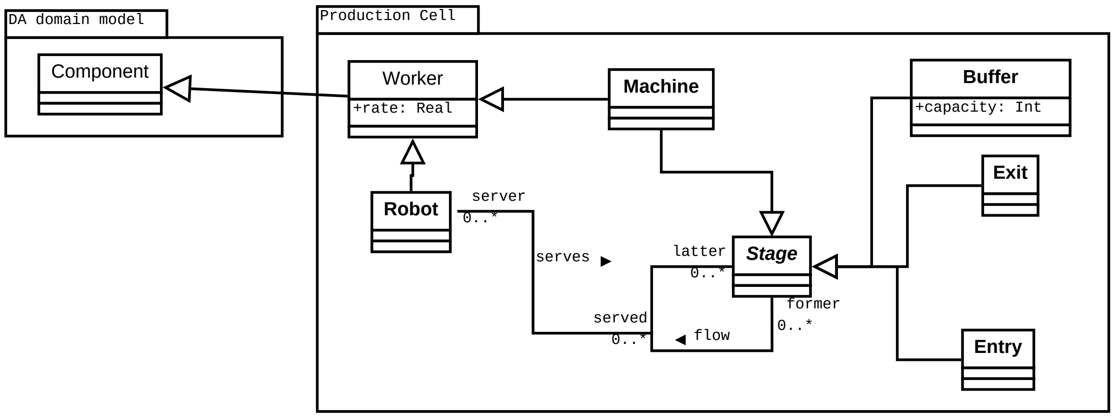


FIGURE 4.7: FMS domain model

4.5.2 Implementation of the DSML through the UML profiling technique

According to the domain model previously described, a UML profile can be developed. More in detail, the FMS profile imports the DAM profile, since the DAM profile is entirely reused for dependability modeling (Figure 4.8). The FSM profile contains the *production_cell* package, in which all stereotypes and tagged values related to *production cell* domain model are defined. In particular we define the five stereotypes related to the five conceptual classes, individuated by the domain model, which are: *Machine*, *Robot*, *Buffer*, *Entry* and *Exit*. Two abstract stereotypes (*Worker* and *Stage*) have been also defined to facilitate the overal structure, according to the one used in the domain model. Lastly, the *Worker* stereotype inherits from the *DAM::DaComponent* stereotype since it describes a faulty component.

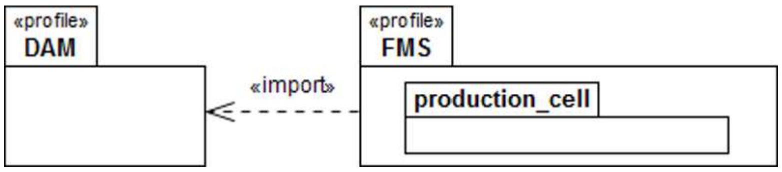


FIGURE 4.8: FMS Profile: package structure

4.5.3 Creation of the high-level model

The high-level model shall represent a production cell. Of course we implement the model using the *FSM* profile previously defined, to annotate all the characteristic that we need to model. The model consists of two UML diagrams: a class diagram (Figure 4.9) and a composite structure diagram (Figure 4.10). The first describes the layout of the production cell and highlights its elements by means of UML stereotypes and tagged values belonging to *production_cell* package; the second diagram models dependability aspects of the robot R1 (the one that is fault-prone). The robot R1 consists of three main components: a mechanical arm, a microcontroller and a power group. These elements are specified by means of DAM stereotypes; in particular we use: the *DaComponent* stereotype to specify elements that are characterized by a MTTR and/or a MTTF; the *DaRedundantStructure* stereotype in order to model redundant fault-tolerant structures and the *DaSpare* stereotype to model spare components.

4.5.4 Level 1 Model2Model transformation

In order to generate formal models, the target languages must be formalized according to model transformation techniques. Since we are interested in generating both a GSPN and a FT model, the syntaxes of these languages have to be defined in proper meta-models. Then two model transformations are created in order to proper translate a UML model. First the fault tree model, as in Figure 4.11, can be generated on the basis of robot structure, by means of the *dam2ft* transformation that has been already defined and implemented in [54].

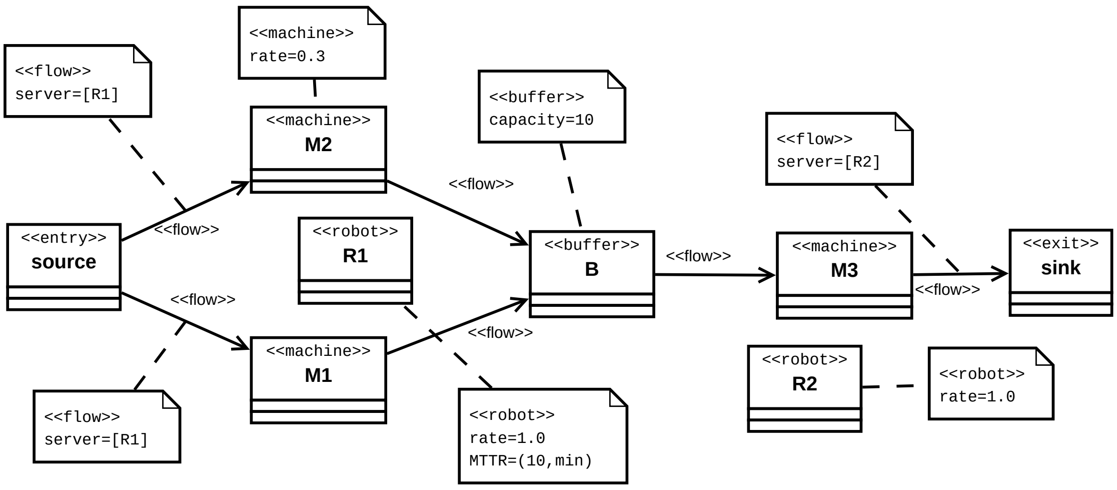


FIGURE 4.9: Model of the production cell

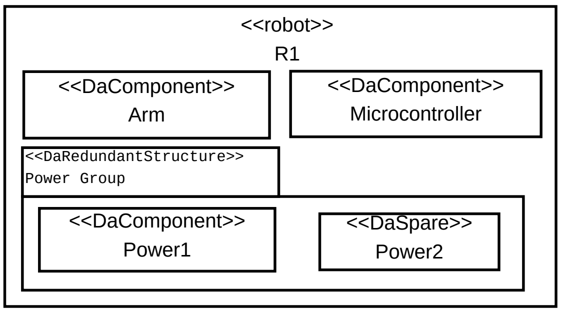


FIGURE 4.10: Model of the faulty robot structure

Moreover the *fms2gspn* transformation is used to generate the GSPN performance model of the production cell. This transformation consists of several rules, each of which triggers on a stereotype of *production cell* package (*Robot*, *Machine*, *Buffer*, *flow*, etc.) generating for each of them a GSPN subnet such as the one depicted in Figure 4.13 for the faulty Robot. All the subnets are linked together in order to produce a single GSPN performance model. The result of this transformation is depicted in Figure 4.12.

4.5.5 Level 3 Model2Model transformation

The FT and GSPN model are now ready to be transformed into a multi-formal one. In order to make this passage two sources of information are necessary. From the high-level model we understand which of the formal models are tied together: in the example, the R1 robot represents a single model element that is characterized by both performance and reliability aspects. Moreover the nature of the formal models (i.e. the formalisms in which they are expressed) determines how these models are tied together: in the example, the occurrence probability of the top event of the fault tree gives the rate of the transition of the subnet related to R1 robot. The formalization of this connection is made by means

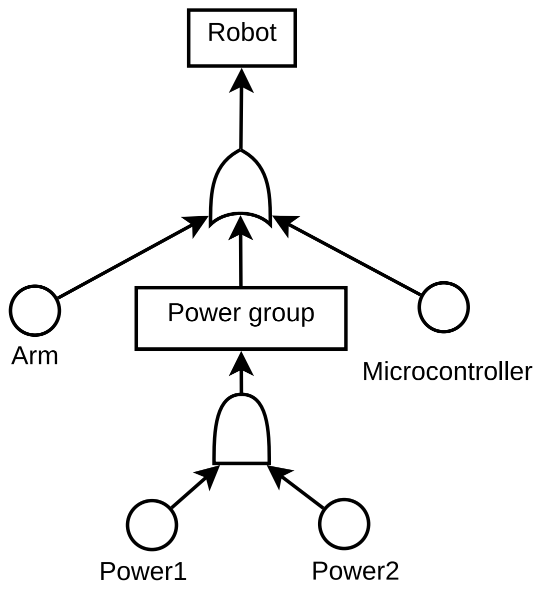


FIGURE 4.11: Fault Tree model of robot structure

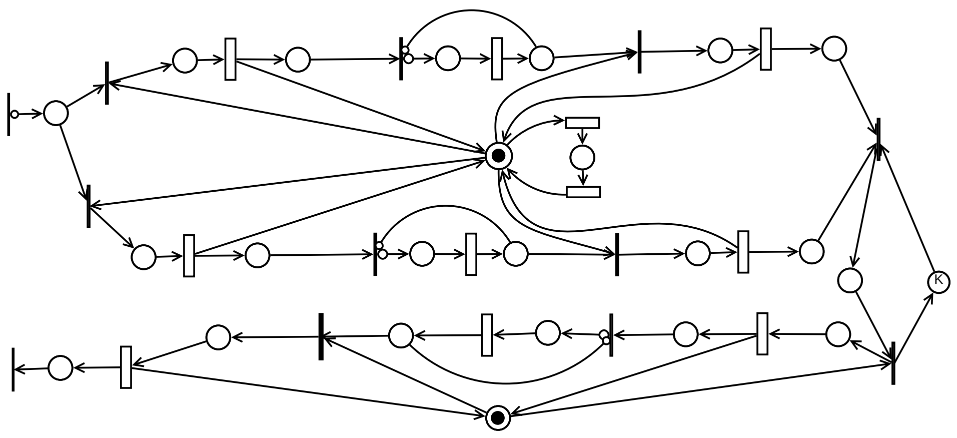


FIGURE 4.12: GSPN model of the production cell

of a compositional operator. In particular, according to the OsMoSys methodology, we encapsulate the models described before by means of the Copy-Elaborated-Result (CER) operator. According to the semantics of this operator, the result of the analysis of a FT model is copied as parameter of GSPN one. The resulting model (depicted in Figure 4.14) can be further analyzed by means of a proper automated solution process.

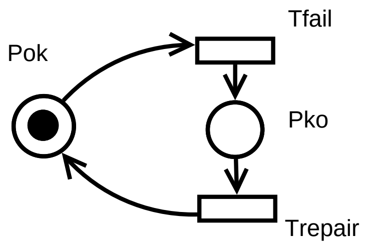


Figure 4.13: GSPN subnet of the faulty robot

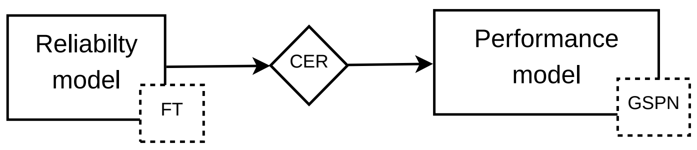


Figure 4.14: Multi-formal model

Chapter 5

New high-level modeling languages

The application of the approach described in Chapter 4 starts from the modeling of the system and its requirements in a DSML (*high-level stage*, described in Section 4.1). The modeling and analysis of different properties of the systems requires the usage of a specific DSML. With reference to the case study shown in Section 4.5, for example, it is important to note that the DSML has been created extending the horizontal language DAM, able to model dependability and maintainability aspects, while the modeling of domain specific features, as the *worker rate* or the *buffer capacity*, requires the specialization of the horizontal DSML with vertical concepts.

Obviously, the set of horizontal languages is not complete, since some critical properties have not been addressed in existing DSMLs. From the other side different application domains requires to be extended with new vertical concepts. Hence there is the need of both new horizontal and vertical languages: in this Chapter two new high-level languages are hence proposed: *DAM-Rail* and *CIP_VAM*. The first language extends DAM with specific concepts coming from the railway domain, while the second is a new high-level modeling language which is able to capture vulnerability characteristics of physical infrastructures. These languages will be used in Part III of this thesis to support high-level modeling and quantitative analysis of real-world systems during high-level stage.

5.1 RAM modeling and analysis in railway domain: *DAM-Rail*

The high-level modeling of RAM (Reliability, Availability, Maintainability) attributes of critical systems cannot exclude the application of the DAM UML profile as horizontal language. However, its application in railway domain, has shown the necessity to specialize it with vertical concepts: to this aim we successfully specialize DAM for the railway domain, creating the *DAM-Rail* language.

The objective of the DAM-Rail profile is to provide railway modelers with specific concepts, close to their experience and skills, that cannot be represented at DAM level. Of course, DAM stereotypes must still be used to model general structures: in fact the metro driverless vehicle model, in Section 7.2, makes use of both DAM and DAM-Rail stereotypes since, in this case, the focus is on railways service degradation categories.

5.1.1 DAM-Rail domain model

DAM-Rail adds railway concepts to DAM. Its domain model is organized as depicted in Figure 5.1. It consists of two main packages: *System* and *O&M*. *System* describes the structural features of the railway domain, *O&M* deals with the modeling of the railway Operation & Maintenance. The *System* package is, in turn, structured in several packages, each one addressing a specific technology.

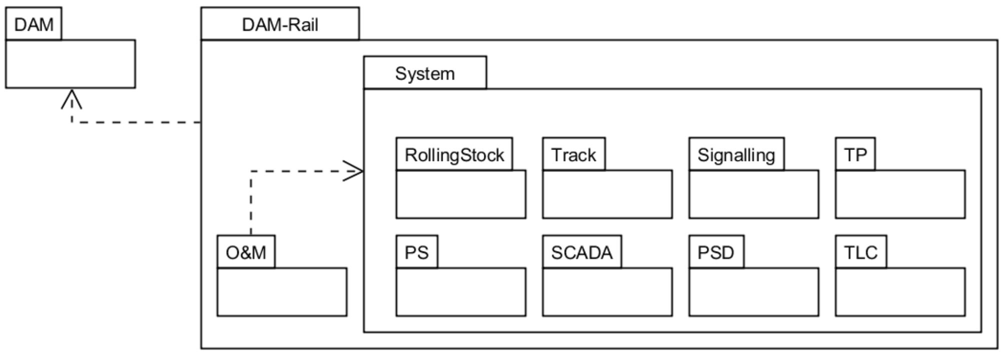


FIGURE 5.1: DAM-Rail package structure.

According to a classical railway organization, we identify eight sub-packages:

1. *RollingStock* includes concepts and their attributes for a complete vehicle description (e.g. vehicle length and engine power);
2. *Track* contains concepts related to the infrastructure as well as track elements (e.g. segment, crossing and point);

3. *Signalling* encompasses concepts of the signalling system (e.g. fixed and moving block);
4. *TP* includes concepts related to traction power (e.g. traction transformer, catenary and third rail);
5. *PS* contains concepts related to power supply (e.g. station cabinet and medium-voltage converter);
6. *SCADA* deals with supervisory control and data acquisition modeling (e.g SCADA rack and server);
7. *PSD* includes concepts related to platform screen doors subsystem (e.g doors engine and screen);
8. *TLC* contains concepts related to telecommunications technologies (e.g. terrestrial trunked radio and on-board system).

Hence a single package can be considered a "technological" DSML itself. As an example, an excerpt of the *RollingStock* package is shown in Figure 5.2: it describes the portion of domain concepts that will be used in Section 7.2 to implement the metro driverless model, needed to perform quantitative availability evaluation.

The concepts included in the *RollingStock* package concern with the vehicle and its subcomponents. They are:

- *Vehicle*: it expresses the railway vehicle concept, characterized by the number of cars, the length and other attributes (not shown in the Figure 5.2);
- *Pantograph*: it describe the pantograph, which capture energy from the power line and transfer it to the engine; it is characterized, for example, by the distance between the vehicle and the catenary;
- *Engine*: it is used to annotate specific features of the engine, such as its power;
- *Wheel*: it describes the wheels of a vehicle, specifying its characteristics as the diameter and the border thickness;
- *AirCompressor*: it represents the air compressors unit installed on a vehicle, specifying, for example, its pressure;
- *GasTank*: it expresses the gas tank concept, describing the power measured as British Thermal Units (BTUs).

Each of them extends the *Component* class of DA domain model of DAM, by adding the specific attributes representing domain specific features. In a similar way other packages have been defined.

The *O&M* package deals with railway concepts of Operation & Maintenance, focusing primarily on the features to be considered in the evaluation of railway services. It is important to consider that, in the railway domain, a common approach defines degradation categories for each service. Figure 5.2 depicts also an excerpt of this package including the degradation categories (Service Modes - SMs) related to the services provided by vehicles. In detail:

- *DelaySM* may be used to specify the expected minimum and maximum delays at the end of the trip (for example, concerning the *transport* service, a "good" level is when the delay is minor than 3 minutes, while an "acceptable" level is when the delay is included between 3 and 10 minutes);
- *SpeedSM* may be used to specify the maximum speed (ratio) reached by the vehicle (for example it is "good" with a maximum speed of 100% and "acceptable" with a maximum speed of 60%);
- *CoolnessSM* may be used to specify the refrigeration ratio inside a car (for example it is "good" when is 100% and "acceptable" when 60%);
- *PowerCaptureSM* may be used to specify the power capture capability ratio of a vehicle from the electrified line (for example it is "good" when is 100% and "acceptable" when 60%).

Notice that, by inheritance, the relationships between components and services are derived from DAM, as well as their threats characterization. Similarly, the impact of impairments on services is already fully modeled in DAM (through the two associations between *Component* and *Service* and between *Service* and *Step*) and it does not require to be specialized in DAM-Rail (see Figure 5.2).

5.1.2 DAM-Rail profile

The DAM-Rail packages are directly mapped into the DAM-Rail profile. The concepts introduced by *RollingStock* and *O&M* are mapped, one by one, on UML extensions through stereotypes: each stereotype inherits from *DAM::DaComponent* or from *DAM::DaStep* stereotypes. For example the *Engine* conceptual class, belonging to the DAM-Rail domain model, is translated into the *Engine* stereotype in the DAM-Rail

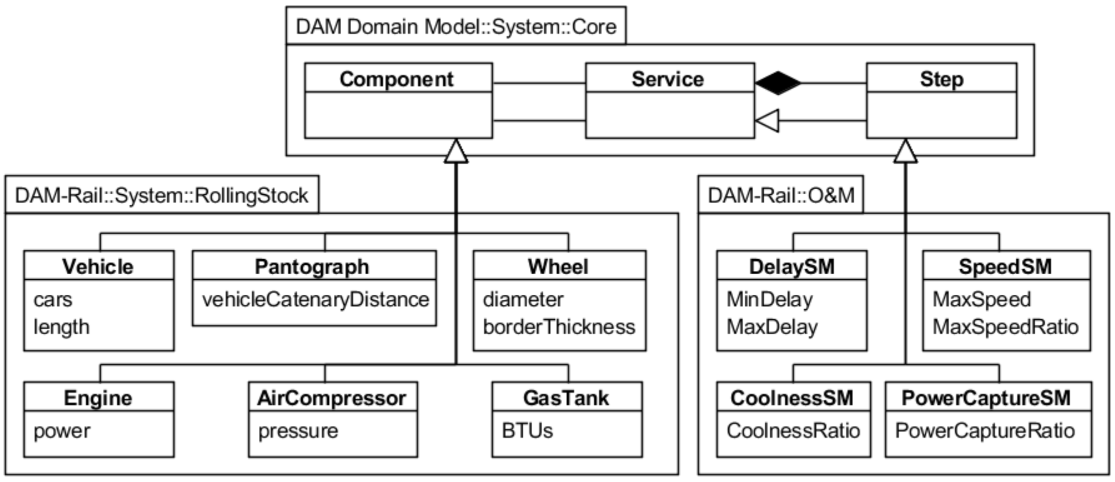


FIGURE 5.2: DAM-Rail *RollingStock* and *O&M* packages (excerpt)

profile inheriting from *DAM::DaComponent*, as well as the *DelaySM* conceptual class is translated into the *DelaySM* stereotype inheriting from *DAM::DaStep*.

In Table 5.1 a partial list of the DAM-Rail stereotypes (those shown in Figure 5.2) and their attributes is reported.

TABLE 5.1: DAM-Rail stereotypes and tags (portion).

Stereotype	***Inherits / Extends***	***Tags: type***
System::RollingStock		
Vehicle	DAM::DaComponent / -	cars: Integer[0..1] length: NFP_Length[0..1]
Pantograph	DAM::DaComponent / -	vehicleCatenaryDistance: NFP_Length[0..1]
Engine	DAM::DaComponent / -	power: NFP_Power[0..1]
Wheel	DAM::DaComponent / -	diameter: NFP_Length[0..1] borderThickness: NFP_Length[0..1]
AirCompressor	DAM::DaComponent / -	pressure: NFP_Real[0..1]
GasTank	DAM::DaComponent / -	BTUs: Integer[0..1]
O&M		
DelaySM	DAM::DaStep / -	MinDelay: NFP_Duration[0..1] MaxDelay: NFP_Duration[0..1]
SpeedSM	DAM::DaStep / -	MaxSpeed: NFP_Real[0..1] MaxSpeedRatio: NFP_Percentage[0..1]
CoolnessSM	DAM::DaStep / -	CoolnessRatio: NFP_Percentage[0..1]
PowerCaptureSM	DAM::DaStep / -	PowerCaptureRatio: NFP_Percentage[0..1]

5.2 Vulnerability high-level modeling and analysis: the *CIP_VAM* language

The vulnerability high-level modeling of Critical Infrastructures (CIs) has been not successfully treated in literature. Several approaches have been taken from the literature to model CIs under several aspects but, at best of our knowledge, few works focus on the design and development of DSMLs for modeling CIs vulnerability and protection.

In [63] UML-CI is introduced, a UML profile aiming at defining different aspects of an infrastructure organization and behavior. Other works address several issues not specifically related to CIs. For example, the CORAS language is oriented to risk analysis of changing systems [50], UMLsec allows for expressing security information in system specification [64].

For these reason we propose to extend directly UML with new vulnerability and protection modeling capabilities through the definition of a new UML profile named CIP_VAM profile (Critical Infrastructure Protection, Vulnerability Analysis and Modeling). The original contribution of CIP_VAM is to correlate both infrastructure and attack **with the protection system**, applied to defend the assets. The CIP_VAM language is general, i.e. it may be applied to the protection of physical assets: it can be considered an horizontal language similarly to the DAM language.

With the aim to provide the language with a clear structure and with all the information needed to its complete definition, the definition of CIP_VAM has followed three main steps:

1. identification of the main levels in which the domain under study could be logically decomposed, in order to separate the concepts belonging to different semantic concerns;
2. characterization of the entities and concepts involved at each level; this step has been performed by identifying and studying the main available sources of information: information retrieved from public databases, interviews with domain experts, current literature (CIs modeling, domain specific modeling languages and dependability profiles, protection systems and localization models, attacks classification);
3. identification of the relationships between the domain levels and of the consequent dependencies among concepts and entities across the levels.

Three different levels have been identified:

- the *Infrastructure* level models assets (under physical, functional and economic aspects) that must be protected and their relationships;
- the *Attack* level deals with attacks and sabotages that can occur against an infrastructure;
- the *Protection* level focuses on devices and techniques used to protect assets.

Since we choose to design and implement CIP_VAM using UML, a UML profile has been developed, according to the guidelines provided in [45, 65]. At first the definition of the Vulnerability Analysis domain model was given. That means to define a conceptual model for each level (from step 1), including the main language constructs representing the essential concepts (characterized in step 2) and the set of valid relationships that must be considered between the above domain concepts (identified in step 3). Of course, the domain conceptual model also includes the set of constraints that drive the way in which the language constructs may be combined to build models, the concrete syntax of CIP_VAM and its semantics, i.e. the correspondence between the elements of the abstract syntax and the domain entities which are represented by the syntactical elements of UML.

Once the conceptual model was completed, the domain models have been mapped to the Vulnerability Analysis Modeling Profile (the *CIP_VAM* profile), by identifying for each domain concept the most suitable UML notation. In the next subsections we introduce the Vulnerability Analysis domain model and the CIP_VAM profile.

5.2.1 The Vulnerability Analysis domain model

CIP_VAM is organized in three packages (Figure 5.3), each one related to one of the three levels *Infrastructure*, *Attack* and *Protection*. The CIP_VAM domain model is general enough, i.e. it may be applied to the protection of physical assets of whatever application domain. In detail:

- the *Infrastructure* package provides a full description of the physical system to be analyzed; in particular, it contains asset and environmental related concepts. At the state we specifically cope with physical assets, nevertheless CIP_VAM can be easily extended to include non-physical assets;
- the *Attack* package introduces the threats due to attack events conducted against the assets within the infrastructure;
- the *Protection* package introduces protection related concepts and provides a description of techniques and countermeasures which may be applied to defend an asset.

Dependencies between packages stand for relationships between concepts from the involved levels. It means that at least one model element of one package requires at least one model element of the other package for its specification or implementation, or rather the complete semantics of the depending element is either semantically or structurally

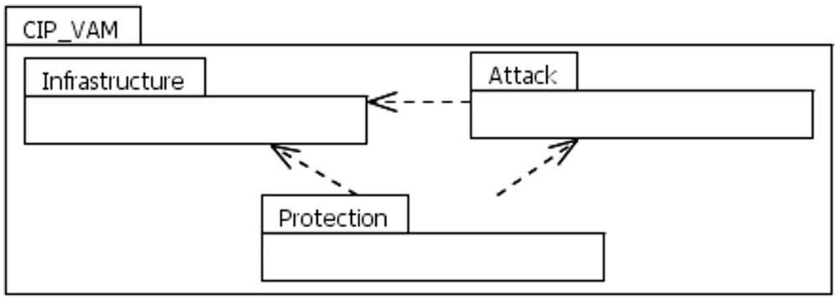


FIGURE 5.3: Package organization of *CIP_VAM* domain model

dependent on the definition of the supplier element. Three different dependency relationships are present. Within this context, the target of an attack is always an asset, hence a relationship exists from *Attack* to *Infrastructure*. The *Protection* package has dependencies relationships with both *Infrastructure* and *Protection* packages because a protection is applied to a specific asset against one (or more) class of attacks. More details about the dependency relationships will be given in the following, during the description of the model elements belonging to these packages.

***Infrastructure* package.**

The first of the three levels introduced in CIP_VAM is the *Infrastructure* level. It is easy to understand that to study and analyze a CI, it is necessary to decompose and represent it as a set of smaller subcomponents. More precisely criticality and vulnerability levels associated to a CI is strongly affected by the criticality and vulnerability levels of the CI subcomponents, even if it is not given by the sum of the values associated with the components. For this reason the *Infrastructure* level has the aim of studying and analyzing the main subsystems and subcomponents of a CI.

The Infrastructure domain model is depicted in the Figure 5.4. Following the description of the package scope of modeling given before, this model formalizes the elements concerning with physical infrastructure modeling. In detail, four main concepts have been identified: *Site*, *Interface*, *Object* and *Service*. These concepts are represented by four UML classes of the conceptual class diagram. The *Site* class shows the *subSites* self-association: it is a UML composition due to the fact that a contained site has no reason to exist without the external container, in other words the elimination of the more external site provokes the elimination of the internal sub-sites. Similarly, the *Service* class has a self-association named *subService*: also in this case a contained service has no reason to exist without the external container. On the contrary, the *subObject* association is a UML aggregation of two objects, in the sense that a contained element can be autonomous and independent from the container.

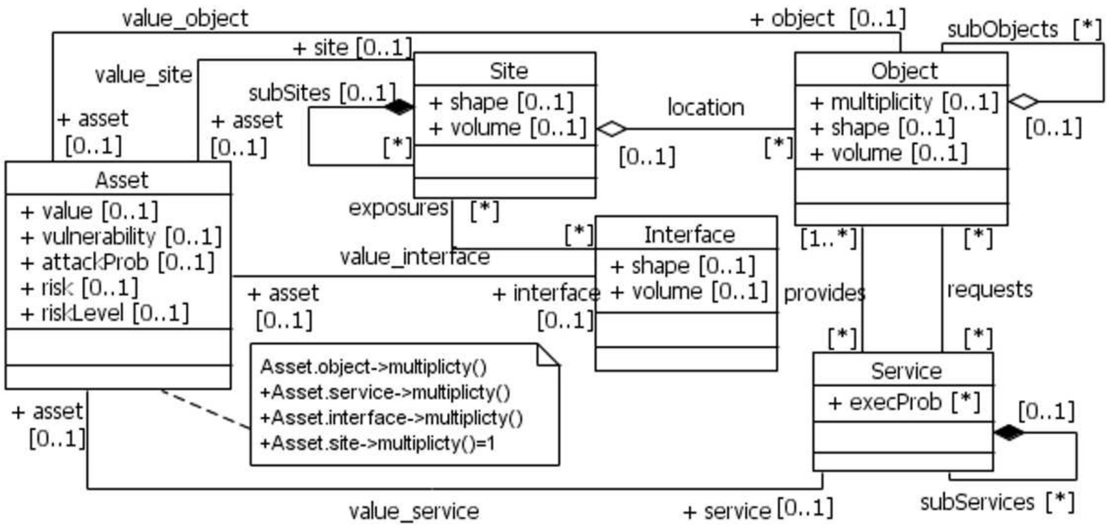


Figure 5.4: *Infrastructure* domain model

Another UML aggregation is named *location*: this aggregation expresses the physical location of several objects inside a site. Two different association are established between object and service named *provides* and *requests*: they model the situation of an object that can provide and/or requests different services. A UML association is established between the *Site* and the *Interface*: the *exposure* association models the interfaces between sites, as the doors between two rooms, or an intersection of different corridors.

Notice that *Asset* conceptual class plays a particular role in the infrastructure model: it encompasses the concepts needed to vulnerability analysis and it may be "applied" to sites, interfaces, objects and services. Specifically, an asset is characterized by the following attributes: economic value (*value*), likelihood that an attack is successful given that it is attempted (*vulnerability*), probability of an attack (*attackProb*), quantitative estimate of potential for an unwanted outcome (*risk*), qualitative estimate of risk (*riskLevel*). Sites, interfaces, objects and services may be assets if an economic value is given to them: the constraint on the *Asset* class (the note on the Asset in Figure 5.4) guarantees that if one instance of *Asset* exists then exactly one association is established with an instance of either *Site*, or *Interface*, or *Object*, or *Service*.

***Attack* package.**

The second of the three levels introduced in CIP_VAM is the *Attack* level. This level allows to represent the main concepts concerning attacks and sabotages that can occur against a CI. Obviously, this level is in a strict relationship with the *Infrastructure* one, since the objective of an attack is to damage, destroy, sabotage an asset causing a loss that can be economically quantifiable (given the asset specification). The effect of an attack, if it is successful, against the asset depends from its nature and from the asset characteristics. In fact, it is impossible to destroy a service inside a CI, while it is possible to cause a failure in it; from the other hand it does not make sense to provoke a failure in a physical site, where it is possible to damage or to violate it, if not public.

The *Attack* domain model is depicted in Figure 5.5. It depends on the *Infrastructure* package since some model elements belonging to the latter are necessary to properly define some model elements of the first. The main class of this package is *Attack*, which depends on *Asset* (from the *Infrastructure* package) through the *Threat* association. *Threat* models effects that the attack wants to cause on the asset. The *Attack* shows also the self-aggregation *subAttacks* due to the fact that it can be specified by a compositional relationship. The attack is also associated with the *Action* class, representing its compositional nature.

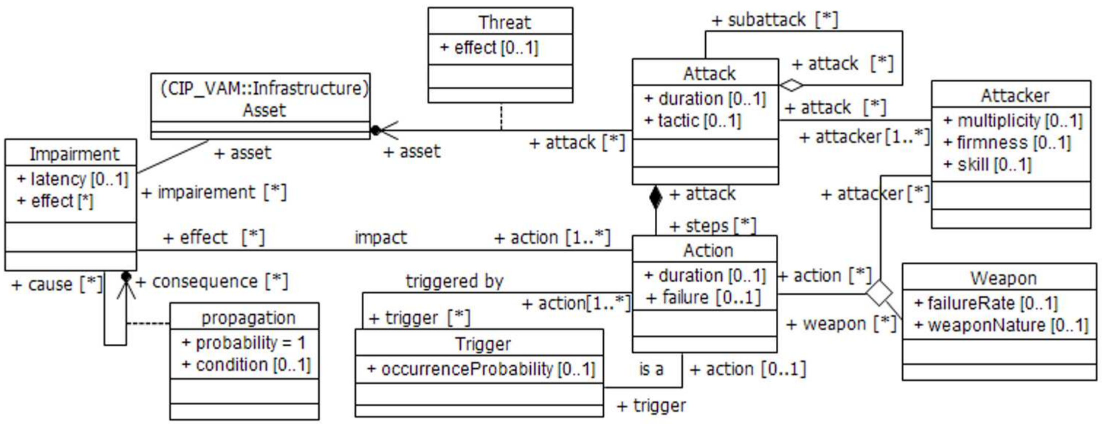


FIGURE 5.5: *Attack* domain model

Both *Attack* and *Action* concepts can be characterized over time by a temporal *duration* (i.e., the time elapsed between the start and the end of the attack/action); *Attack* is also characterized by *tactic*, that is the nature of the attack (kidnapping, armed attack, sabotage, etc.). The impact of the actions on the asset is modeled by the *Impairment* class.

The *propagation* self-association indicates how a damage can propagate to other impairments, with a given *probability* and under specific *conditions*.

Attacker models the misactors which perform the actions. An attacker is characterized by: number of physical people performing an action (*multiplicity*), psychological predisposition to overcome deterrents and defence systems (*firmness*), capability exploited in carrying out an attack effectively (*skill*). *Weapon* represents the weapon used by the attackers to perform the actions. A weapon is characterized by its failure probability (*failureRate*), for example a bomb that does not explode, and by its type (*weaponNature*). A ternary association models the relationship among *Attacker*, *Action*, and *Weapon*.

A *Trigger* is an event enabling an action: since it is also an action, it is in a specialization relationship with the *Action* class. It has an occurrence probability (*occurrenceProbability*). The association *triggeredBy* relates the *Action* class with the *Trigger* class, specifying which are the objects, instances of the latter class, that enable the execution of an object, instance of the first class. One of the main assumption at the base of this domain model is that CIP_VAM does not want to model all the possible attack brought to all the assets of the CI but rather than quantify a modeled attack against a single asset. For this reason multiplicity of the *Threat* association class is *1-to-n* and not *m-to-n*.

***Protection* package.**

The third level of the CIP_VAM domain model is the *Protection* level. A huge number of protection and defence mechanisms can be employed in the design of a CI security system. Most of them are devoted to the detection of specific kinds of attack. This level wants to represent the entire set of protection systems nowadays available in CIs. Obviously, this level is in a strict relationship with both the *Infrastructure* and the *Attack* levels since the aim of a generic protection system is to protect the asset against the attacks. All these systems must be installed within a RIS; with respect to physical protection all the available technologies can be applied both on a site, an object and an interface.

The *Protection* domain model is depicted in Figure 5.6. At this step of the study, protection procedures are not considered, but the proposed framework can be easily expanded to support such facilities.

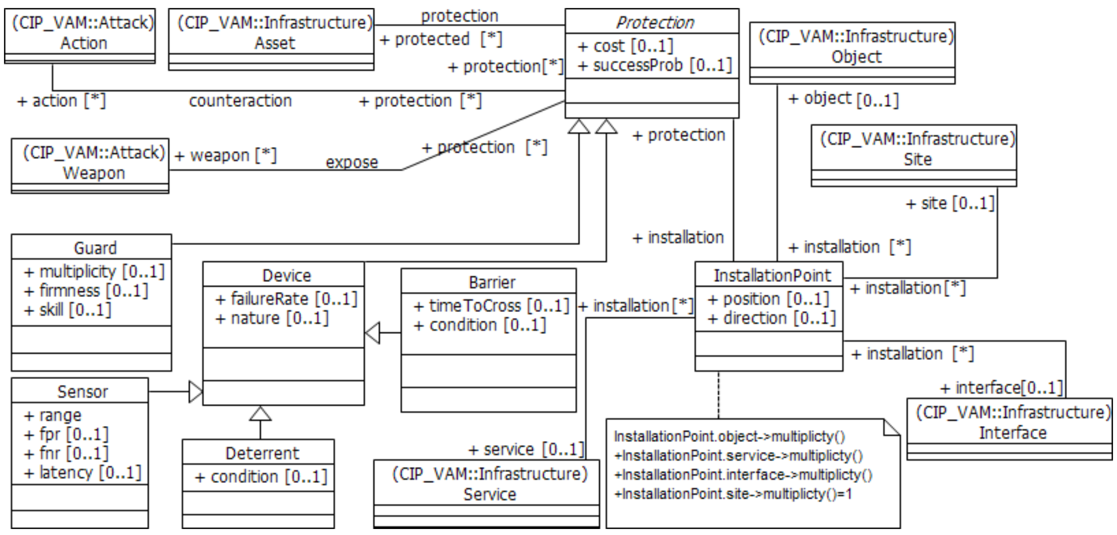


FIGURE 5.6: *Protection* domain model

Protection is an abstract class modeling a protection/defence system: it is characterized by a *cost* and the probability to be effective against an attack (*succesProb*). A protection may be applied (through an instantiation point) to a site, an interface, an object or a service in order to protect an asset. *InstantiationPoint* also specifies the *position* of the application point of the protection system (e.g., the height where a closed circuit video camera is applied on a wall) and the *direction*. The abstract concept of *Protection* is specialized by two concrete classes: *Guard* and *Device*. *Guard* models the presence of one or more guards or, in general, of a security service based on human personnel. In turn, *Device*, that is characterized by a failure probability (*failureRate*) and a nature, is further specialized by the classes *Barrier*, *Sensor*, and *Deterrent*. A barrier has the aim to lock an interface (to which it is associated), or to increase the time and the difficulty to overcome it. The attribute *timeToOvercome* specifies the crossing time of the barrier. A sensor is characterized by the maximum extension of the area covered by the sensor (*range*), and by its false positive/negative rates (*fpr*, *fnr*). A deterrent is whatever is able to demoralize an attacker from performing a malicious action and it may be characterized by the condition under which the deterrence has effect (*condition*).

5.2.2 The *CIP_VAM* Profile

In this Section the mapping from the conceptual domain model to a concrete UML profile is described. Each class, as well as its attributes, associations and constraints, is considered, and the most suitable UML concepts are identified, according to the guidelines expressed in [45, 65]. The outcome of this iterative process is the *CIP_VAM* profile which provides the UML extensions to be used to represent vulnerability and protection aspects in a UML model. These extensions allow to refine standard UML semantic adding specific concepts for vulnerability analysis of CI. The *CIP_VAM* profile includes the definition of stereotypes, tagged values and constraints together with a library of specific data types. The profile consists of the *CIP_VAM Library* and a set of *CIP_VAM UML extensions* as shown in Figure 5.7.

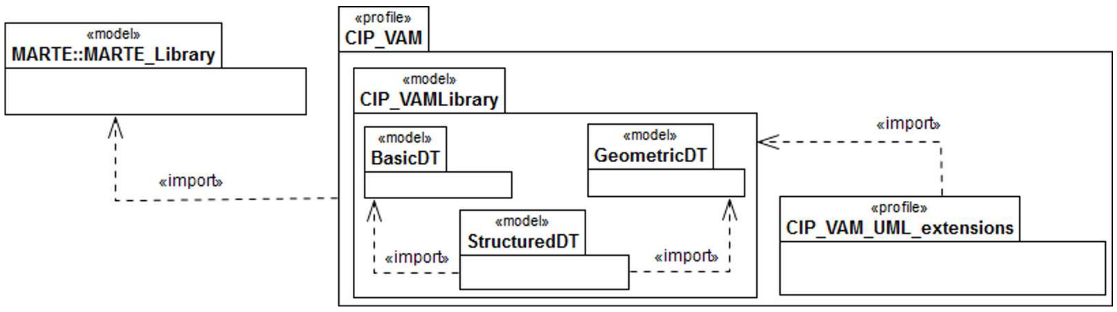


FIGURE 5.7: *CIP_VAM* profile

The UML extensions package contains the stereotypes, their attributes (tagged values) and the constraints which extend the UML meta-model.

The *CIP_VAM Library* defines the whole set of basic and complex data types necessary to the correct definition of the tagged values while the *CIP_VAM UML extensions* encompasses the set of stereotypes and constraints necessary to extend the UML metamodel.

As explained here below, the *CIP_VAM Library* imports the MARTE Library that is part of the MARTE UML profile [48]. Specifically, CIP_VAM uses MARTE's primitive types and its capability to specify quantitative and qualitative Non-Functional Properties (NFP) through its Value Specification Language (VSL). In addition the *CIP_VAM* profile uses the Measurement Units package to represent quantity, measure, unit, unit conversion for the most used physical quantities such area, length, time, energy, etc.

The *CIP_VAM Library.*

The *CIP_VAM Library* defines basic, geometric and structured data types and it is structured into three packages, *BasicDT*, *GeometricDT* and *StructuredDT*, respectively (see Figure 5.7). Obviously the last package imports the other two in the sense that the definition of some properties in its specified, depends on the namespace of the other two packages.

The *CIP_VAM Library* is composed as following: a set of simple enumerations are defined in *BasicDT* (Figure 5.8(a)); the geometric data types in *GeometricDT* are necessary in order to model physical structures and spaces, e.g. the localization of objects inside a space and the dimension of both objects and sites (Figure 5.8(b)); the *Structure* package defines complex data types by aggregating *BasicDT* and *GeometricDT* types (Figure 5.8(c)). Both *GeometricDT* and *StructuredDT* use types defined in the MARTE Library.

BasicDT. The *BasicDT* (Figure 5.8(a)) defines the following enumerations:

- *RiskLevel* represents a qualitative classification of the degree to which an asset might be exposed to potential losses. Possible values are: negligible, acceptable, tolerable and unacceptable.
- *ProtectionNature* represents the nature of a protection. Possible values are: acoustic, biological, chemical, electric, magnetic, mechanical, optical, radiation, thermal.
- *Tactic* is the physical nature of the weapon used to bring on the attack (or a single action inside a complex attack). Possible values are: armed attack, arson, barricade, bombing, hijacking, hostage, intrusion, kidnapping, sabotage, suicide.

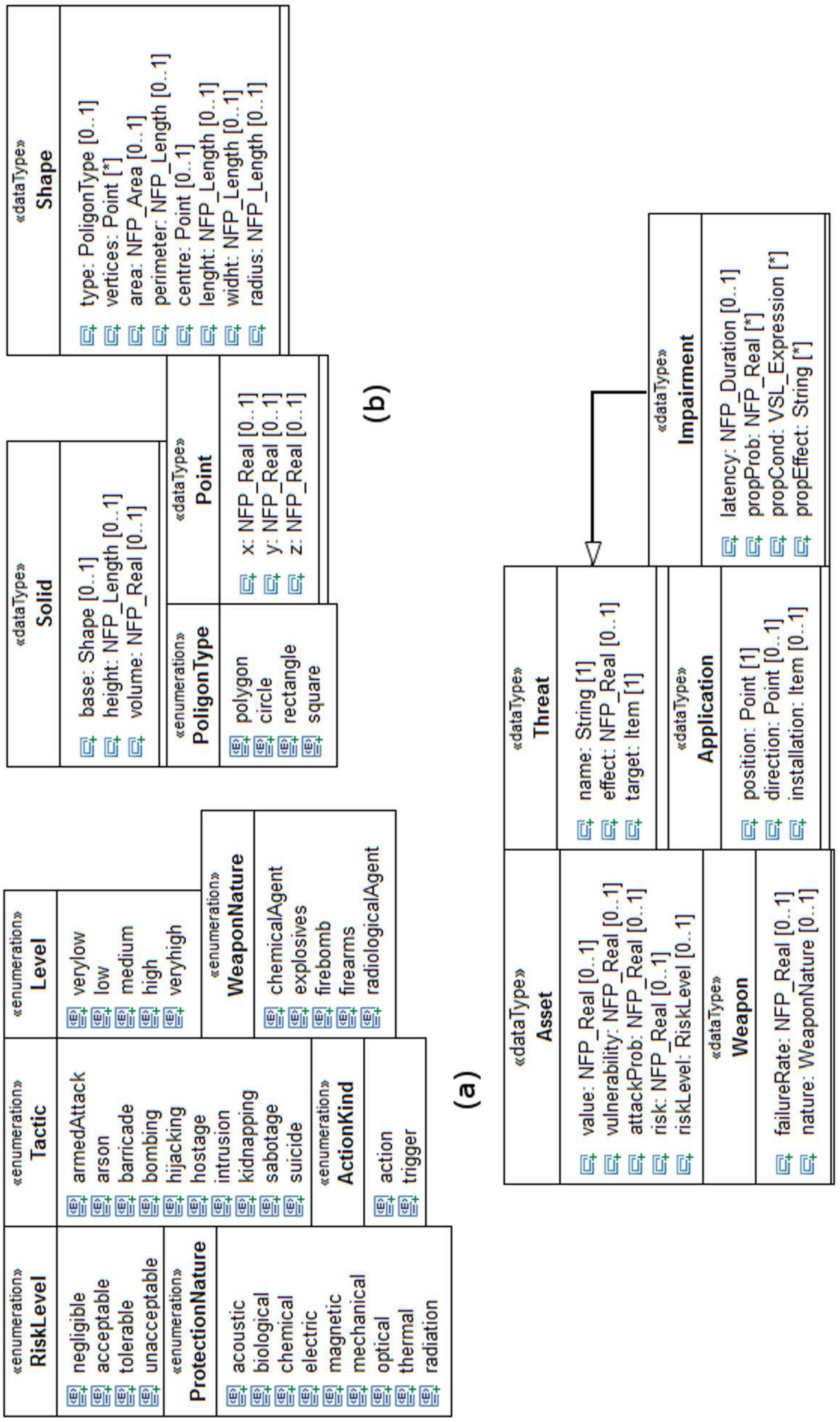


FIGURE 5.8: *CIP_VAM Library*: (a) Basic enumeration types, (b) Geometric data types, (c) Structured data types

- *ActionKind* allows to discriminate between simple actions and triggers. In fact possible values are: action and trigger.
- *Level* represents a generic qualitative level, it is used in different parts of the profile such as motivation and skill levels of both attackers and human defenders. Possible values are: very low, low, medium, high and very high.
- *WeaponNature* represents the nature of a weapon. Possible values are: chemical agent, explosives, firebomb, firearms, radiological agent.

GeometricDT. The geometric data types are necessary in order to model physical structure and spaces: the localization of objects inside a space and dimension of both objects and site. Figure 5.8(b) represents the *GeometricDT*: both enumeration and structured data types are present. In detail:

- *PolygonType* is an enumeration of simple geometrical 2D shapes. Possible values are: polygon, circle, rectangle and square.
- *Point* represents a point in a 3D space. It is a datatype (a tuple) having three different fields:
  - *X*: x-axis coordinate of the point. It is a real value and it is optional since the point can only have y and z coordinates;
  - *Y*: y-axis coordinate of the point. It is a real value and it is optional since the point can only have x and z coordinates;
  - *Z*: z-axis coordinate of the point. It is a real value and it is optional since the point can only have x and y coordinates.
- *Shape* represents a 2D shape. It is a datatype (a tuple) having several fields:
  - *type*: it is a Poligontype variable and assigns the type to the shape. It is optional since the shape can be of a type not in the PolygonType set of values;
  - *vertices*: list of Points that constitute the border of the shape; vertices have an undefined number of Points;
  - *area*: value that represents the numerical value of the area of the shape;
  - *perimeter*: value that represents the length of the border of the shape;
  - *centre*: it is a Point that represents the barycentre of the shape;
  - *length*: length of the shape;
  - *width*: width of the shape;

  - *radius*: for circular shape, it indicates the radius of the shape.

- *Solid* represents a 3D geometrical volume. It is a datatype (a tuple) having three different fields:
  - *base*: shape describing the base of the solid;
  - *height*: value that represents the measure of the vertical dimension of the solid;
  - *volume*: it represents the volume of the solid.

StructuredDT. The *StructuredDT* package is related to complex datatypes, created reusing those defined in *BasicDT* and *GeometricDT* packages. Figure 5.8(c) depicts the structure of this package.

The types contained in this package are the following.

- *Weapon* represents a weapon used as a tool in an attack phase (an action). It is a dataType having two different fields:
  - *failureRate*: rate of failure of the weapon;
  - *nature*: physical nature of the weapon (kidnap, firearm, etc...) determined according to the AttackNature type previously expressed.
- *Asset* is the data type related to the economic values and risk of an asset:
  - *value*: economic value of the asset;
  - *vulnerability*: probability of being damaged given an attack;
  - *AttackProb*: quantification of the probability being attacked;
  - *Risk*: quantification of the risk associated with the asset (according to the well-assessed formula Risk = attackProb *Vulnerability*damage);
  - *riskLevel*: qualitative level of the risk.
- *Application* represents the localization of the installation of a protection onto an item (site or object):
  - *position*: physical location of the application;
  - *direction*: orientation of the protection (let us consider as an example a camera that wants not only the point on which it has been fixed but also the one where the camera looks to);
  - *installation*: Item on which the protection is installed.

- *Threat* represents a threat brought by an attack to an asset:
  - *name*: name of the threats;
  - *target*: item representing the asset toward which the attack is brought;
  - *effect*: percentage of the value of the asset damaged by a successful threat.
- *Impairment* specializes Threats by adding some properties:
  - *latency*: that is the latency of the propagation of the Impairment to other affected Impairments;
  - for each propagation we have (as three arrays):
    - *propEffect*: affected Impairment;
    - *propProb*: probability of having a propagation on the affected Impairment;
    - *propCond*: condition under which we have the propagation of the Impairment.

The *CIP_VAM UML extensions*.

The *CIP_VAM UML extensions* represent the subset of the *CIP_VAM* profile that defines the stereotypes, the tagged values and their constraints. This package import the complete CIP_VAM Library, using the namespace defined by the latter as types for some properties.

The *CIP_VAM UML extensions* are organized according to the domain model structure (Figure 5.3). An overview of the *CIP_VAM UML extensions* is shown in Figure 5.9. Transforming a domain model into a UML profile is not an automatic task. First, the domain model concepts have been classified according to the taxonomy expressed in [65]: abstract, firm, uncertain or parametric[1]. The firm classes are directly mapped into stereotypes while ad-hoc solutions are adopted for the other classes. In a second stage, for each stereotype obtained from a firm class the actual UML meta-classes extended by the stereotype is identified. Such activity is performed both by imaging which UML construct a modeler would choose and by looking at similar UML profiles. As an example, *Object* and *Service* extend UML Classifier, since it is reasonable that an object can be represented by a UML Component or a Class; in addition, *Service* extends the UML Use Case.

Some optimizations are now possible: (1) abstract stereotypes are used to group common features. An example is the *Item* stereotype of the Infrastructure package that has been

[1]Abstract classes refer to accessory concepts, firm classes are used as language constructs, uncertain classes sort indeterminate concepts, and parametric classes categorize concepts that can change depending on the sub-problem domain.

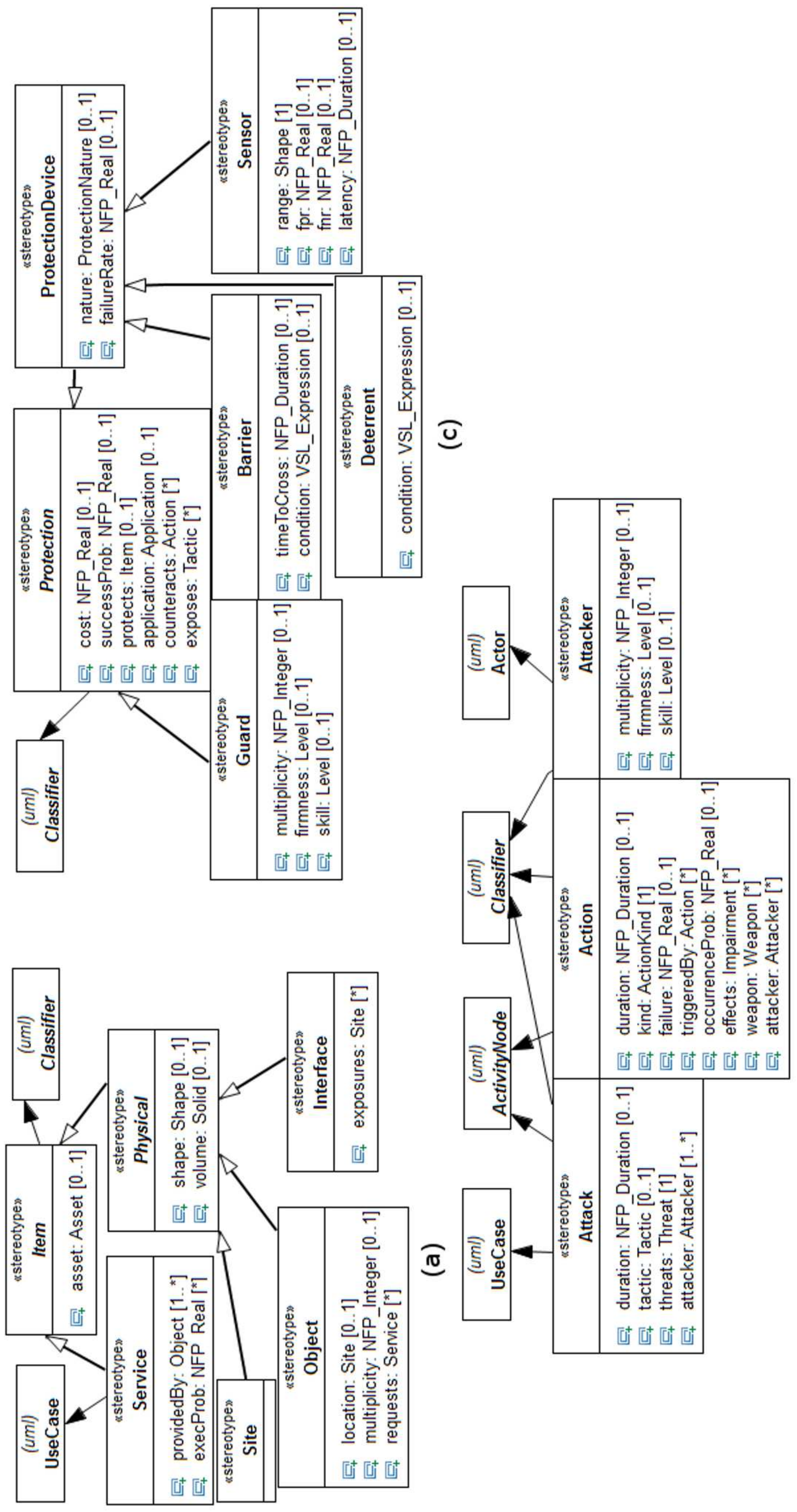


FIGURE 5.9: Overview of the UML extensions: (a) Infrastructure package, (b) Attack package, (c) Protection package

introduced in order to not replicate the *Asset* tagged-value in all the concrete stereotypes of the package: however no UML Classifier can be tagged with the *Item* stereotype; (2) optimization patterns are used, such as the hiding of existing concepts: a class of the domain model can be translated into a datatype (this is the case of *Asset*) and an "is-a" relationships can collapse into just one stereotype to which a "kind" attribute is added belonging to an enumerative type. A example of this last case is the hierarchy *Action* and *Trigger* of the *Attack* package that is translated into the *Action* stereotype and its *ActionType* tagged-value which can be set to *action* or *trigger*. Notice that some relationships included in the domain model have not been translated into profile elements since we choose to rely on the existing UML notation when this is appropriate, as prescribed in [45]. An example of this case is the *subService* association that is implemented by means of the ≪include≫ relationship between use cases and the *owner* attribute of Classifier.

Before the description of the profile levels, some considerations are due in order to define how some concepts of the domain model have been not directly covered by stereotypes:

- subSites, subObjects, subServices, subAttacks: these associations are implemented by exploiting the including mechanisms that are present in UML such as nested classifiers (for subSites and subObjects) and inclusion of Use Cases (for subServices and subAttacks);
- the association between Action and Attack: it can be implemented by nesting classifiers and/or UML Actions inside a Use Case;
- value_site, value_object, value_service, value_interface: all these associations are implemented by the asset tagged value in the Item stereotype.

Infrastructure. This package implements concepts of the Infrastructure domain model. Figure 5.9(a) depicts the UML extension structure for Infrastructure related information. while detailed information are in Table 5.2.

Attack. This package implements concepts of the Attack domain model. Figure 5.9(b) depicts the structure of the UML extensions for the Attack package providing general information while more details are in Table 5.3.

Protection. This package implements concepts of the Protection domain model. Figure 5.9(c) depicts the structure of the UML extensions for the Protection package providing general information while more details are in Table 5.4.

TABLE 5.2: Details of the UML extensions for the Infrastructure package

Item	Item is a generalization of all the main concepts of the Infrastructure domain model inserted in order to optimize the profile. There is no direct mapping from this stereotype and a domain model concept (thus its abstract nature).
extensions	Classifier
generalizations	None
tagged-values	
Asset	Optional asset variable indicating the economic value of the item, its risk and other properties according to the Asset data type.
Service	Service implements the correspondent domain concept.
extensions	Classifier (via Item), Use Case
generalizations	Item
tagged-values	
ProvidedBy	It implements the provides association between a Service and an Object which offers the Service.
ExecProb	The probability to be executed by external agents. It implements the homonymous attribute of the domain model class.
Physical	Physical stands for all the Items that are characterized by physical properties such as shape and location. All the Items except Services are Physical. Physical is an abstract stereotype, there is no related domain concept.
extensions	Classifier (via Item)
generalizations	Item
tagged-values	
Shape	2D shape of the Physical element. It implements the homonymous attribute of the domain model classes Site, Object and Interface.
Volume	3D shape of the Physical element. It implements the homonymous attribute of the domain model classes Site, Object and Interface.
Site	Site implements the correspondent domain concept.
extensions	Classifier (via Physical)
generalizations	Physical
Object	Object implements the correspondent domain concept.
extensions	Classifier (via Physical)
generalizations	Physical
tagged-values	
Multiplicity	It describes the number of the Objects of a specified kind.
Location	It implements the location association between an Object and a Site which contains the Object.
Requests	It implements the requests association between an Object and a Service which is requested by the Object.
Interface	Interface implements the correspondent domain concept.
extensions	Classifier (via Physical)
generalizations	Physical
tagged-values	
Exposures	It implements the exposures association between a Site and an Interface which is attached to the Site.

Table 5.3: Details of the UML extensions for the Attack package

Attack	Attack is a stereotype that implements the homonymous concept of the related domain class.
extensions	Use Case, Activity Node, Classifier
generalizations	None
tagged-values	
Duration	Time elapsed between the start and the end of the Attack. It implements the homonymous attribute of the related domain concept.
Tactic	Tactic of the Attack.
Threat	Threat brought to related Asset. It implements the Threat association class between Attack and Asset.
Attacker	Data related to the Attacker that initiates the Attack. It implements the Attacker associated to the Attack.
Action	Action is a stereotype that implements the homonymous concept of the related domain class.
extensions	Activity Node, Classifier
generalizations	None
tagged-values	
Duration	Time elapsed between the start and the end of the Action.
Kind	Tagged value that implements the Trigger generalization.
Effects	List of the Impairments caused by the Action.
Failure	Percentage of failure of the Action.
TriggeredBy	It implements the triggeredBy association between Trigger and Action.
OccurrenceProb	Occurrence probability of the Trigger.
Weapon	The Weapon that is used to bring on the action by the attacker.
Attacker	Data related to the Attacker that initiates the Action.
Attacker	This stereotype implements the Attacker concept of the domain class.
extensions	Actor, Classifier
generalizations	None
tagged-values	
Multiplicity	Number of Attackers.
Motivation	Capability of the Attackers to resist to dissuasion actions performed by defence systems.
Firmness	Capability to accomplish at the attacking Action.

Table 5.4: Details of the UML extensions for the Protection package

Protection	Protection is an abstract stereotype that summarizes all the extension points and tagged-values common to all the concepts for the Protection domain class. It implements the Protection class of the domain model.
extensions	Classifier
generalizations	None
tagged-values	
Cost	Cost of the Protection system.
SuccessProb	Success probability of the Protection system.
Protects	It implements the protection association between Protection and Asset.
Counteracts	It implements the counteraction association between Protection and Action.
Exposes	It implements the expose association between Protection and Weapon.
Application	It implements the association between Protection and Installation-Point.
Guard	Guard is a stereotype that implements the homonymous concept of the related domain class.
extensions	Classifier (via Protection)
generalizations	Protection
tagged-values	
Multiplicity	Number of Guards.
Firmness	Capability of the Guard to resist to attacking Action.
Skill	Capability to accomplish at the protecting Action.
Protection Device	ProtectionDevice implements the Device class in the domain model.
extensions	Classifier (via Protection)
generalizations	Protection
tagged-values	
Nature	Technical nature of the ProtectionDevice system.
FailureRate	Failure rate of the ProtectionDevice system.

Sensor	Sensor implements the homonymous class in the domain model.
extensions	Classifier (via ProtectionDevice)
generalizations	ProtectionDevice
tagged-values	
Range	Action range of the Sensor.
Fpr	False positive ratio; percentage of false alarms generate by the Sensor.
Fnr	False negative ratio; percentage of malicious Actions that have not been sensed.
Latency	Time elapsed between the start of an Action and the detection by the Sensor.
Deterrent	The Deterrent stereotype implements the homonymous domain model class.
extensions	Classifier (via ProtectionDevice)
generalizations	ProtectionDevice
tagged-values	
Condition	Condition for which the Deterrent is effective for blocking the Attacker.
Barrier	The Barrier stereotype implements the homonymous domain model class.
extensions	Classifier (via ProtectionDevice)
generalizations	ProtectionDevice
tagged-values	
Condition	Condition for which the Barrier is effective for the Attacker.
TimeToCross	Time necessary to penetrate the Barrier.

Chapter 6

New methods, formal operators and techniques

The formal modeling of critical systems (*formal stage* of the approach, presented in Section 4.1) requires the usage of structured approaches able to support and facilitate the model development. A modeling approach could be based on the OsMoSys (Object-baSed multi-formalism MOdeling of SYStems) modeling methodology [16], which adopts a component-based vision of the systems. The vision underlying OsMoSys is to apply the concepts of component software engineering to the development of formal models in order to provide a practical support to model engineering. According to OsMoSys methodology, a model is an object of a *Model Class* which encapsulates the details of its implementation (the model structure is expressed by a formal language). Hence, models are assumed as components which communicate via interfaces. An interface is a subset of elements of the model structure which may be used to exchange information among models (such as values of parameters, indexes or the overall state of a model). The role of the interfaces has already been addressed in [66, 67].

The OsMoSys methodology may be very effective in modeling critical systems, since the component-based approach emphasizes the separation of concerns. Nevertheless, the modeling of critical systems is characterized by a variable number of replicated objects, as well as by the heterogeneity in physical characteristics and behaviors of the involved components. Hence we need to extend OsMoSys by defining new formal operators which easily provide a specification for generating models on the basis of a set of parameters. Furthermore we need to introduce new modeling techniques which help the modelers during the development of models.

This Chapter introduces new methods, operators and techniques to support the development of formal models. These innovations will be applied in Part III to support the quantitative analysis of real-world systems, during the *formal stage*. In detail a new component-based architecture for performability and QoS evaluation is proposed: this architecture provides a structured approach to formal modeling of critical systems, integrating behavioral, environmental and dependability-related aspects. Then the new *Model Template* operator is introduced: this formal operator helps the reuse, increasing the abstraction level given by the *Model-Class* in OsMoSys. In the last part of this Chapter two effective techniques are presented to cope with model scalability: *Stub Models* and *Reduced Models*. These techniques allow to significantly reduce the number of submodels, properly substituting them with "equivalent" models. These techniques provide similar advantages of the analogous ones applied in software development. They can be adopted by modelers during model development (to validate a single submodel) or during their analysis (to reduce solving times).

6.1 Performability and QoS quantitative evaluation: a component - based architecture

Performability and Quality of Service (QoS) certainly represent the most important issues that critical system designers have to consider. In fact only the achievement of acceptable performability and quality of service levels will let these systems to be competitive and dependable. These systems, in fact, need to assure an high-level of fault tolerance to fulfill the requested level of service, under the defined environmental conditions and also in presence of failures. While QoS can be influenced by users qualitative evaluation, the Performability is often expressed through mathematical formal, known as Service Availability (SA) indexes. As it is evident, SA is strongly related to the dependability of components, since their failure will tend to reduce system functionalities and, consequently, its operational availability. SA is usually represented by mathematical indexes often described by the ratio between performed (actual) and target (designed) service. In particular such indexes are defined and specified within contracts between customers and systems designers or manufacturers. These indexed are strictly dependent on the reference application domain: a common SA index adopted in railway domain is, for example, the *punctuality* defined as the ratio between the number of on-time trips and the total number of trips arrived at a certain station. In particular punctuality can be expressed as: $punctuality = \frac{(t_s - t_l)}{t_s} \cdot 100$ where t_s is the number of scheduled trips within a certain time period and t_l is the number of lost trips (i.e. the number of trips which does not arrive at the considered station) calculated over the same time interval.

Anyway in order to guarantee acceptable SA levels, high-dependable components (i.e. high reliable and maintainable items) are commonly adopted. This is partially true since the SA is highly influenced by performed operational strategies: in a critical system different operational strategies can be applied with the aim to minimize failures effects on SA.

Formal models can be adopted to perform an accurate assessment of SA levels and quantitative aspects of QoS, induced by system design and operational alternatives; they need to consider the overall interactions among different items composing a system, specific operational strategies, failure probabilities and the users' behavior. For this reasons this Section proposes an integrated architecture to model and analyze system operations and to assess consequently both SA and QoS levels. In particular, the proposed architecture is suitable for simulative approaches. Outputs returned by such model architecture can be analyzed with three different goals:

1. they assess the impact of disturbances on service estimating the variation induced on SA and QoS;
2. they estimate the efficiency of different alternative system configurations, allowing to choose the one that has the greater impact on cost reduction;
3. they are able to support the identification of the optimal configuration of the plants and of operation and maintenance staff.

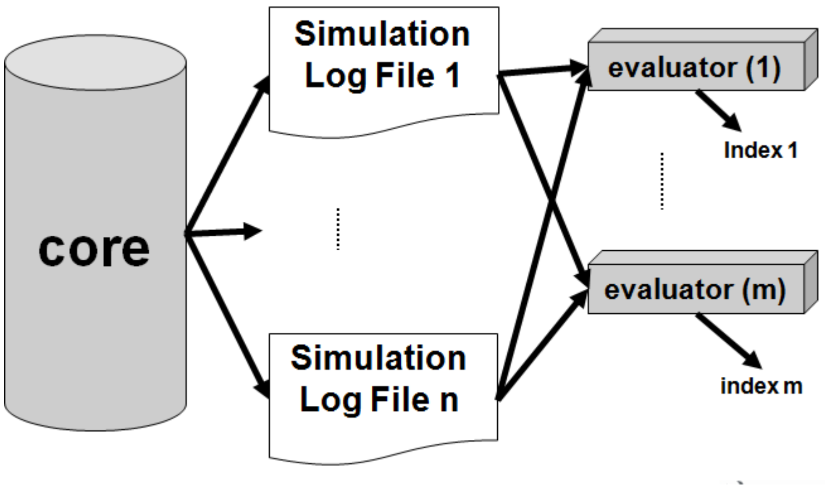


FIGURE 6.1: Overview of the model architecture

The necessity to estimate a non-defined (and project dependent) set of SA (and quantitative QoS) indexes makes necessary the decomposition of the architecture in a central model that can be analyzed through customizable evaluators, each one is able to measure a specific parameter. The explicit architecture is depicted in Figure 6.1: the *core* component is the kernel of the architecture and it reproduces system operational conditions service taking into account any kind of stochastic disturbance on service; the

Simulation Log Files developed in output, in a very large number, contain all relevant information of a single simulation (e.g. the list of all instant of arrival and departure of each train at each station); a group of *evaluator* make use of *Simulation Log Files* to evaluate interest parameters.

The proposed model architecture needs to encompass several characteristics of a critical system concerning with four different aspects: system service, operational strategies, environmental conditions and state of subsystems (in terms of service modes). For this reason the *core* component is structured as the composition of four cooperating modules, as depicted in Figure 6.2.

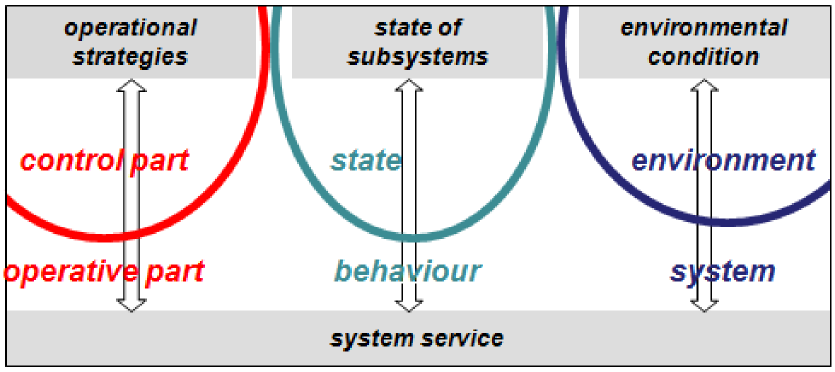


FIGURE 6.2: Internal organization of the *core* component

Each module cooperates with the *system service* one since the latter is able to integrate external influences and controls, modeling their impact on service. In the following paragraphs each module is described.

***Operational strategies* module.**

This module is responsible for the implementation of operational strategies. Operational strategies depend on the specific system and can range from routing rules for network systems to timetables and service management strategies for transportation systems, to control part of embedded systems, etc. The isolation of this module, with respect to the others, adheres to a system decomposition in **control part vs operative part** (classically adopted in embedded systems).

***Environmental conditions* module**. This module is dedicated to the environment modeling, which surrounds and influences the system under analysis. It shall model all the influence aspects, from users' behavior to field data, from malicious users to terroristic attacks. The isolation of this module, with respect to the others, adheres to a system decomposition in **environment vs system**.

***State of subsystems* module.** The state of subsystems module deals with the modeling of operating mode of all subsystems involved in service fulfillment (when performability evaluation is required). In particular, this module contemplates the evolution between different operating modes according to the failure rates and probability of state passing of each failure-prone component. The isolation of this module, with respect to the others, adheres to a system decomposition in **state vs behavior**.

***System service* module.** This module models the system service. Inputs of this module are all physical, mechanical and technological characteristics of the system components, allowing to represent service performed by the system. Such module collaborates with the others, considering the information coming from them to analyze system service.

6.2 Enabling model reuse: the *Model Template* operator

This Section describes the new operator of *Model Template*, a new formal operator which extend OsMoSys with the aim to summarize a set of models sharing a common structure in an unique concept. The most widespread meaning of "template" is that coming from Class Template in programming languages (typically C++); in this context, a Class Template represents a class where some attributes are typed with a generic type T showing a common behavior for all the possible T. Moreover, in this context, it is also possible to parametrize a class template by non-type parameter (usually an integer) that implies the parametrization of data structure belonging the class, according to the value of the parameter.

Starting from these general considerations, *Model Templates* may introduce a powerful feature in formal modeling, they allow to specify, with a single model description, an entire family of related models characterized by a common behavior. Similarly to Class Templates introduced by programming languages, *Model Templates* require one or more *type* parameters to specify how to generate a specific instance from a generic model template. The type parameters refer to the type of one or more elements of the model structure, including the type (i.e., the Model Class) of submodels. It is also possible to use *non-type* parameters, i.e., to specify the number of replicas of a subset of elements (including submodels). The complete definition of *Model Templates* in OsMoSyS is part of an ongoing work. To date we have specified, inside the OsMoSYS methodology, the concept of *Model Templates* with *non-type* parameters; the formalization of *type* parameters is scope of future works.

Some research papers proposing template based approaches have been published, all limited to some extent: in [68] typed parameters template are introduced for Petri Nets models. In [69] the focus shifts onto Stochastic Activity Networks and on the possibility to change the behavior of the model according to *non-type* parameter values. Other works focus on the possibility to both replicate and join submodels inside a compositional approach [70, 71]. To the best of our knowledge, there are no works trying to overcome the limitations of specific formalisms (Petri Nets and SANs), or even to define the "generic" concept of *Model Template*.

In order to define the *Model Template* according to the OsMoSys notation, some preliminary definitions are due. Formally, a *Model Class* $MC_{\mathcal{F}}$ is a triplet (T, S, SM) where: T is the class name, S is the graph structure of the class, SM is a set of submodels. $MC_{\mathcal{F}}$ is compliant with a formalism $\mathcal{F}$ and its structure S is a graph of elements of $\mathcal{F}$. Specifically, from now on we will call N the set of nodes and E the set of edges of S. For example, if a *Model Class* is compliant with the Petri Nets formalism, its structure will comprise $Place$, $Transition$ (the nodes of the graph) and Arc (its edges). It holds: $S = External_S \cup Internal_S$; $External_S \cap Internal_S = \oslash$ where $External_S$ is the subset of the *interface* elements of the *Model Class*, while $Internal_S$ is the subset of elements that are encapsulated by the class. Both external and internal elements in S may have a set of attributes denoted P_S. It holds: $P_S = EP_S \cup IP_S$, where EP_S is the set of the attributes that may be set when an object of type N is instantiated, and IP_S is the set of attributes that are statically defined by the class definition.

Let us define a *Model Template* $MT_{\mathcal{F}}$ as a pair $(MC_{\mathcal{F}}, PAR)$ where $MC_{\mathcal{F}}$ is a *Model Class* and PAR is a set of *non-type* parameters:

$$PAR = \ \{p_1, p_2, \dots p_n\}, n \geqslant 1.$$

Definition 6.1. A non-type parameter is a triplet $p_i = (l_i, SS_i, f_i)$, where l_i is the name of the parameter; SS_i is a subgraph of S, $SS_i = (NN_i, EE_i)$ with $NN_i \subseteq N$ and $EE_i = \{e = (m, m_1) \in E | m, m_1 \in NN_i\}$.

A synthetic notation for the Model Template is $MT_{\mathcal{F}} < l_1, l_2, \dots l_n >$.

Definition 6.2. An *instancing function* is $f_i : \mathbb{N} \longrightarrow \mathcal{M}^1_{\mathcal{F}}$ where $\mathcal{M}^1_{\mathcal{F}}$ is the set of the *Model Classes* compliant with the formalism $\mathcal{F}$.

An *instancing function* f_i must specify how the Model Template should be (automatically) instantiated by using the *non-type* parameters. A synthetic notation for an instance is $MT_{\mathcal{F}} < v_1, v_2, \dots v_n >$ where v_i is the value on which f_i is computed.

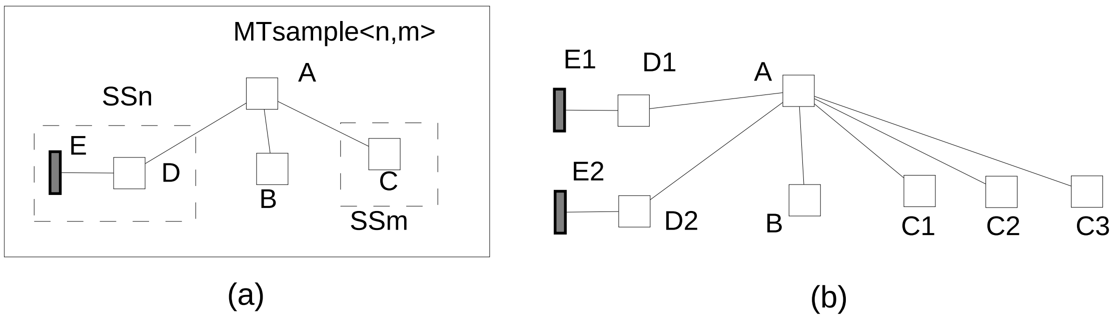


FIGURE 6.3: A Model Template example

An example is depicted in Figure 6.3, in which a simple case of replication is shown.

A Model Template is on the left (Figure 6.3(a)), let it be:

$$MT_{sample} = (MC_{sample}, \{p_n, p_m\})$$

where: $p_n = (n, SS_n, f_n)$ and $p_m = (m, SS_m, f_m)$ are the *non-type* parameters. The subsets of the model structure in the dashed boxes (SS_n and SS_m) may be replicated and a template model may be generated by providing the value of the two non-type parameters. Both subnets include a submodel (the white square). The model template is denoted by $MT_{sample} < n, m >$. $MT_{sample} < 2, 3 >$ is shown in Figure 6.3(b), where the model is obtained by specifying the values 2 and 3 for the number of replicas of SS_n and SS_m, respectively[1]. In this example we use a simple instancing function that copies a sub-graph as many times as indicated in the parameter value: all the replicas are then connected to the rest of the model by replicating arcs connecting the sub-graph with the rest of the structure, too. In the example, the subgraph SS_n and the arc from A to D are replicated twice. Of course, a renaming function is also needed to avoid conflicting names.

The *instancing functions* must be provided in order to state how the template models have to be generated. In the example a simple instancing function is used which allows for replication of a part (or the whole) of the model structure. It is defined by induction as follows.

We denote as $MT_{\mathcal{F}} < x >$ a Model Template with a parameter (x, SS_x, f) where $SS_x = (NN_x, EE_x)$. Let $\overline{EE}_x$ be the subset of E_x connecting nodes of NN_x and of $N - NN_x$.

- $S_1 = S$: the function does not change anything in the structure of the Model Class;

[1]Note that E is an interface element: this approach can be used to replicate model interfaces, too.

- $\forall v \geq 2$, let $f(v-1) = (T_{v-1}, SS_{v-1}, SM_{v-1})$ and $f(v) = (T_v, SS_v, SM_v)$:
    1. T_v is calculated by a renaming function;
    2. $NN_v = NN_{v-1} \cup NN_x$ and $EE_v = EE_{v-1} \cup EE_x \cup \overline{EE}_x$, i.e., the graph generated at v is built by joining the one generated at $v-1$ with a new replica of SS_x and the arcs connecting the new replica with the rest of the structure (not included in SS_x);
    3. $SM_v = SM$

Notice that it might hold that $SS_x = S$, thus allowing for the replication of an entire model.

6.3 Dealing with scalability: ***Stub Model*s and *Reduced Model*s**

In many cases the complete model of the entire system under analysis suffers from scalability. The adoption of the OsMoSys approach, that is the decomposition of the system according to a component-based view, does not increase the scalability. This problem can be critical during the solving phase, involving enormous difficulties or not allowing the solution. Especially when the solution process is conducted by simulation, an enormous memory usage can be required by solvers: if we consider a sensor network, for example, the model of the single node must be replicated N times where N represents the number of nodes in the network; and if increase the level of detail, developing M submodels for each node, then the complete model is composed of N*M submodels.

The solution of the model requires greater simulation times and memory space with the increasing of system dimensions: thus call for the introduction of scalable solutions. These problems are due to the increasing number of submodels and the huge use of hard-solving elements (such as the usage of data structures or matrices). A possible solution to deal with memory usage relies on storing data in a database, while keeping in memory just the references to the tuples. This strategy requires formalisms and tools which make possible to access data stored in databases. A formalism suitable for this strategy is the one of Stochastic Activity Networks that, with supported by the Mobius framework, allows to use C++ code in the model (implementing queries in the gate elements), but it is very hard to implement. Moreover, this strategy is not effective enough for our scope because it just helps overcome the memory consumption issue, while leaving unaffected the computation time. Also the concept of *Model Template*, described in the previous Section, helps the modeling phase but not the solution process.

Therefore, we propose two different approaches to cope with scalability issues: (i) stub models, that replace some levels of peer stack with simplified models, and (ii) reduced models, that take the place of the entire peer stack. These two techniques are not mutually exclusive and can be jointly used. They introduce different approximation levels that we have analyzed, by means of simulations, in the case studies: in fact we show obtained results in the Chapter 8, applying both of them, during the quantitative evaluation of a real-world system.

6.3.1 Stub Model

A stub model, as in software development techniques, may simulate the behavior of an existing, more detailed, model or be a temporary substitute for a yet to be developed model, hence reducing the complexity of a complete one. The stub models can be used to perform analysis circumscribed at a set of (and also a single) Model Class by simulating the behavior of the remaining ones.

The use of stubs is applicable in a model with a layered structure: clearly, a stub for a generic model layer must exhibit the same interface of the models it replaces. An example is reported in Figure 6.4, where a stub model substitutes a set of models maintaining compliance with the interfaces of those to which it is connected: M1 and M2. If M1 and M2, for example, represent two node in a network, a stub model can substitute the entire set of network models with a single transition that fires with the transmission delay.

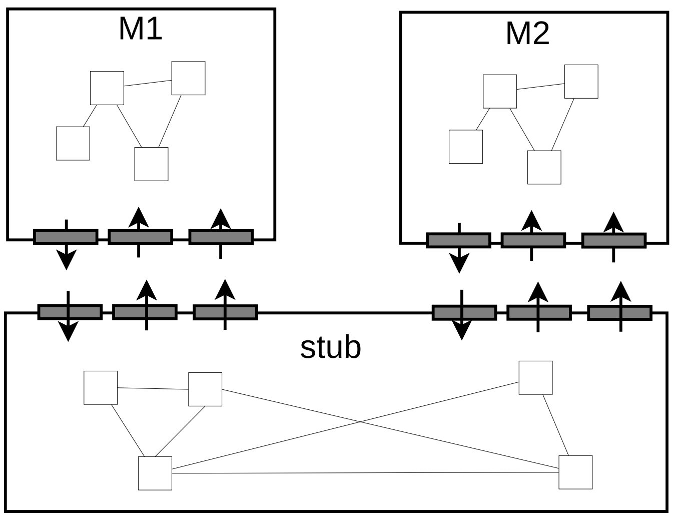


FIGURE 6.4: Stub Model

Stub models, like the complete one, do not prevent from identifying a single element of a replica allowing to assign it a different behavior. They are able to reduce memory occupation for the substituted models; though they still entail a significant computational complexity. The approximation introduced by a stub model is proportional to its ability to represent the missing levels: the greater the accuracy of the model, the lower the error on results.

The adoption of stubs can be useful for two different purposes:

1. to focus on the model of a specific level, in which case a stub acts as a substitute for the models of the lower levels;
2. to build a smaller scale model of the overall conferencing system, by considering just one or few levels both for participants and for the server.

6.3.2 Reduced Model

In many cases the analysis of a critical system with a component-based approach can lead the connection of several submodels. But sometimes the analysts need to have informations about just few part of the complete systems: for this reasons it is possible to simplify several portion of the models, collapsing them in a simple model. Obviously this reduction is possible when the formalism allows a great expressive power. Also in this case the SAN formalism is suitable for this technique. An example, taken from the application to real case study, is reported in Figure 6.5 where a set of clients, in a network model, is modeled by a trivial SAN where the greater part of the behavior is implemented through the input and output gates.

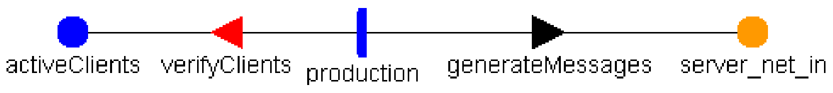


FIGURE 6.5: The client reduced model

Part III

Quantitative analysis of real-world systems

Chapter 7

Railway Systems

Railway system are surely critical systems since they need to assure high-level of dependability against possible failures, to provide high levels of service both for passengers and goods, to reduce the vulnerability of their infrastructures against criminal actions. According to the classification given in Section 1.1, they can be classified as business, safety and security critical systems since the impact of failures can induce negative impacts on all these aspects. Railway systems can be classified also as complex systems, since they require complex interactions between heterogeneous components, both hardware and software, as well as with continue and discrete temporal evolution. This complexity makes the design, construction and maintenance very difficult, in particular with the aim of controlling both the QoS and the performability: it is necessary a continue assessment during all the lifecycle, during also the service phase. From the vulnerability point of view, a railway system need to be equipped with complex protection systems, to avoid criminal attacks and/or to reduce their impact. Also protection systems need to be adequately designed since preliminary phases of the lifecycle, especially in case of limited economic resources: the answer to the question of whether the railways are sufficiently protected is not unique/absolute, but it depends on the relationships that exist among the three main aspects that are the core of the problem: the infrastructure, the protection system and the possible attack scenarios.

The achievement of sufficient QoS levels is influenced by the attainment of acceptable performability levels. More and more often target values are subject of contracts for designer companies, and their attainment is strictly connected with economic bonus (when exceeded) and penalties (when not reached). Furthermore the achievement of good QoS levels increases the system competitiveness in the people transportation domain. This is another hard task due to the presence of several inter-correlated and time-varying phenomena.

Criticality and complexity of such systems has conducted, over years, to the birth of a new engineering: the “railway systems engineering”. This discipline deals with the design, construction and operation of all types of railway systems. It involves the cooperation of a wide range of engineering disciplines, including civil engineering, computer engineering, electrical engineering, mechanical engineering, industrial engineering and production engineering. Also business economy is highly involved since the total costs of a system are very high.

Between railway systems, metro systems are sometimes seen as simplified railways since they are built only for passenger transportation, with reduced lengths and similar vehicles are used; this is not true because high capacities and frequencies are required, stressing the involved components. Failures can have serious consequences since their impact is felt by a lot of trips, and some restore strategies can be difficult to implement (e.g. the recovery of a stopped train through diesel vehicle, after the loss of power supply, requires the removal from the line of a huge number of vehicles).

This Chapter shows how the proposed methodology has been applied successfully to these systems; it is composed of three main parts: in the first part it shows how the proposed approach has been successfully adopted for quantitative availability evaluation of metro subsystems, starting from high-level models developed using the DAM-Rail language. In the second part of this Chapter performability evaluations are shown with the aim to show the potentialities of the approach. In the last part, the application of the same methodological approach for the vulnerability evaluation of railway infrastructure systems is shown: in this case the CIP_VAM language has been used to produce high-level models of security-critical physical infrastructures. The transformations towards formal models for quantitative vulnerability analysis are also addressed.

7.1 *U-Rail*: UML model-driven methodology for non-functional properties analysis in railway domain

U-Rail (Figure 7.1) is a model-driven methodology specific for railway systems, increasing the assessment process efficiency of non-functional properties, allowing the automatic generation of formal models (combinatorial and state-based) and other process artifacts needed in both tender and certification phases. The U-Rail core is the modeling of a railway system (both under structural and functional aspects) using high level modeling languages: DAM-Rail for dependability attributes and CIP_VAM for vulnerability modeling. Since both languages have been created extending UML with profiling technique, they can be used singly or together (in fact a UML model is conformant to one language or to both of them). The combined usage of both language helps the reuse: if we take, as an example, the vehicle, there is a need to evaluate its availability since it has a great impact on the performability of the railway service but it is also a vulnerable infrastructure.

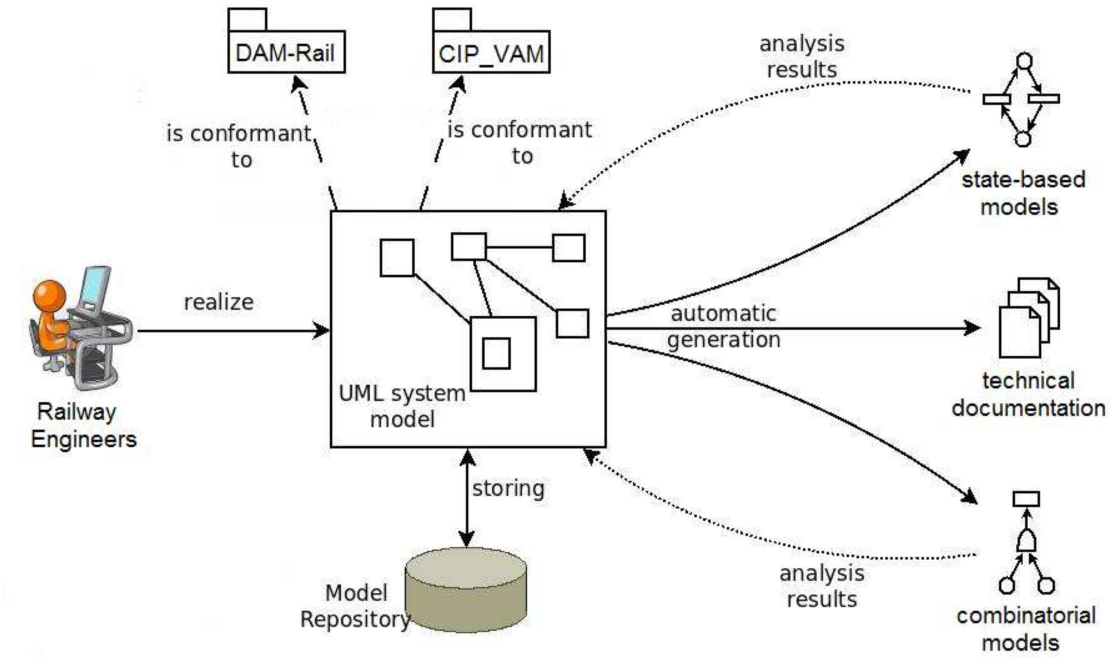


FIGURE 7.1: U-Rail methodology

Following the stages of the general approach, the first step of U-Rail is the representation of the system by the realization of a UML system model that captures dependability and vulnerability aspects of the system itself. The conformity to the DAM-Rail language allows the highly-fashion annotation of several RAM features such as:

- components, redundancy, part-of relationship;
- RAM targets, function dependency;
- failure rates and modes;
- O&M strategies, service restore and component repair procedures;
- headway, track features, speed profiles.

The conformity to the CIP_VAM language allows the highly-fashion annotation of several vulnerability features such as:

- economic values of critical assets;
- vulnerability targets;

- attacks and threat propagations;
- technical features of protection systems.

Automatic transformations are then defined to generate both (multi-)formal models and technical documents. Formal models can be used to assess quantitative properties while technical documentation describing the system can be automatically updated. Model transformations can also feedback the results of analyses to railway engineers integrating them into the UML model; hence the results are visible into the high-level model that acts also as the interface between engineers and formal models.

The third pillar of U-Rail is constituted by a model repository where existing UML models of both entire systems and their subsystems can be stored. This allows to:

- trace model changes during the system evolution;
- reuse assessed UML model where new projects start;
- extract historical data and information about developed projects.

The proposed approach can help the usage of formal models in industry, developing not only a complete model-based approach, but also a model-driven process able to generate automatically formal models. Furthermore the proposed methodology can be combined with standard practices that drive development and verification, so its adoption in industrial processes does not need expensive extra-efforts, but it can support also the automatic generation of the required technical reports in the standard formats.

The adoption of U-Rail methodology may lead to an estimated costs reduction of approximately 30% just for RAM activities. In order to provide an example, on the basis of typical project schedule and scope of work, during the design phase, for RAM activities about 2.000 hours are planned, consequently the reduction is quantifiable in about 600 hours. Considering a average cost of 70 Euros/hour, the reduction is of about 42.000 Euros for every project. The usage of a high-level central language significantly decreases also the training time of a new RAM analyst and the editing time of a new technical document. An improvement in RAM team reactivity during the project lifecycle is also observable. During the Design stage, an approximately estimation of the cost reduction is about of the 20% of the total cost for this stage. Furthermore U-RAM enables the industries to help "non expert" customers in new systems design and in performance requirements definition.

Preliminary applications have been conducted in order to generate Bayesian Network model of a driverless vehicle (Section 7.2), applying the transformations defined in Section 4.4. Future evolutions of the U-Rail are: the development of a second version of

DAM-Rail that will focus on the integration with Stochastic Activity Network models and MetroModelSim (Section 7.3) for performance evaluation.

7.2 Availability modeling of a railway system

The main service provided by a metro system is the *transport* of passengers. This service is evaluated through mathematical expressions, defined by the customer needs and specified in a tender document. As an example, in this case study, we adopt the Operating System Service Availability (A) as a service measure, that is: $0 \leqslant A \leqslant 1$, $A = 1 - \frac{\sum_i d_i}{Nsec}$, where d_i is the delay, recorded in seconds, measured at terminal station at the end of the i-th trip (varying i from 0 to the number of trips performed in a day); and $Nsec$ is the total daily service time expressed in seconds. In this case study let us assume a target value for the A of 97.0%, with a 18 hours daily service and a headway between trips of 360 seconds (180 nominal trips). For the overall system we need to demonstrate that a trip arrives on time with a probability of 98.8% and delayed with a probability of 1.2%, in particular with a probability of 0.6% the delay must be shorter than 3 minutes, with a probability of 0.45% the delay can be longer than 3 minutes but shorter than 10 and with a probability of 0.15% the delay can be longer than 10 minutes (with a maximum of 75 minutes). With the aim of classifying the effects of a failure on the *transport* service, three failure consequence classes (FCCs) are hence defined. Note that these classes, depending on the specific formula, are not common to different projects. The three failure consequence classes here defined are reported in Table 7.1.

TABLE 7.1: Transport: failure consequence classes.

FCC	***Definition***
Negligible (NG)	A failure that causes a delay to service shorter than 3 minutes.
Tolerable (TL)	A failure that causes a delay to service longer than 3 minutes and shorter than 10 minutes.
Intolerable (NTL)	A failure that causes a delay longer than 10 minutes.

The main service *transport* is performed by the cooperation of a set of basic services, offered by the different subsystems (e.g. power supply, signalling and vehicle fleet). Without losing generality, here we suppose that the set of basic services, provided by the vehicle fleet, are *movement*, *conditioning* and *powerCapture*. As for the *transport* service, also for the basic services some failure consequence classes are defined (see Table 7.2).

The vehicle fleet consists of 13 vehicles, ten of them are used for service, one is a hot spare (ready to be used) and two are under maintenance. Each vehicle consists of several components whose failures have an impact on the vehicle availability, and consequently on the system Operation Service. For brevity's sake here we restrict our study to the

Table 7.2: Vehicle services: failure consequence classes.

FCC	***Definition***	***Effects***
	***movement* service**	
Return To Depot (RT)	A failure that causes the return to the depot at the end of the trip without service delays (100% of max speed).	Transport.NG
Partial Block (PB)	A failure that causes the landing of passengers at the first station and return to the depot at reduced speed (40% of max speed).	Transport.TL
Total Block (TB)	A failure that causes a total block of the vehicle (0% of max speed) and the need for a thrust.	Transport.NTL
	***conditioning* service**	
Acceptable (AC)	A failure affecting one conditioner of a single car (coolness ratio reduced at 60%).	No Effect
Down (DW)	A failure affecting both conditioners of a single car (coolness ratio at 0%).	Transport.NG
	***power capture* service**	
Down (DW)	A failure where vehicle is not turned on by power supply	Movement.TB Conditioning.DW

components and failure modes listed in Table 7.3 which reports just a fragment of the FMEA (Failure Modes and Effects Analysis) document of the vehicle subsystem.

Table 7.3: Vehicle FMEA Fragment.

Level	**Component**	**Failure Mode**	**Failure rate** $[h^{-1}]$	**num / tot**	**Effects**
1	Electric Unit	-	-	-/1	-
2	Pantograph Unit	-	-	-/2	-
3	Pantograph	Loss of contact	2.5E-6	1/2	PowerCapture.NM
				2/2	PowerCapture.DW
1	Traction Unit	-	-	-/1	-
2	Engine Unit	-	-	-/2	-
3	Engine	Short circuit	8.3E-6	1/2	Movement.PB
				2/2	Movement.TB
		Mechanical breakdown	1.2E-5	1/2	Movement.PB
				2/2	Movement.TB
1	Undercarriage	-	-	-/1	-
2	Wheel Unit	-	-	-/12	-
3	Wheel	Crack	2.5E-7	1/2	Movement.PB
				2/2	Movement.TB
		Border breakdown	2.5E-7	1/2	Movement.PB
				2/2	Movement.TB
1	Air Conditioning	-	-	-/1	-
2	Air Unit	-	-	-/3	-
3	Air Compressor	Electrical	5E-7	1/2	Conditioning.AC
				2/2	Conditioning.DW
		Over Temperature	2.5E-7	1/2	Conditioning.AC
				2/2	Conditioning.DW
	Gas Tank	Pressure Loss	2.7E-8	1/1	Conditioning.DW

7.2.1 Vehicle DAM-Rail model

Modelling this case study requires the application of stereotypes coming from DAM and DAM-Rail to annotate service modes and vehicle component features. The Use Case Diagram representing services (stereotyped as *DAM::DaService*) is shown in Figure 7.2. The actor *user* of the metro system is associated with the *transport* service, which is the only service perceived by him. Indeed, as said before, the *transport* service strictly depends on the vehicle *movement* and vehicle *conditioning* services, which in turn depend on the *powerCapture* service. The tagged value *usedResources* denotes, for each service, which components provide that service. The *transport* does not specify this tagged value because this service is provided by the cooperation of the basic ones.

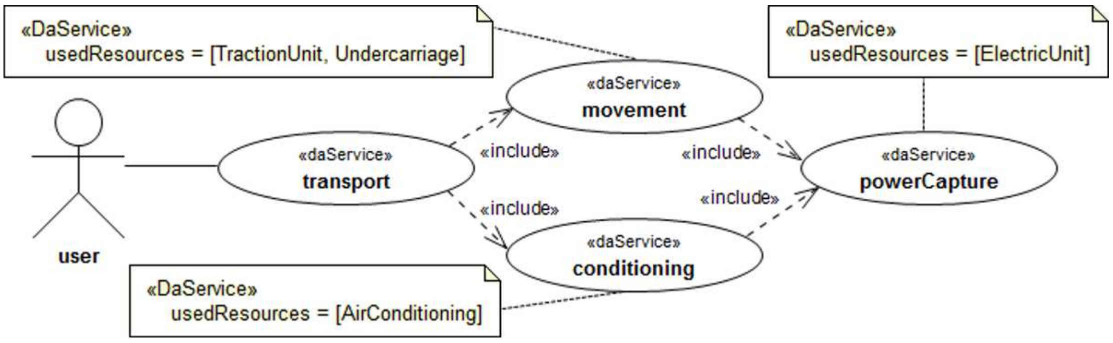


Figure 7.2: Use Case of Services.

Each use case is associated to a State Machine Diagram, which models the state passing between the different service modes. The service modes expected for a single service correspond to the failure consequence classes of the service, i.e. each state depends on the consequences of a failure, hence the nominal state represents the service mode free of failures. As an example, the top of Figure 7.3 depicts the State Machine Diagram of the *transport* service: the four states model the *Nominal (NM)* service mode and the other three service modes saying the degradation level of the service. Similarly, the bottom of the Figure 7.3 depicts the State Machine Diagram for the *conditioning* service. The *entry action* of the latter diagram expresses the effects of the vehicle *conditioning* service degradations on the *transport* service, according to Table 7.2; in the same diagram, the states representing degraded service modes need to specify the tagged values *failure* to express the relationships between the failure occurrences and the state passing.

The Component Diagram in Figure 7.4 provides the description of the vehicle structure previously described in Table 7.3.

The slice of the complete Component Diagram, which refers to the *AirConditioning* subsystem, is reported in Figure 7.5. The slice points out the tagged values. At the

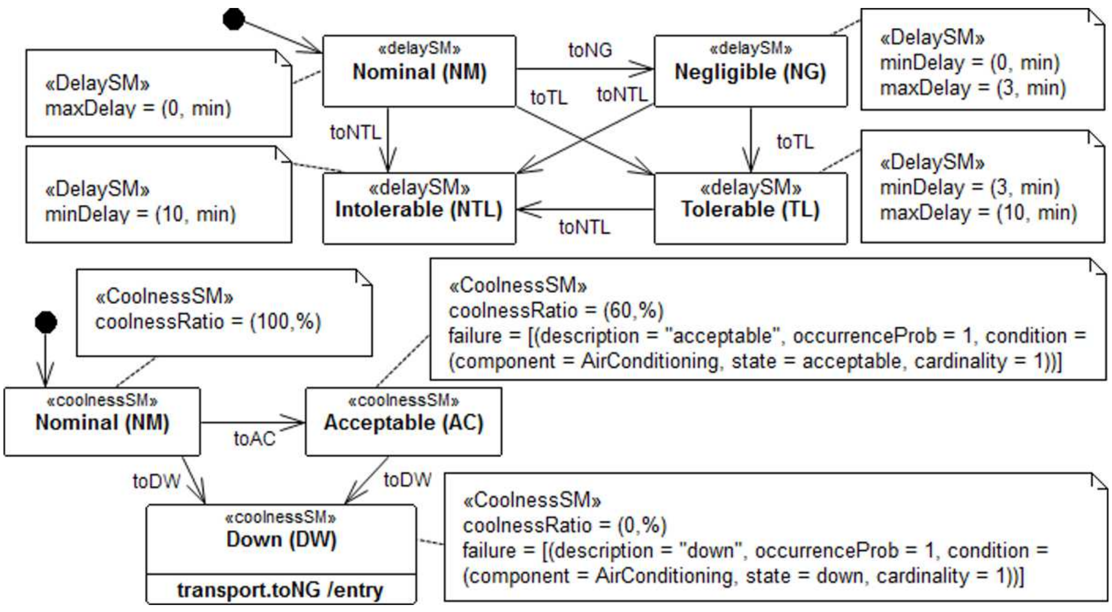

«delaySM» Nominal (NM)
«delaySM» Negligible (NG)
«delaySM» Intolerable (NTL)
«delaySM» Tolerable (TL)
toNG
toTL
toNTL
«DelaySM» maxDelay = (0, min)
«DelaySM» minDelay = (10, min)
«DelaySM» minDelay = (0, min) maxDelay = (3, min)
«DelaySM» minDelay = (3, min) maxDelay = (10, min)
«CoolnessSM» coolnessRatio = (100,%)
«CoolnessSM» coolnessRatio = (60,%) failure = [(description = "acceptable", occurrenceProb = 1, condition = (component = AirConditioning, state = acceptable, cardinality = 1))]
«coolnessSM» Nominal (NM)
«coolnessSM» Acceptable (AC)
«coolnessSM» Down (DW)
transport.toNG /entry
toAC
toDW
«CoolnessSM» coolnessRatio = (0,%) failure = [(description = "down", occurrenceProb = 1, condition = (component = AirConditioning, state = down, cardinality = 1))]

FIGURE 7.3: *Transport* and *Conditioning* State Machine Diagrams.

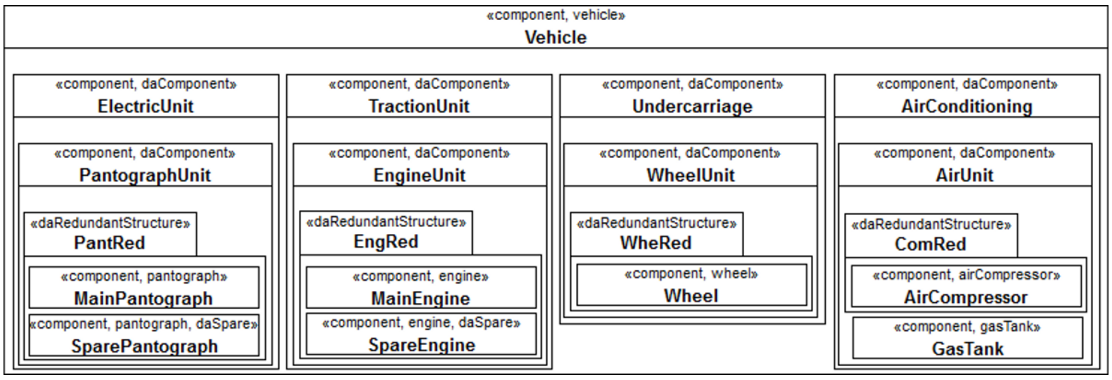

«component, vehicle» Vehicle
«component, daComponent» ElectricUnit
«component, daComponent» PantographUnit
«daRedundantStructure» PantRed
«component, pantograph» MainPantograph
«component, pantograph, daSpare» SparePantograph
«component, daComponent» TractionUnit
«component, daComponent» EngineUnit
«daRedundantStructure» EngRed
«component, engine» MainEngine
«component, engine, daSpare» SpareEngine
«component, daComponent» Undercarriage
«component, daComponent» WheelUnit
«daRedundantStructure» WheRed
«component, wheel» Wheel
«component, daComponent» AirConditioning
«component, daComponent» AirUnit
«daRedundantStructure» ComRed
«component, airCompressor» AirCompressor
«component, gasTank» GasTank

FIGURE 7.4: Driverless vehicle Component Diagram.

lowest level the basic components are *GasTank* and *AirCompressor*: some tags, in particular failures modes, including their occurrence rate, have been annotated for both these components. The *AirCompressor* is enclosed in the *ComRed* redundant structure. Leveling up, at the *AirUnit* level, two failures have been annotated. Notice that the dependency on the failures of the basic components is modelled by a "condition" expression in the DAM Backus-Naur Form (BNF) syntax. At last, *AirConditioning*, which is the *usedResource* of the *conditioning* service (Figure 7.2), has been annotated with two failures which reference to the state of the air conditioning unit to the *conditioning* service, provided by itself.

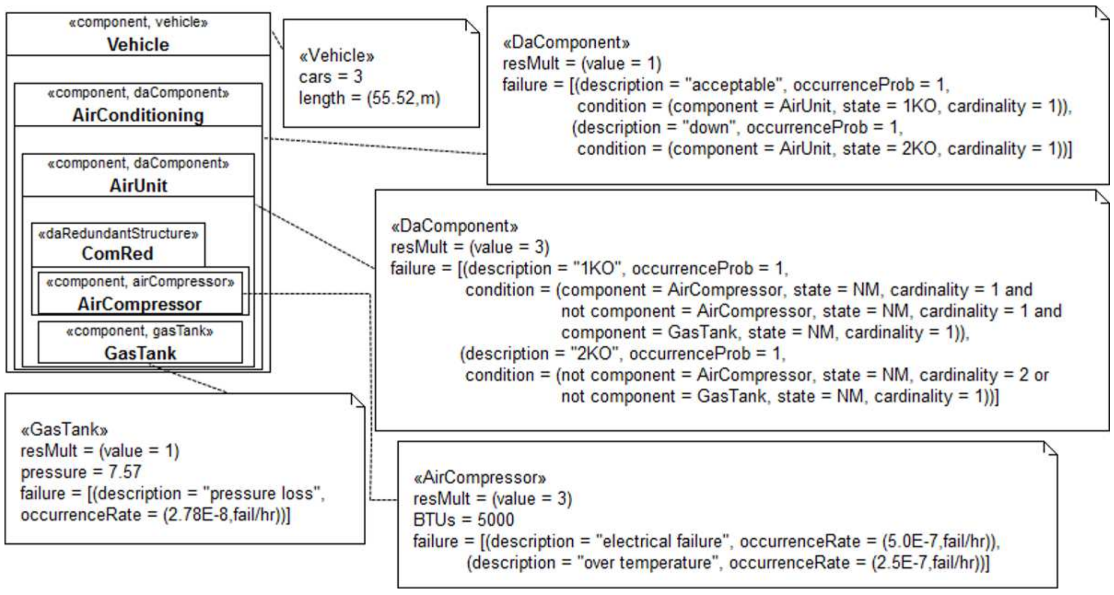

«component, vehicle»
Vehicle
«component, daComponent»
AirConditioning
«component, daComponent»
AirUnit
«daRedundantStructure»
ComRed
«component, airCompressor»
AirCompressor
«component, gasTank»
GasTank
«Vehicle»
cars = 3
length = (55.52,m)
«DaComponent»
resMult = (value = 1)
failure = [(description = "acceptable", occurrenceProb = 1,
condition = (component = AirUnit, state = 1KO, cardinality = 1)),
(description = "down", occurrenceProb = 1,
condition = (component = AirUnit, state = 2KO, cardinality = 1))]
«DaComponent»
resMult = (value = 3)
failure = [(description = "1KO", occurrenceProb = 1,
condition = (component = AirCompressor, state = NM, cardinality = 1 and
not component = AirCompressor, state = NM, cardinality = 1 and
component = GasTank, state = NM, cardinality = 1)),
(description = "2KO", occurrenceProb = 1,
condition = (not component = AirCompressor, state = NM, cardinality = 2 or
not component = GasTank, state = NM, cardinality = 1))]
«GasTank»
resMult = (value = 1)
pressure = 7.57
failure = [(description = "pressure loss",
occurrenceRate = (2.78E-8,fail/hr))]
«AirCompressor»
resMult = (value = 3)
BTUs = 5000
failure = [(description = "electrical failure", occurrenceRate = (5.0E-7,fail/hr)),
(description = "over temperature", occurrenceRate = (2.5E-7,fail/hr))]

FIGURE 7.5: Air conditioning.

7.2.2 Applying DAMRail2BN transformation

Figure 7.6 depicts the application of *dam2bn* to the *AirConditioning* component of the vehicle.

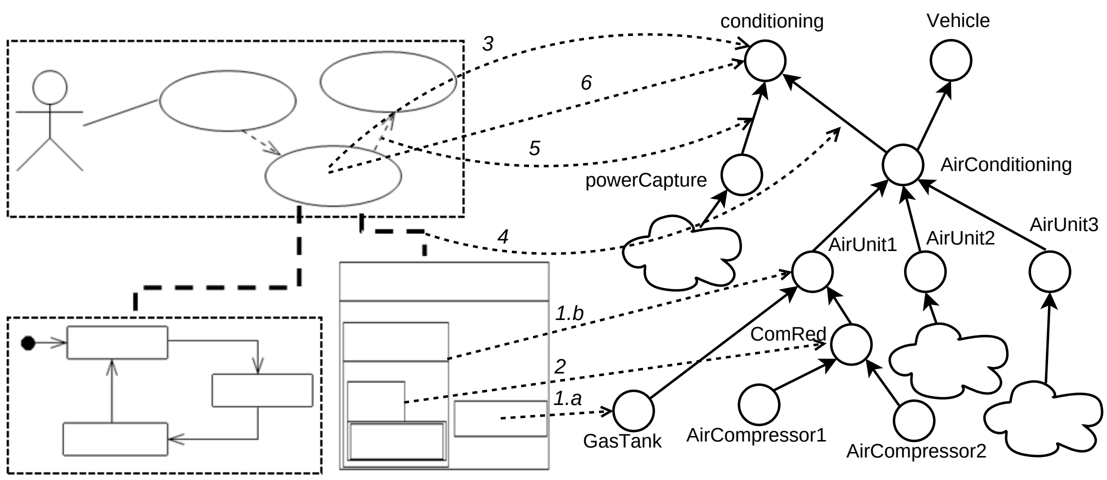


FIGURE 7.6: DAM-to-BN transformation steps.

It works according to the rules described in Section 4.4: Rules 1 and 2 are applied to the Component Diagram in Figure 7.4, Rules from 3 to 5 are applied to the Use Case Diagram in Figure 7.2, Rule 6 is applied to the State Machine Diagram of the service in Figure 7.3. For each basic component, Rule 1 generates a number of nodes equal to the value specified by *ResMult*, hence *GasTank* is translated into one node (dotted line 1.a) and *AirCompressor* generates two nodes. According to the *failure* tagged value, the random variable values associated to the *GasTank* node are two (nominal service mode and pressure loss), while the random variable values associated to the *AirCompressor* node are three (nominal service mode, electrical failure and over temperature). Rules 1 and 2 also generate the *AirUnit* nodes: the value 3 specified in the tagged value *ResMult* replicates three times the structure, one for each *AirUnit* (dotted line 1.b). Rule 1 and 2 also generate the arcs between nodes. Similarly the nodes *AirConditioning* and *Vehicle* and their related arcs are generated from the Component Diagram. The *usedResources* tagged value of the *conditioning* service is used to generate the node conditioning and the link from the *AirConditioning* node to the *conditioning* node. Through the application of the Rules from 3 to 5, the same node also takes a contribution from the *powerCapture* node, according to the relationships between services. Analogously *dam2bn* works to generate the BN model of the movement service. Finally the CPTs are generated according to the application of the Rule 6. The entire Bayesian Network is created for the service *transport*, on which, in a second step, the M2T transformation has been applied in order to generate the concrete file for the JavaBayes tool. Analysis results, performed with the JavaBayes tool, show how the vehicle dependability is compliant to the foreseen targets.

7.3 *MetroModelSim*: performability and QoS quantitative analysis of metro systems

QoS offered to customers hence represents the main purpose for which a metro System is built and it needs to be considered by operators [72]. The QoS is a measure of the "customer satisfaction", which involves other different issues such as: the tickets cost, waiting time at stations, total on-board travel time, and quality parameters like cleanness, safety level, travel time variability and so on. According to classical literature in the field of transportation engineering [73], customers' satisfaction is maximized when the so called "user's generalized cost" is minimized, where the independent variables are bounded among a certain set of discrete alternatives. Specifically for a single passenger, the generalized cost (C_i), choosing an alternative i, can be expressed as a linear combination of the K attributes ($X_{K,i}$) concerning that alternative, weighted by their respective homogenization coefficients $\beta_{K,i}$, which mostly represent specific costs of the attribute: $C_i = \sum_K \beta_{K,i} \cdot X_{K,i}$. As previously mentioned, for a single customer attributes $X_{K,i}$ can represent both quantitative variables as well as qualitative variables correctly discretized.

The perceived QoS is strongly dependent on performability levels delivered by the system. This measure does not concern just with "the ability to perform", but it would estimate "the ability to maintain performance, also degraded, in presence of failures", properly combining System performance with its dependability. Performability indexes are expressed through mathematical formulas referred to the offered service, measuring both the correct and the degraded service carried out by the System; these indexes are defined in tender documents and may vary for the specific project. They are often described by the ratio between performed (actual) and target (designed) service; as an example, a common performability index is given by "punctuality (P)", which can be defined as $P = \left(\frac{t_s - t_l}{t_l}\right) \cdot 100$, where t_s is the number of scheduled trips within a certain time period and t_l is the number of lost and delayed trips (i.e. the number of not-realized trips or trips that arrive over a certain delay threshold at the measurement station) calculated over the same time interval. Performability indexes are commonly known as "RAM indexes" due to the strict relationships with Reliability, Availability, Maintainability features of the dependability.

The European Standard CENELEC EN 50126 (see Section 1.5.1) defines a complete process for the management of RAMS in railway systems. If, on one hand, this norm is well focused on the process, on the other hand it does not impose a specific approach for the quantitative evaluation, leaving wide margin to adopt an appropriate methodology depending on the adopted formulas that, as said previously, are specific for the project.

To date, the performability evaluation of a metro system, during decisional stages, is carried out through analytical approaches at system level, combining results of combinatorial models applied on subcomponents. As already explained in Section 1.4, this is due to the presence of not skilled personnel in the usage of state-based models; this approach, obviously, lead to a poor accuracy of the analysis with respect to the real behaviour, with an enormous underestimation of performability levels that can cause also loss of tenders for new metro system where high performability levels are required. To better clarify which are the industrial assumption, let's make the following example: after a failure stopping a vehicle, the entire system is assumed as down for a long interval of time, and the service during this period is considered completely unperformed. With a more accurate model, it should be possible to model complex dynamics such as the degraded "shuttle service" (a set of trains traveling in a portion of the entire line).

A lot of service models have been presented in the classical literature in transportation domain, commonly classified in three categories according to the level of detail: *macroscopic*, *mesoscopic*, and *microscopic*. As an example, in [74], a model-based approach for large-scale railways, based on a compositional approach, has been exploited; all the basic models are developed using Petri Nets formalism but they are not able to represent failures and to evaluate dependability attributes. To the best of our knowledge, there are not noteworthy experiments oriented to the performability evaluation in metro systems, hence combining service models at system level with dependability models at sub-system level.

7.3.1 Model architecture for performability and QoS evaluation in metro systems

The integrated architecture proposed in Section 6.1 has been adopted. In particular the *core* provides the analysis of daily service, logging important informations (e.g. the list of all instant of arrival and departure of each train at each station) into the simulation log files. The four components of the *core* are particularized into the following (Figure 7.7): *operational strategies*, *state of subsystems*, *passengers travel demand* and *movement on track*.

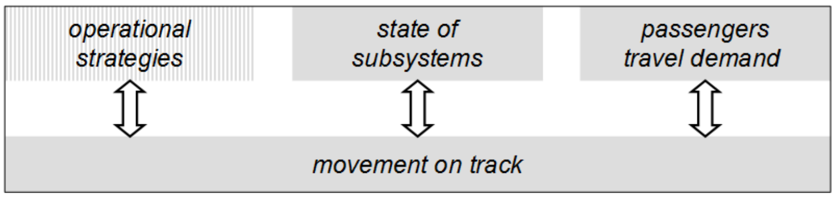


FIGURE 7.7: *Core* component organization

The first module is responsible for the implementation of operational strategies controlled by computer-based control centre. In particular, it is possible to distinguish three main functionalities: the respect of the timetable, the execution of ordinary strategies and the implementation of the recovery strategies, when needed. The module authorizes train departures according to timetable and obviously respecting signalling system aspects which safely regulates train movements on the track. Moreover such module enables the activation of apposite train movements which aim at performing specific operational strategies mostly addressed to restore ordinary service after a component failure.

The *state of subsystems* module deals with the simulation of operating mode of all subsystems involved in service fulfilment. In particular, this module simulates the evolution between different operating modes according to the failure rates and probability of state passing of each failure-prone component (vehicles, on board signalling system, central signalling system, infrastructure, etc.). The model relies on Bayesian Networks, constructed as that reported in Section 7.2 (properly translated into Stochastic Activity Networks through Level 2 M2M transformations), where top nodes represent different operating modes of the component and arcs represent the component failures. Obviously this module allows to simulate the nominal operation, in case, for example, of timetable validation, setting just all the failure probabilities to zero.

The *passengers travel demand* module is dedicated to the simulation of passengers travel demand. In particular passengers origin-destination matrix relative to a certain time period, is considered as input data. Such module is constituted of an assignment model which by means of consistency equations calculates, during simulation, passengers boarding and alighting flows at stations, returning as output the on-board flows for each station and for each train run.

The *Movement on track* module is responsible for simulation of train movements on track. Inputs of this module are all physical and mechanical characteristics of both track and vehicles. Such module assumes the master role during simulation, collaborating with other modules. A classical compositional strategy based on the cooperation of basic elements (e.g. stations, block sections, terminals, pocket tracks, etc.) is used; thanks to well-defined composition rules, these basic elements can be joined together in order to produce the track layout you want to simulate in an enough simple way. To simulate both ordinary and degraded train service, a dynamic integration between different simulation approaches has proved to be an effective solution to overcome the limits of applicability of models at high level of detail, but computationally inefficient (i.e. microscopic), and models unable to describe local and transient train dynamics even though very efficient (i.e. mesoscopic). In fact a combined approach, which dynamically integrates micro- and mesoscopic models allows to efficiently simulate large-scale networks or deal with a large number of simulations (e.g. when performing probability analyses), preserving the accuracy needed to evaluate QoS and SA. In particular, the mesoscopic approach is implemented by means of Stochastic Activity Networks (SAN) formalism and uses timed transitions to model travel times between stations, whilst the microscopic approach, designed in C++, explicitly simulate train dynamics integrating the Newton's motion formula .

7.3.2 Quality of Service and Service Availability evaluation in railway systems

To clearly understand the usefulness of the adopted simulation architecture, it is necessary to implement a practical application on a case study, in particular a Mass Rapid Transit system has been considered. As shown in Figure 7.8, this network is constituted of a 12 km long double-track layout with 15 stations and two terminals to let trains change their path or reverse direction. An ETCS level 1 signalling system type regulates train movements on the track. Furthermore a pocket track to store away corrupted trains is located between station 7 and 8. Scheduled train headway is set to 6 minutes while train dwell times are all equal to 20 seconds for each station. The depot is pinpointed between first and second station, it has three connections with the line:

one is used for entering vehicles in service, another one is used instead to allow the service entrance of hot spare vehicles, and the last one is used for train admission to the depot. Total train running time (including dwell times at stations) is 1251 sec. for 1-15 direction and 1257 sec. for 15-1 direction. Minimum train inversion time at is about 20 sec. at terminal 15 and 140 sec. at terminal 1, while the maximum synchronization time awaited at terminals to perform a constant headway is 66 sec.

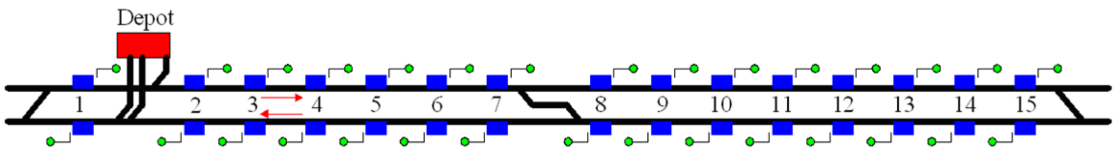


FIGURE 7.8: Schematic layout of the system

Passenger travel demand considered for the line is reported in terms of on-board passengers flows (both for 1-15 and 15-1 direction) referring to a working-day peak-hour in the morning. As can be seen, the maximum passenger flow assessed for 1-15 direction is equal to 8500 pax/h, while for the opposite direction (15-1) this value is 3189 pax/h (Figure 7.9).

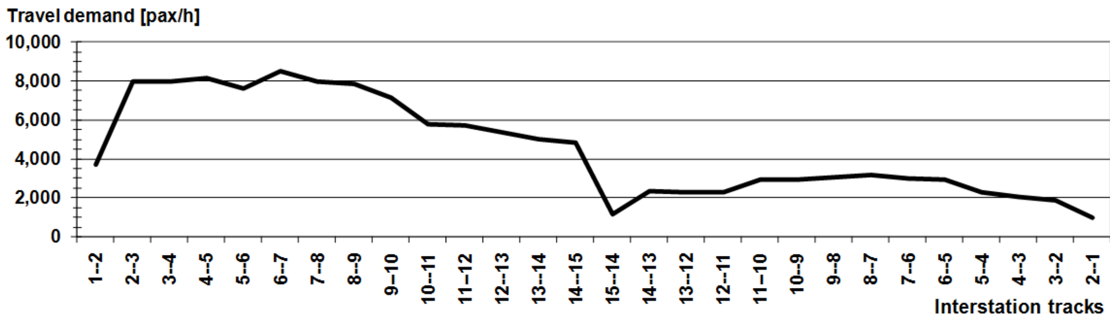


FIGURE 7.9: Passenger travel demand

In this application a total time interval of 3 hours has been observed, considering that hourly passenger flows previously described, preserve exactly the shown trend within each hour of the considered period. According to the scheduled train headway defined by timetable, a total of 30 train runs for each direction have been analyzed. In particular during ordinary service conditions, simulation outputs return total on-board passengers flows for each train run and for each inter-station track (Figure 7.10). As can be seen, each train run shows the same passengers load (in fact on-board passengers diagrams relative to each train run are overlapped).

In particular no limits have been set for train capacity (i.e. the maximum number of passengers that a train can contain). Moreover for the calculation of the users generalized cost the total journey time (i.e. the sum of the on-board travel time and the average waiting time at stations) has been considered as cost function attribute, using a value of the specific cost β of 5euros/h. Within ordinary service the total passengers generalized

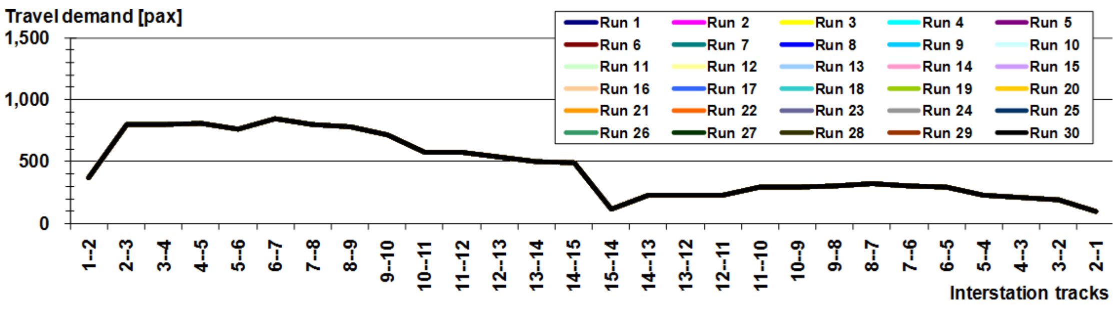


FIGURE 7.10: On-board passengers for each train run, during ordinary service condition

cost estimated over all passengers flows during the entire time interval is 63893 euros. Successively a failure scenario has been considered, supposing that the second train run along 15-1 direction experiences a breakdown after his departure from station 15 causing a degraded functioning state of the train no. 1 which increases the respective travel time of 2 times. Then in such conditions three different recovery strategies have been analyzed and for each one the effects on both QoS offered to passengers and SA have been assessed. Descriptions and simulation results obtained for each strategy are reported in the following paragraphs.

First operational strategy: return to depot and successively substitution

This strategy consists in keeping on service the corrupted train in degraded conditions, until it reaches the depot near station no. 1. Once this has entered the depot it is substituted by a hot spare vehicle (train no. 9) which starts its service from station no. 1 (obviously along 1-15 direction). As shown in Figure 7.11, due to the higher travel time experienced by the broken vehicle, a consistent delay is transferred to other trains. The delay suffered by trains induces a passenger overloading of runs (Figure 7.12), especially for those which enter on service after the occurring of the failure event.

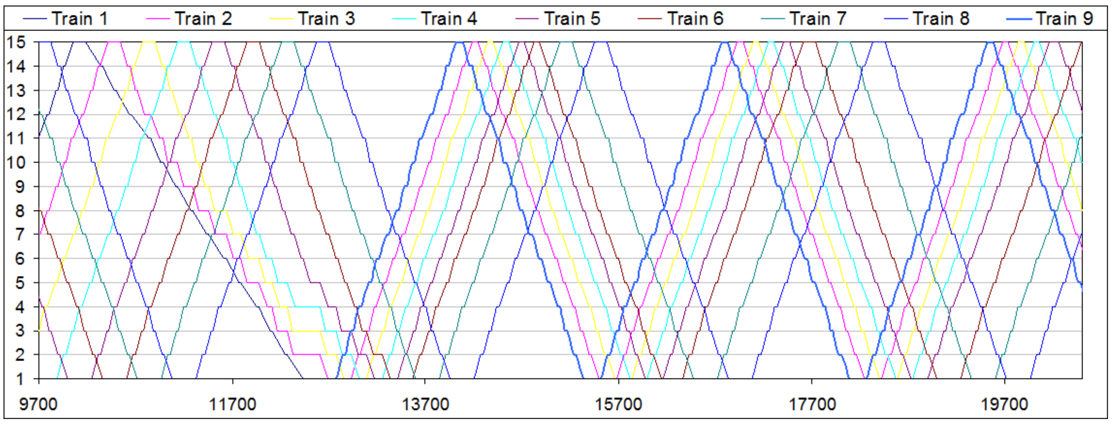


FIGURE 7.11: System operation for the first recovery strategy

Moreover this effect is obviously higher for train runs along 1-15 direction (because of the higher demand level) and slowly tend to fade away after the entrance of the spare.

Anyway for this strategy the total generalized cost estimated is 83905 euros. Therefore with respect to ordinary service conditions such strategy determines an increase in the total generalized cost (i.e. a decrease in passengers satisfaction) of about 31%. The effect induced by such strategy on SA has been detected through measuring the punctuality index at terminal station no. 1. In particular such index is equal to 76.57%.

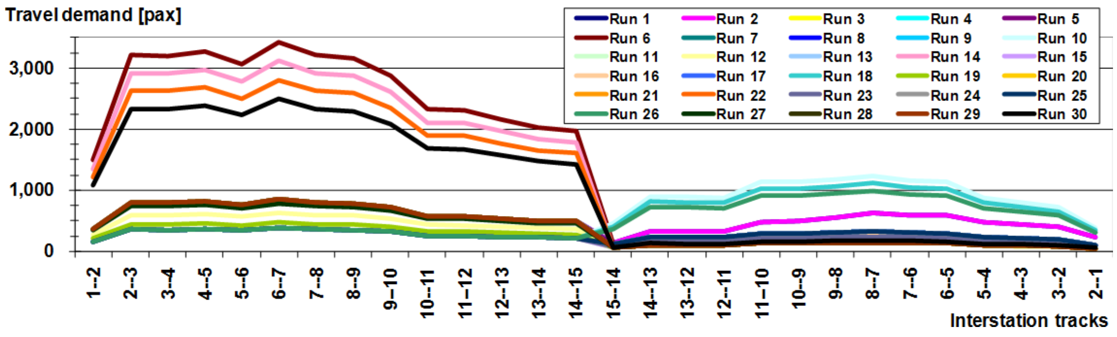


FIGURE 7.12: On-board passengers for each train run for the first recovery strategy

Second operational strategy: preventive insertion

Such strategy instead considers that a minute after the failure has occurred a hot spare from the depot is put on service from station no.1 along 1-15 direction, while the corrupted train, although is degraded, continues its service along 15-1 direction until it reaches station no. 1 and enters the depot.

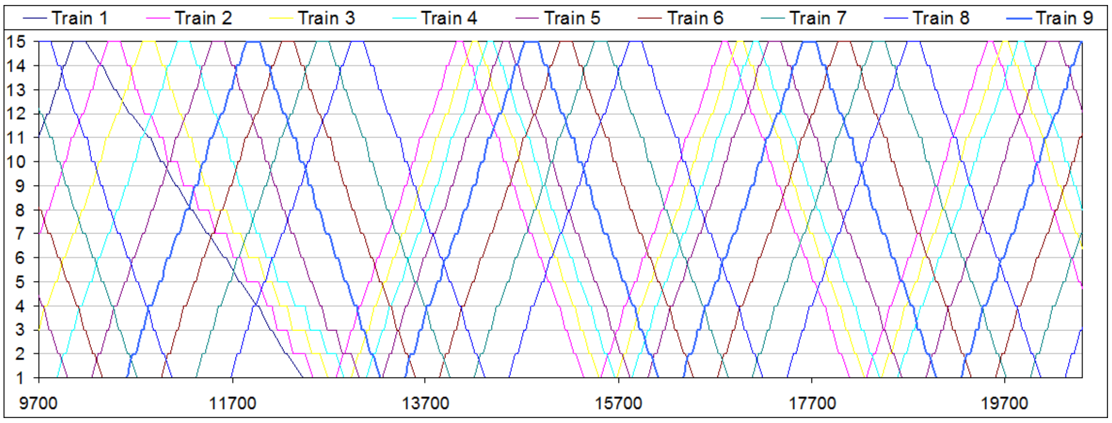


FIGURE 7.13: System operation for the second recovery strategy.

As in the first strategy, also for this one a passenger overloading of train runs happens (Figure 7.14) especially for 1-15 direction, but in this case train loading rates are lower, since the delay transferred from the corrupted train to other trains is lower. However for this second strategy the assessed total passengers generalized cost is 70614 euros. This means that with respect to regular conditions such strategy induces an increase of about 11% of the total generalized cost. The immediate introduction of the spare vehicle makes punctuality index be 83.23%, reducing the lost runs from 6 to 4 (Figure 7.13).

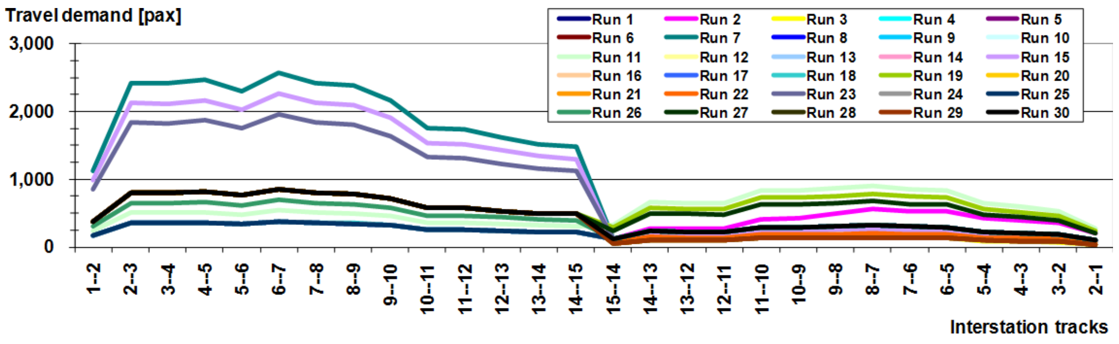


FIGURE 7.14: On-board passengers for each train run for the second recovery strategy

Third operational strategy: store away on pocket track

The last analyzed strategy consists in keeping on service the broken train in degraded conditions along 15-1 direction, until it reaches section between station 8 and 7 where it is stored away on the pocket track there located. Then a hot spare from the depot is put on service from station no.1 along 1-15 direction (Figure 7.15).

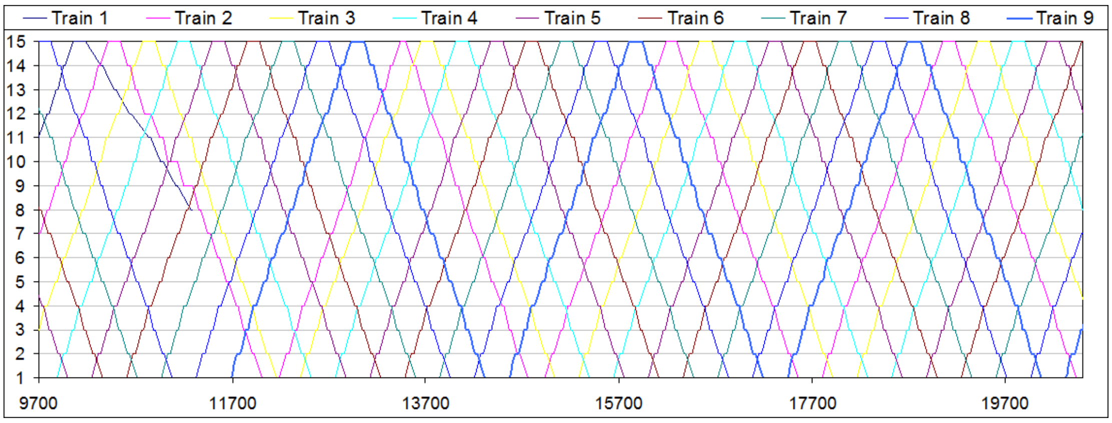


FIGURE 7.15: System operation for the third recovery strategy

This highlights that such strategy strongly mitigates the impacts of train knock-on delays transferred from the corrupted train to the other runs (Figure 7.16). Anyway the total passengers generalized cost calculated for this strategy is 64234 euros, and with respect to ordinary conditions determines a cost increase (i.e. a satisfaction decrease) of only 0.5%. The value of the punctuality index during the considered simulation period is 94,57%, but higher costs for the installation of the pocket track are necessary to put in practice this strategy.

7.3.3 Quality of Service vs Service Availability: a case study

QoS perceived by passengers of a railway system is strongly dependent on the levels of provided service availability. In particular passengers' satisfaction is related to the gap

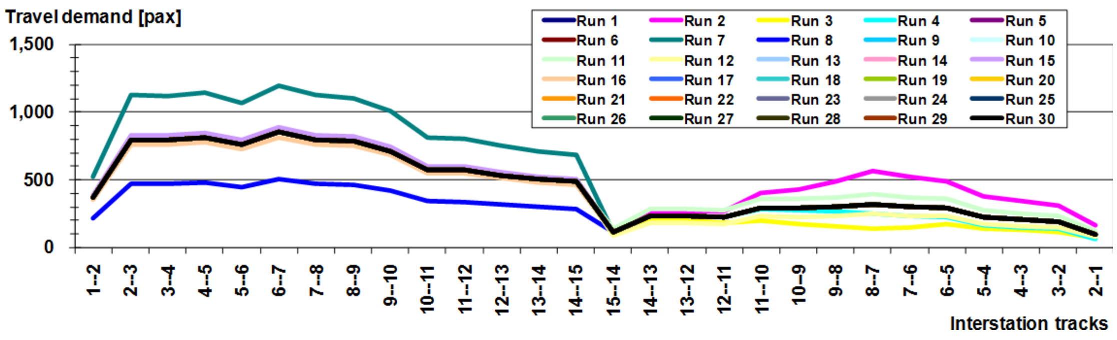


FIGURE 7.16: On-board passengers for each train run for the third recovery strategy

between expected (ideal) and perceived (actual) levels of QoS. In turn perceived QoS strongly depend on the discrepancy between targeted and performed SA levels achieved by service providers. Such relationship is clearly described in Figure 7.17, as illustrated by AFNOR, the French Organization for Standardization [27]. As can be seen there is a conjunction between elements related to customers (expected and perceived service) and elements pertaining to service providers (targeted and delivered service) which close in a loop, and only if this loop is retained the service is considered as successfully offered.

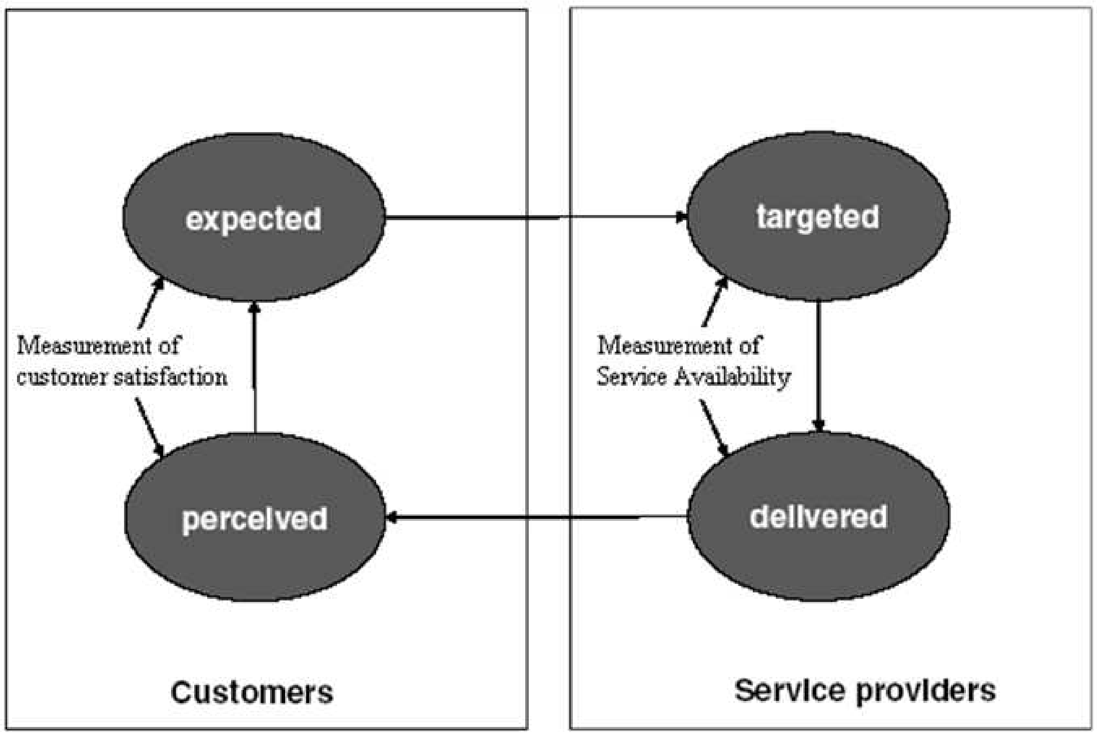

expected
targeted
Measurement of customer satisfaction
Measurement of Service Availability
perceived
delivered
Customers
Service providers

FIGURE 7.17: Service level loop in railway transportation systems (AFNOR, 2006)

For instance in a railway passenger system, it seems clear that passengers' satisfaction levels will increase only if railway actors are able to decrease the gap between objective and supplied service availability. But this relationship need to be verified and demonstrated. For this reason we implemented the following failure management strategies on the same system previously presented (Section 7.3.2):

1. Strategy 1: one minute after the failure has occurred, a spare from the depot (train no. 9) is put on service starting from station no. 1 along direction 1-15, while the corrupted train, although in a degraded state, continues its service until it returns to station no. 1 and enters the depot (where it is stored away).
2. Strategy 2: the broken train is kept on service until it reaches the pocket track (between stations 8 and 7) where it is stored away. A minute later a spare from the depot (train no. 9) is put on service starting from station no. 1 along 1-15 direction.

In the case of a failure causing the train performance reduction of 25% we have the following results.

The application of strategy 1, with respect to the previous scenario causes only slightly knock on-delays (Figure 7.18). In fact a light passenger overloading of train runs is observed (Figure 7.19). Punctuality index assessed is 92.64%, while the total generalized cost is 64952 euros, causing an increase of passengers cost (i.e. a decrease of passengers satisfaction) of only 1.66% with respect to ordinary conditions.

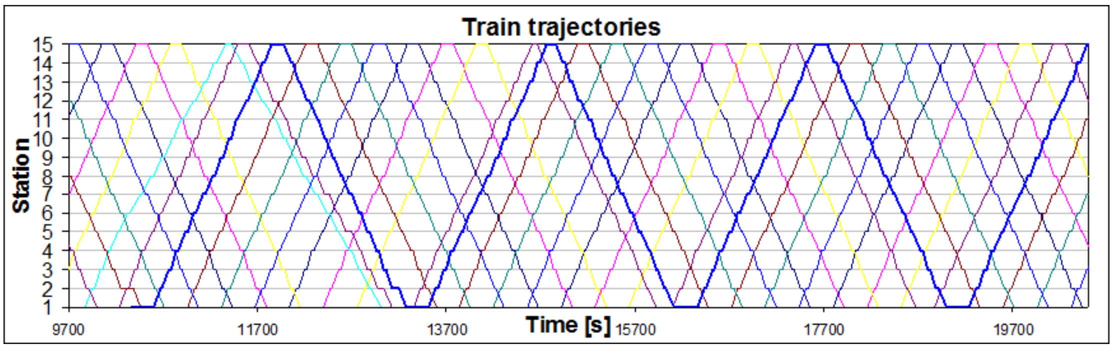


Figure 7.18: Train trajectories (Strategy 1)

As in the results obtained in Section 7.3.2 strategy 2 minimizes knock-on delays (Figure 7.20), in fact the punctuality index is 95.56%, but here the total passengers generalized cost is 65605 euros determining an increase of 2.68% with respect to normal service (see overloading of Run 3 in Figure 7.21). It is very important to notice that for this failure scenario, strategy 2 induces a lower QoS perceived by passengers, although it is more efficient from the SA (i.e. system performance) point of view, since when train performances are reduced only slightly (as in this case) it can be more convenient for passengers to have light increases of on-board running times rather than alight from the corrupted train run and wait for the next run. This proves that a design or operational intervention, which presents the highest performance efficiency, not always coincides

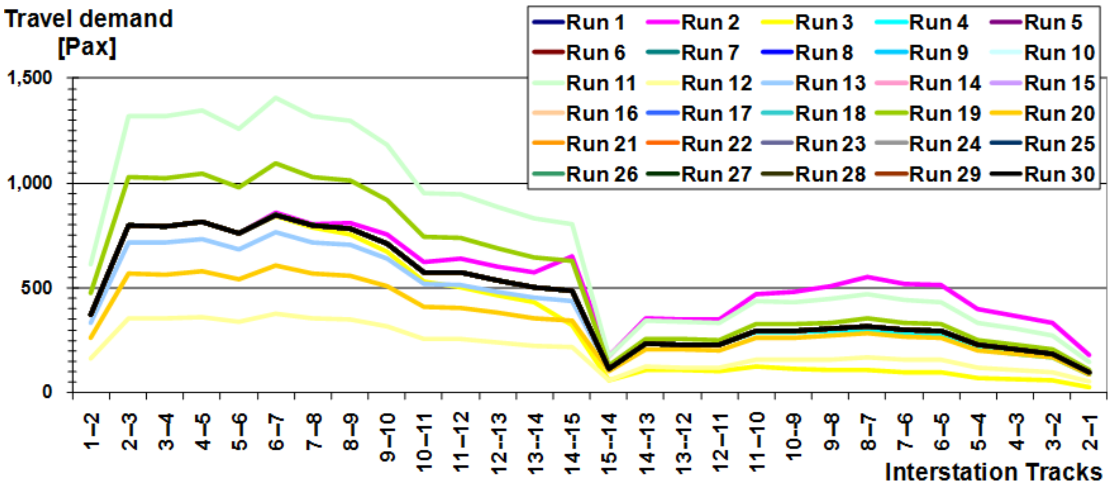


FIGURE 7.19: On board pax flows for each train run (Strategy 1)

with the one which assures the highest QoS levels: in the considered example,in fact, users prefer to reach the destination with short delays than unboarding and boarding the successive run (also if the total delay is minimized).

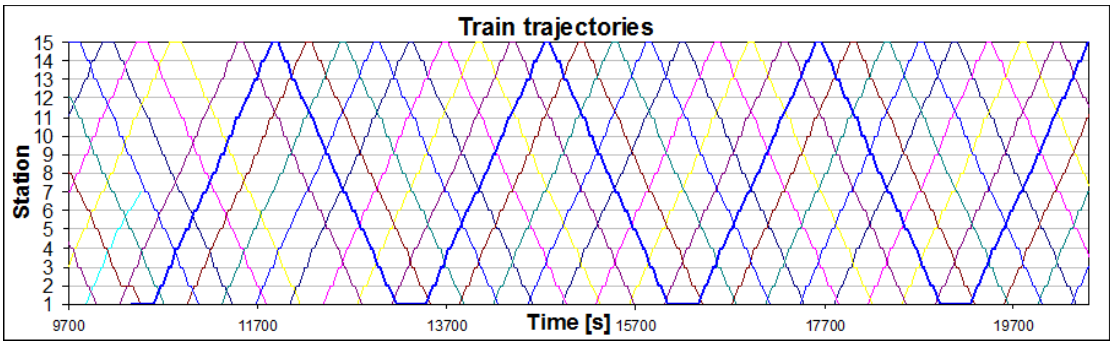


FIGURE 7.20: Train trajectories (Strategy 2)

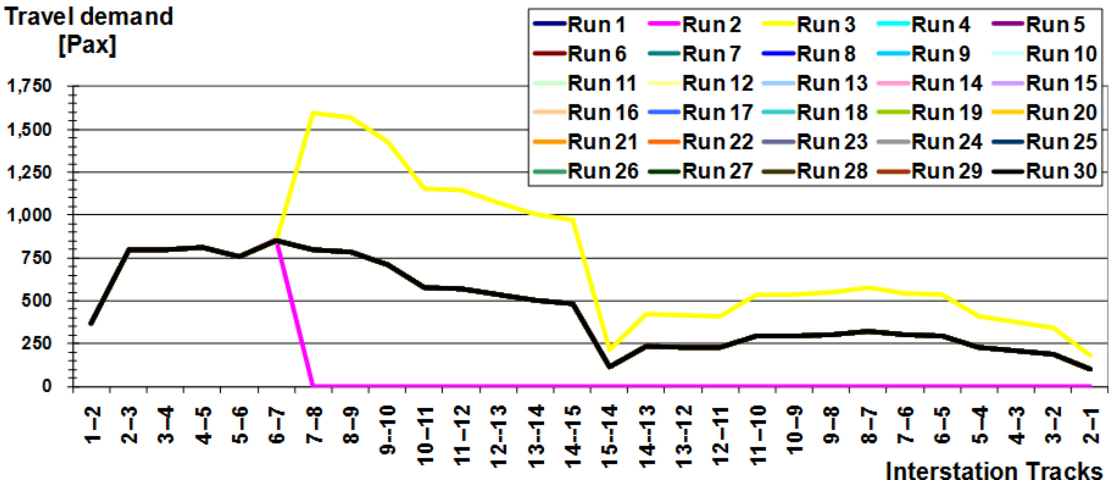


FIGURE 7.21: On-board pax flows for each train run (Strategy 2)

7.4 Vulnerability modelling and analysis of railway infrastructure systems

Railway systems are also security critical systems, then they need to obtain an acceptable level of its physical infrastructures. The vulnerability quantitative evaluation can be performed using the same approach: formal models can be developed with the aim to give a measure of vulnerability while Model-Driven can generate these models.

In this Section the CIP_VAM profile is applied to vulnerability and protection modeling of a railway station. The physical layout of the station, potential attack scenarios and protection systems are modeled in order to provide a high level representation to be used for the subsequent analysis phase. For the sake of brevity only an excerpt of the model is presented.

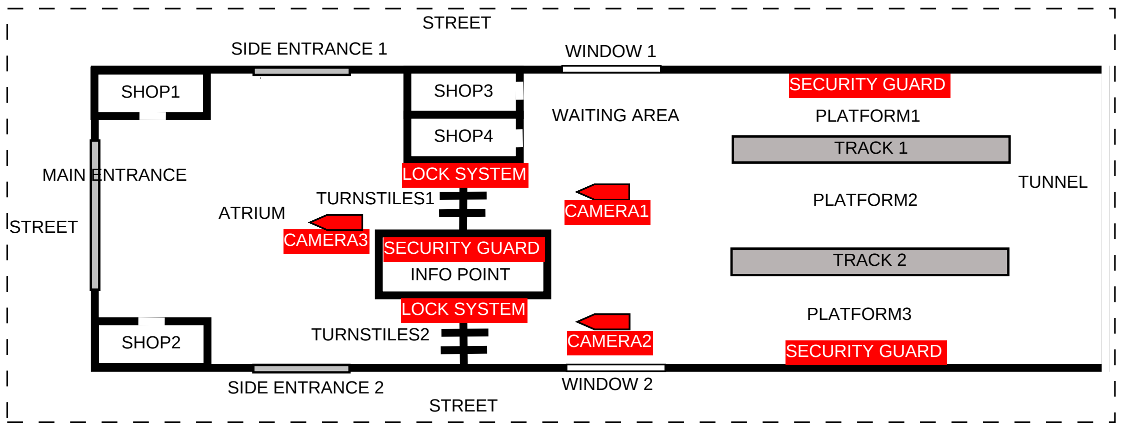


FIGURE 7.22: Layout of the Station

Figure 7.22 depicts the station layout; it has three accesses from the street: a main entrance and two side entrances. The station is internally divided into two functional areas: an atrium (on the right), that is freely accessible, and a waiting area (on the left), that can be accessed only by crossing turnstiles. Close to the two lines of turnstiles there is an info point that host security personnel. Four shops are located in the station: two of them are accessible from the atrium while the others from the waiting area. Two tracks host the two railroads, three platforms allow easy access to the vehicles, which arrive and leave the station through a tunnel. The protection systems are highlighted in red. Three video-cameras monitor the main entrance and the turnstiles, which are also equipped with a lock system controlled from the info point. Four security guards are also present: two of them supervise the platforms and the waiting area, the others control the access to the waiting area from the info point.

Infrastructure model.

Figure 7.23 (a) depicts the UML model of the station by the constructs provided by the Infrastructure package. The station is modeled by the stereotype *Site*, it in turn contains several sub-sites (shops, atrium, waiting area, etc.). The associations represent the interfaces of the station on the street and the tunnel (three entrances, two windows and two portals). Notice that passengers inside the station have been represented by the stereotype *Object* through a member *Person* (with multiplicity tag equal to $n) defined inside the station specification. The tagged values related to the *Shop4*, *Person* and *SideEntrance2* have also been shown in the figure. Those related to *Shop4* show that CIP_VAM is able to represent geometrical features: *shape* annotates the length of 5 meters, the width of 3 meters and the rectangle shape of the shop, adding also the measure of the perimeter; *volume* adds information about the height. The tagged value *asset* identifies the assets to protect. In this example, it is applied on the *Site Shop4*, on the *Object Person* and, as described in the following, on the element modeling the service provided by the station. The probability of an attack against the shop is 0.1, the economic value of the shop is $shop4value and the risk level is *acceptable*. On the contrary, the probability of an attack against persons is 0.99, the value of each person is $personValue and the risk level is *unacceptable*.

Figure 7.23 (b) depicts the *StopInStation* service provided by the two objects *RailRoad1* and *RailRoad2*. This relationship is expressed by the tagged value *providedBy*, related to the *StopInStation* use case. This service is included by the *RailwayOperation* since it needs the correct execution of *StopInStation* to be properly carried out. Both services are assets since there is an economic loss due to potential unexpected stops: these values are $stopValue and $operationValue for the station stop service and the railway operation service, respectively.

Attack model.

Let us consider the following attack scenario (Figure 7.24 (a)): a terrorist wants to detonate a bomb set down close to the turnstiles (inside the shop4 or onto the platform3). Two use cases (stereotyped as ≪attack≫) are created, both of them including the *intrusion_in_waiting_area* attack. The tagged values associated to the actor *Terrorist* expresses that the attacks can be conducted by just one terrorist exploiting medium skill and medium firmness. The *tactic* tagged value is set to *intrusion* for the use case representing the intrusion in the waiting area while it is *bombing* for the others. The value of *threat* in *place_a_bomb_in_shop4* use case models the destruction chain of the successful execution of this attack: the explosion inside the station will propagate to all the members included in *Station*: hence there will be a destruction of the shop4 with an economic loss of $shop4value, the loss of lives (person) with the economic loss of $personValue (multiplied by $n) and the destruction of the two railroads with the

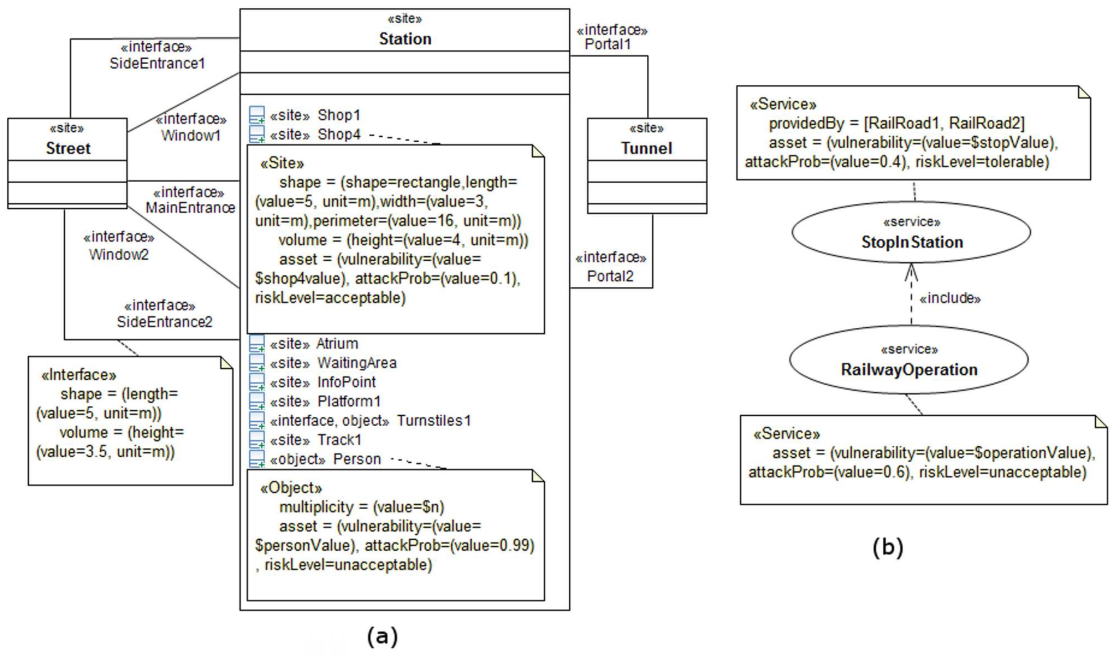

«site»
Station
«interface»
SideEntrance1
«interface»
Portal1
«site»
Street
«interface»
Window1
«site» Shop1
«site» Shop4
«Site»
shape = (shape=rectangle,length=(value=5, unit=m),width=(value=3, unit=m),perimeter=(value=16, unit=m))
volume = (height=(value=4, unit=m))
asset = (vulnerability=(value=$shop4value), attackProb=(value=0.1), riskLevel=acceptable)
«site»
Tunnel
«interface»
MainEntrance
«interface»
Window2
«interface»
Portal2
«interface»
SideEntrance2
«Interface»
shape = (length=(value=5, unit=m))
volume = (height=(value=3.5, unit=m))
«site» Atrium
«site» WaitingArea
«site» InfoPoint
«site» Platform1
«interface, object» Turnstiles1
«site» Track1
«object» Person
«Object»
multiplicity = (value=$n)
asset = (vulnerability=(value=$personValue), attackProb=(value=0.99), riskLevel=unacceptable)
(a)
«Service»
providedBy = [RailRoad1, RailRoad2]
asset = (vulnerability=(value=$stopValue), attackProb=(value=0.4), riskLevel=tolerable)
«service»
StopInStation
«include»
«service»
RailwayOperation
«Service»
asset = (vulnerability=(value=$operationValue), attackProb=(value=0.6), riskLevel=unacceptable)
(b)

FIGURE 7.23: UML Diagrams of the station infrastructure: (a) structural view (b) service provided

loss of the service *StopInStation*, provided by them, and the consequent loss of the *RailwayOperation*.

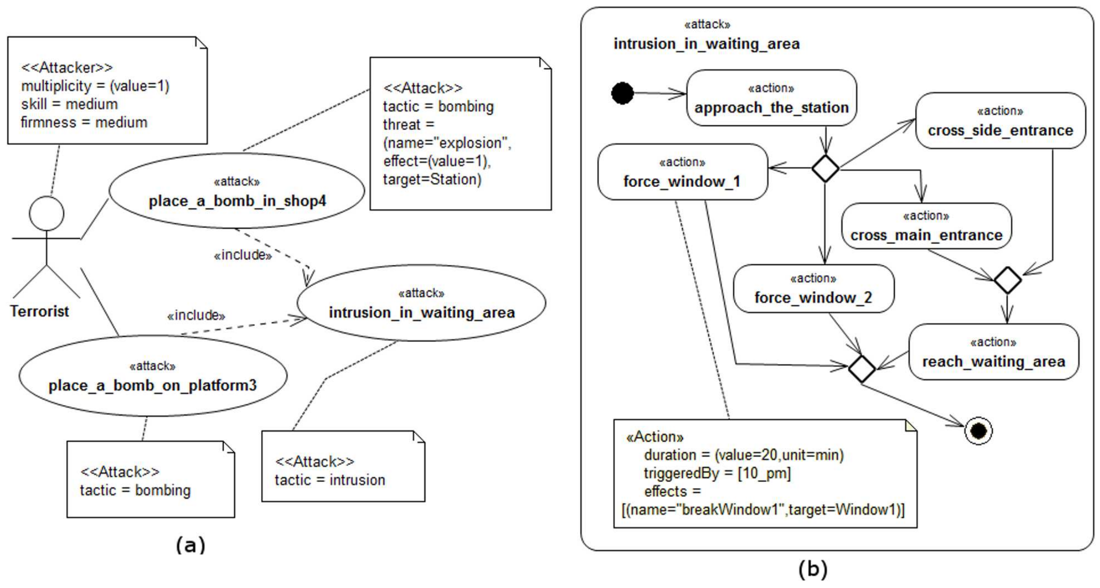


FIGURE 7.24: Attacking station: (a) Attacks Use Case Diagram, (b) Intrusion Activity Diagram

The *intrusion_in_waiting_area* attack scenario is then detailed by means of the Activity Diagram depicted in Figure 7.24 (b). The action sequence models the possibility to reach the waiting area by crossing one of the two entrances, or by forcing a window after 10 p.m. (when the station is closed). This last condition is expressed by the *trigger* tagged value, while the effects of this action are modeled through the *effects* tagged value.

Protection model.

Figure 7.25 depicts the UML model of the protection systems developed in the station. In particular the model highlights how the tagged values can express the technical specifications of the protection systems. The *counteracts* tagged value lists the actions against which the protection systems are effective (e.g. the action *reach_waiting_area* may be detected by the camera and the lock system), while the *exposes* tag adds information about the nature of the attack it suits.

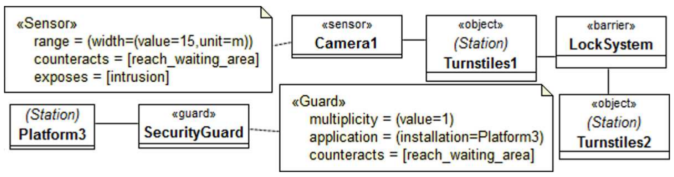


FIGURE 7.25: UML view of some station protections

Formal model generation.

In this last paragraph we want to highlight a possible way of using the CIP_VAM profile in the vulnerability analysis of a critical infrastructure: proper Level 2 M2M transformations can be defined to translate a UML model (composed of Infrastructure, Attack and Protection submodels) into a formal model suitable for analysis. In this case a Bayesian Network (BN) model can be obtained from the UML model in order to perform vulnerability analysis. Here we sketch a simple example to show that infrastructure, attack and protection models concur to the definition of a BN model to evaluate the vulnerability of the station.

Let us consider the *intrusion_in_waiting_area* attack: Figure 7.26 shows a fragment of the BN suitable to evaluate the vulnerability of the station from this attack. The three dashed boxes represent portions of the BN obtained from the information of the UML submodels linked by the dotted line. Starting from the activity diagram in Figure 7.25, a BN node is generated for each activity: when two activities are adjacent, there is a dependency (arc) between the nodes (e.g. *approach_the_station* and *cross_side_entrance*); when a decision node is present, several nodes can have the same parent node/activity. The Infrastructure package is used to reason about co-location of both attacking events and protection system in the same place: as example *attack_going_on* is a node that means the continuation of the attack when the *cross_side_entrance* branch of the activity diagram is considered. Let us note that the *attack_going_on* depends on three nodes that model the presence of the attack inside that area (*cross_side_entrance*), the firmness of the guard (*guard*) and of the attacker (*attacker*): a proper Conditional Probability

Table (CPT) is written to determine the output distribution of the *attack_going_on* node from the other three. The Protection submodel can be used to evaluate the effect of the reliability and of the trustworthiness of a sensor on the detection of attacking event. Table 7.4 represents the CPT of the *attack_going_on2* node (o) whit respect the *force_window_1* (e) and *camera_3* (s) nodes.

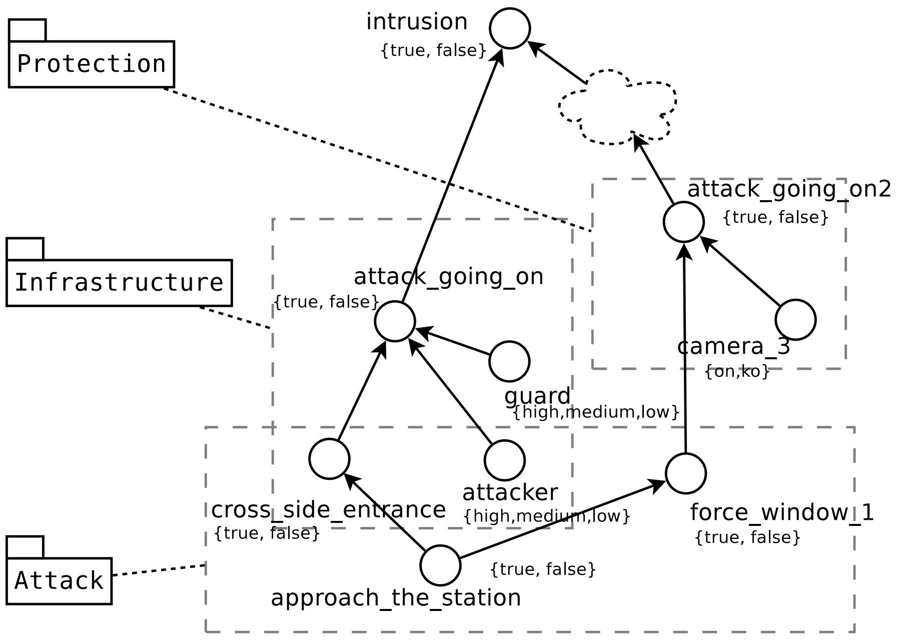


Figure 7.26: Portion of the Bayesian Network obtainable from the case study

Table 7.4: CPT of *attach_going_on2*

e	s	*o(true)*	*o(false)*
true	ok	1-fnr	fnr
true	ko	0	1
false	ok	fpr	1-fpr
false	ko	0	1

Chapter 8

IMS-Compliant Conferencing System

The IP Multimedia Subsystem (IMS) [75] is an architectural model for Next Generation Networking (NGN). This architecture has been designed for the convergence of all devices for telecommunications on an infrastructure based on IP network that is capable of providing advanced multimedia services on top of both mobile and fixed networks. IMS makes extended use of Voice over IP (VoIP) technologies and open IP protocols aiming at supporting heterogeneous devices in the exploitation of the entire portfolio of available multimedia services, which entails support for roaming as well as for flexible and transparent adaptation to context changes. Conferencing services are one of the most challenging among those engineered on top of such framework. They offer advanced communication experience to end-users exploiting a combination of audio, video, instant messaging, desktop sharing and other media, and impose a number of stringent requirements to the underlying network infrastructure. Due to its recent birth, the IMS architecture is currently far from reaching its steady-state with respect to the complete definition of the overall infrastructure and related standards. Furthermore, to date only a few early trials and deployments of the architecture are underway. A crucial point in the different system implementations is a systematic approach to the performance assessment.

The considered system has been already the subject of a thorough experimental benchmarking campaign carried out on a real testbed [76, 77] aimed at conducting a performance and scalability analysis. Experimental results showed that as the number of users increases, CPU utilization of the centralized server becomes the most critical performance index, since such server is responsible for the management of both the signaling dialogs with each conferencing client and the media streams mixing function.

This led the developers to focus on the performance of the media plane, which definitely represents the limiting factor as far as scalability is concerned. The mentioned experimentations required a lot of efforts: for each trial, a real laboratory testbed has been properly set up and a realistic set of conferencing sessions has been reproduced over it. A common approach to performance evaluation of networked systems which does not require real testbed implementations is of course based on the usage of network simulators. Nevertheless the current leading network simulators (such as ns-2, ns-3, OPNET Modeler, etc.) do not offer complete support for IMS [78]; in particular, they do not support the modeling of the media plane through embedded components, but usually rely on third-party implementations of some of the required IMS media plane functions. Compared with the experimental and the simulation approaches, formal modeling presents several advantages. It allows for performance evaluation and performance prediction thus supporting the early phases of the development of the system and providing the possibility to easily evaluate several conference blueprints. This also allows for sensitivity and stability analysis on a number of parameters, including those associated with Quality of Service, which gives the service providers an effective means to plan and schedule conferences according with the expected QoS levels. A further advantage in using formal modeling is the possibility to model and analyze anomalous behaviors due to attacks, faults, overloads or degraded operating modes.

This Chapter shows the performance assessment of these systems, performed at formal stage of the proposed approach. In the modelling phase, advantages given by the formal concept of *Model Template* (Section 6.2) have been exploited. After a brief description of the conferencing framework, to provide the reader with the needed background information, the models organization is presented; then the SANs library models are described and finally some obtained results are presented. The validation of the model has been performed through a comparative analysis with the results of an experimental campaign conducted over a real-world testbed implementation of an IMS conferencing framework.

8.1 System description

A conferencing platform offers advanced conferencing features: system users, named "participants" or "conferencing clients", are enabled to create and join conferences involving any kind of media stream, including audio and video, as well as instant messaging or even gaming. Different conference kinds ("blueprints") are supported and can be highly customized by users aiming to create conferences best fitting their needs. The system is conceived to be scalable, i.e., able to support an increasing number of conferencing clients. This property is obtained by a proper cooperation mechanism among different "centralized conferencing islands", which will be illustrated afterwards. In Figure 8.1 an IMS network suited for the provisioning of conferencing services is shown.

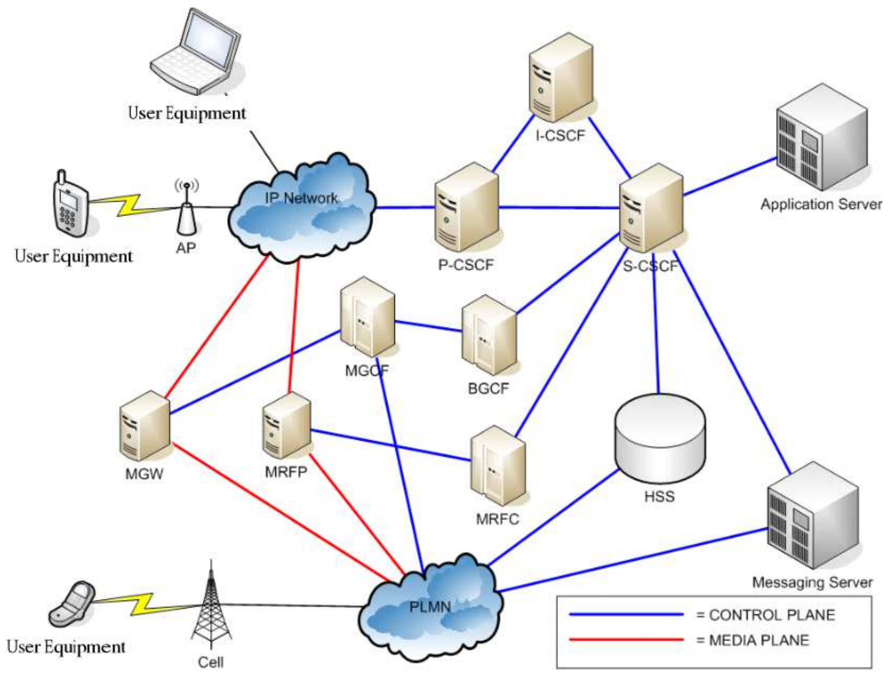


FIGURE 8.1: IMS network

As already mentioned, the conferencing system has the considerable advantage of being realized by exploiting standard architectural design specifications and protocols, deployed by eminent standardization organizations of both the Internet and the Telecommunication communities, namely the 3GPP and the IETF (Internet Engineering Task Force). As for the IMS specification, it complies with the standard defined in the 3GPP document [79] describing how IMS logical elements can be orchestrated in order to make the IMS network capable to support conferencing services. On the IETF hand, the system is an actual implementation of the RFC document dedicated to Centralized Conferencing (also known by the acronym "XCON") [80], that defines both the protocols and the components needed to fit advanced conferencing service requirements. For the sake of conciseness, we herein describe only the main concepts and entities of the aforementioned standards that are relevant for this work.

Based on the mentioned 3GPP specification, two logical planes can be identified: the "control plane" and the "media plane". The control plane deals with issues related to the set-up of the multi-media multi-user communication, as well as those related to overcoming the heterogeneity of both the access networks and the end-user devices. The media plane faces all the matters related to the media flows transmitted and received by users, such as, for example, the transcoding across different media formats and the switching of different media streams among conferencing participants.

The most important entities for our work are:

- *User Equipment, UE*: the device used by the conferencing user to participate in the conference;
- *Application Server, AS*: the server-side entity responsible for the implementation of the conferencing application logic;
- *Media Resource Function Controller, MRFC*: the logical entity devoted to control operations on media streams according to information provided by the AS;
- *Media Resource Function Processor, MRFP*: the entity in charge to perform media processing by acting as a media mixer.

According to the IETF XCON standard, the conferencing application logic involves different functionality, including the signaling management for the call set-up, the delivery of conference notifications to participants, the creation and modification of conference instances according to user preferences, the handling of moderation and so on.

Given this bird's-eye overview of the main reference standards, we now provide further details of the real-world conferencing system we took under analysis. In such system, the AS provides all the XCON functionality. Moreover, it acts as a MRFC by managing the MRFP, which is implemented as a different component that we will call from now on "media mixer". According to the identified logical entities, the IMS elements have been replaced in the system with real-world components, either by properly extending existing open source components (like, e.g., in the case of the Application Server elements), or by creating them from scratch (like, e.g., in the case of the media mixer). The resulting IMS compliant architecture, as well as its implementation details, are introduced in [76, 77, 81].

In what follows, two different conferencing scenarios are considered: a centralized conferencing scenario and a distributed conferencing scenario. The second one has been introduced in order to improve the scalability of the conferencing framework. A recent implementation of such scenario has been proposed in [81].

8.1.1 Centralized Conferencing Scenario

Figure 8.2 presents a simplified view of the system, showing the main IMS components we mentioned before. We refer to such IMS cloud as to a "centralized island" of the system, since all signaling messages from conferencing clients are directed to the same centralized AS and hence realize, at the level of the control plane, a typical star topology.

Being the center of such topology, the AS is also called "conferencing focus", or simply "focus".

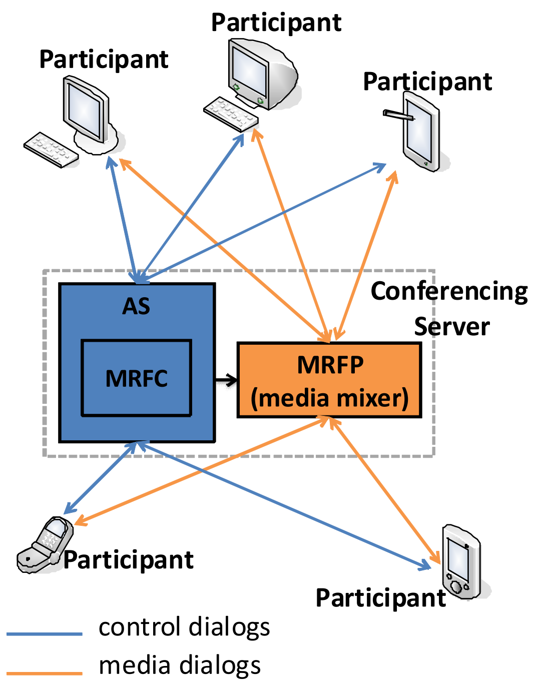


Figure 8.2: A view of the conferencing system

On the basis of the adopted conference blueprint, the AS issues commands to the media mixer, which is the system entity in charge of performing the audio and video mixing functions needed to deliver to each conference user the proper mixed stream, according to her/his preferences as well as to the supported capabilities of the device she/he uses. The multimedia streams generated and/or consumed by each participant are in fact always exchanged with the media mixer, hence realizing a star topology also on the media plane.

The AS and the media mixer can be logically grouped into an integrated server-side logical entity, called "Conferencing Server", making available all the functions needed to provide the conferencing service, both on the control plane and on the media plane.

8.1.2 Distributed Conferencing Scenario

We herein move the attention to the distributed scenario, in which several centralized conferencing clouds are interconnected and cooperate in order to provide the conferencing service in a distributed manner (Figure 8.3).

Under the assumption that all conference participants refer to the focus in the home network of the conference initiator, which we call the "main" focus, the IMS centralized

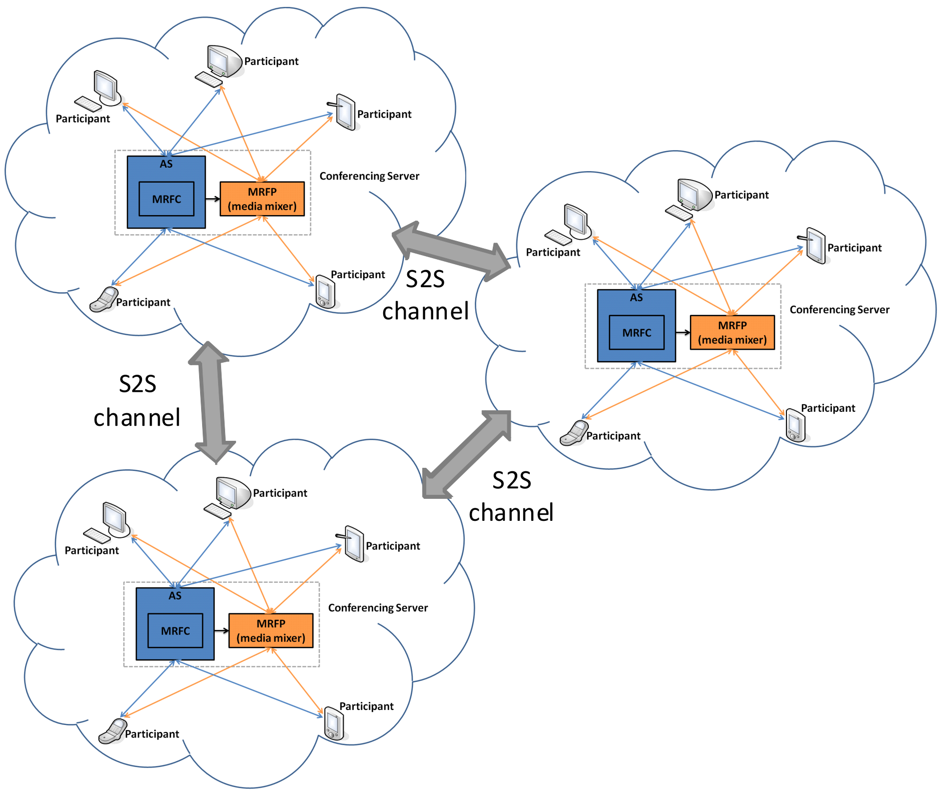


Figure 8.3: Distributed conferencing scenario

conferencing solution keeps on working also in the scenario where the conference participants belong to networks owned by different telecom operators. Though, in a fully distributed scenario, the above case should be dealt with by providing each involved network with a specific focus managing the associated local users who subscribed to the conference. It is up to the main focus in the conference initiator's home network to provide all other focus entities with up-to-date conference information. Such foreign focus entities thus play a twofold role. On one hand, they act as a regular conference focus for the participants belonging to their underlying managed network; on the other hand, they appear as normal participants to the focus in the conference initiator's home network.

Similarly, on the media plane, multiplexing/demultiplexing of multimedia streams generated by local users is up to each local media mixer. The resulting stream coming from foreign media mixers to the main media mixer (i.e., the mixer located in the island where the distributed conference has been created) is handled in the same way as the media flow of an ordinary conferencing client. In order to achieve consistency in a conference involving more islands, a dedicated communication channel exists between each pair of focus entities participating in the conference ("Server to Server (S2S) Channel" in Figure 8.3). Such channel is used to exchange conference information, as well as to manage synchronization issues.

8.2 Model Development Process

Multimedia conferencing systems have already been a playground for the application of formal modeling. Several works focused on the orchestration of the different building-blocks a multimedia conferencing platform relies on, as it is the case in [82]. In the cited work, the authors mainly use Petri Nets and related modeling languages to perform composition correctness validation. Other researchers leverage Petri nets to model conferencing media flows characteristics, as well as time synchronization mechanisms. These approaches are definitely more relevant to our work, since we are concerned with media plane modeling rather than compositional aspects. In [83] authors use TPNs to properly describe the temporal scheduling of the multimedia presentation process, within the inter- and intra-streams synchronization constraints. TPN models of such mechanism are used to support the implementation of a system capable to control streams synchronization in a video conferencing application. As opposed to the mentioned work, we do not analyze media streams management to drive the construction of a system from scratch, but rather to model the workload of an existing conferencing media mixer and study its performance.

Since our goal is to provide a flexible tool for the evaluation of performance and scalability, we use SAN models to describe the conferencing server behavior, while dealing with users mixing preferences and coping with the diversity of the involved media and related codecs. SANs are more suitable than TPNs to our aims because of their modeling power and efficiency. Specifically, they provide the basic modeling mechanisms to easily integrate, in the models, data structures representing messages, as well as to replicate and compose submodels.

In this Section we present the development process adopted to build the performance model of the conferencing system described in Section 8.1. The modeling process is founded on the approach described in Part II: it is compositional and hierarchical, and promotes model reuse. To support the modeling phase, *Model Template*s have been applied, while to come wit scalability problems *Stub*s and *Reduced Model*s have been developed (since the required analysis need to be performed on servers).

8.2.1 Separation of Concerns: Modeling Levels

In Figure 8.4 the structure of both the Participant (Client Side) and the Conferencing Server (Server Side) entities is shown. Since we deal with performance and scalability analysis of the system, we concentrate the modeling efforts on the media plane: in fact experimental campaigns showed that operations like transcoding and mixing of the

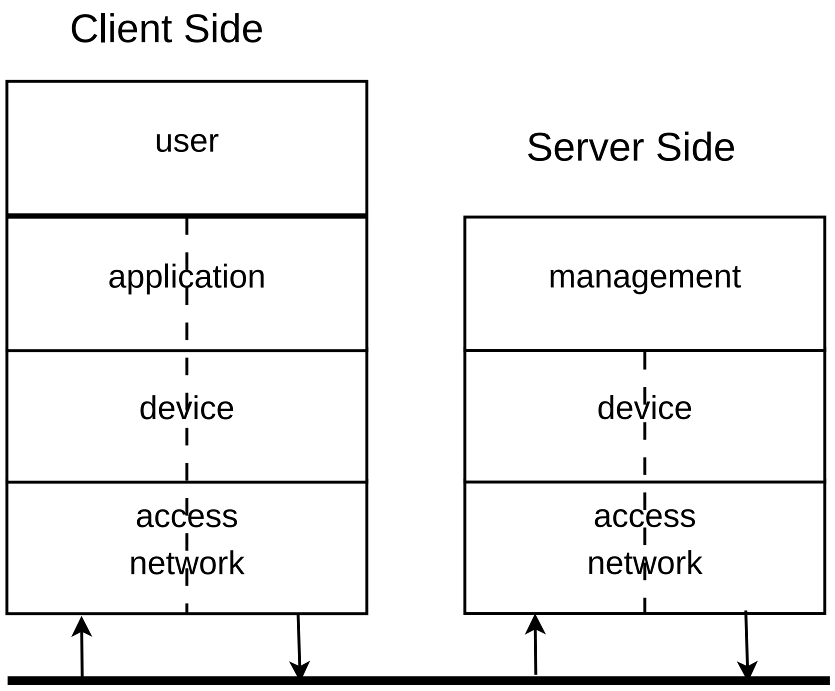


Figure 8.4: Reference modeling schema of conferencing systems

participants' media streams are the most demanding ones in terms of required resources and hence have the strongest impact on the overall system performance.

Client Side is composed by the following levels:

- *User*: at this level the media streams are produced (output streams) or consumed (input streams). The user is characterized by the role she/he plays in the conference (moderator, speaker, observer, etc.) and by the load that she/he produces.
- *Application*: at this level the streams are encoded/decoded. For example, the application samples and encodes the audio signal coming from the user's microphone. The production rate of the messages depends on the type of encoding (i.e., codecs adopted), as well as on both the power and the number of CPUs.
- *Device*: this level is in charge of bundling output streams into packets or viceversa (i.e., assembling input streams). At this level the features of the participant's communication device (the User Equipment) are taken into account. The main parameters are the power and the number of CPUs.
- *Access network*: it represents the access network of the device and is characterized by its connection speed.

Server Side levels are the following ones:

- *Management*: it includes the server-side functionality for the provisioning of the conferencing service on the media plane in each centralized conferencing cloud. More precisely, as already mentioned, we skip the modeling of the Application Server component, while stressing the details of the media mixer component in charge to perform media streams management on the basis of the particular kind of conference.
- *Device*: is the same as the client-side one.
- *Access network*: is the same as the client-side one.

8.3 Modeling the Conferencing Framework

This Section details how the modeling approach is applied to generate the SAN models of the conferencing system. Models are described according to a bottom-up approach: first, we introduce the models realizing each level of the conferencing system in isolation (Figure 8.4); then, we show how they are composed in order to obtain the models of both the Client Side and the Server Side. Some of the models described in this Section are template models, since they are obtained by instantiating specific Model Templates (specifically, User, Management, Client Side and Server Side). The subnets we will introduce to perform the composition in Subsection 8.3.6 are template models, too. The remaining models (Application, Device and Access Network) are not templates. For the sake of conciseness, only the formalization of the User Model Template is reported in the following.

8.3.1 User Level

In the conferencing context a user has the same behavior independently from the specific medium she/he uses (audio, video, etc). This behavior may be modeled by a state machine: a User Model Template that produces and consumes a generic multimedia stream $User_{SAN} = (UserMC_{SAN}, \{(K, S, f_K)\})$ is shown in Figure 8.5, where $UserMC_{SAN} = (userMedia, S, \varnothing)$: the SM set of $UserMC_{SAN}$ is empty and f_K is the instancing function described in the previous Section. User has one parameter K that represents the number of the media streams related to the user. Note that $SS_K{=}S$, i.e., this is the trivial case in which the entire structure of $UserMC_{SAN}$ must be replicated. Hence, $User_{SAN} < 1 >$ coincides with the Model Class $UserMC_{SAN}$. In order to show the flexibility of this mechanism, Figure 8.6 shows the template model $User_{SAN} < 2 >$ modeling a user who makes use of both video and audio.

The interface of this model is represented by the two places *MediaInbound* and *MediaOutbound*, respectively for inbound and outbound traffic. The user may be in a *Producing* or in a *Waiting* state (i.e., "talking" and "not talking" for the audio media) represented by ordinary places. The timed transitions (*P2W* and *W2P*) model the switch between the two states. When the transition *P2W* fires, a token is removed from the *MediaOutbound* place by the output gate *OG0*. On the contrary, when the transition *W2P* fires, a token is added to the place *MediaOutbound* by the gate *OG1*.

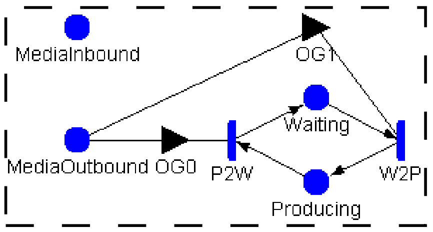


Figure 8.5: The User Model Template

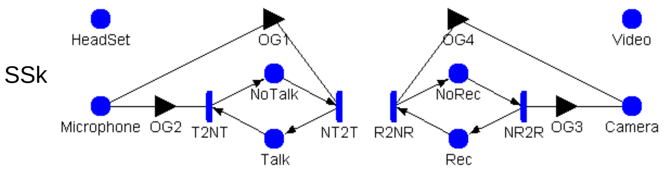


Figure 8.6: The audio conference user model

8.3.2 Application Level

At this level, we model the behavior of the encoding and decoding processes, on both Client and Server Sides. For each medium associated with the Participant, two different models are created, one for each stream direction: inbound-decode and outbound-encode. Therefore, we have a number of instances of the inbound and outbound Application model depending on the media enjoyed by the participant The two patterns of inbound and outbound application models are depicted in Figure 8.7 and Figure 8.8, respectively. The former model is able to maintain a token in the *output* place starting from a set of structured tokens (messages) arriving at the *app_in* place. The latter is instead devoted to the production of structured tokens to be put into the *app_out* place, with the presence of an incoming media flow being represented by a token in the *input* place. In detail, the interfaces of the inbound application model are represented by the places *output* and *app_in*: a structured token in *app_in* indicates the presence of a message ready to be decoded; after the decoding process (that keeps the CPU busy), the decoded data are ready to be played out. On the left of the image there is a watchdog that is responsible for maintaining the token in the *output* place: two places *playing* and *noPlaying* with the connected transitions and gates implement the state machine related to the watchdog.

A specular behavior is manifested by the outbound application model. A token in the *input* place indicates the presence of an external flow that needs to be encoded. In this case, *IG*1 enables the *sampling* transition that fires after a sample length and produces

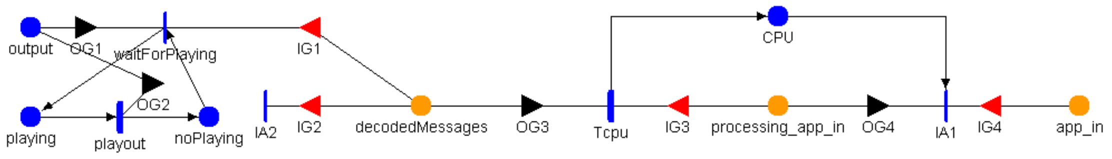


FIGURE 8.7: The inbound Application model

a new sample in the *samples* place. Upon arrival of a specific number of tokens in the *samples* place, depending on the encoding standard used, $IA1$ is enabled and, if the CPU is not busy, the generation of a message is activated. After a predefined processing time, *Tcpu* fires and the message is made available at the output interface *app_out*.

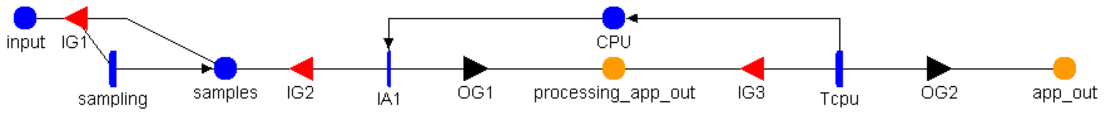


FIGURE 8.8: The outbound Application model

8.3.3 Device Level

The device layer is responsible for the decomposition of all output streams produced by a Participant, encoded with application messages, into one or more packets, or, viceversa, for the assembly of input packets into application messages. All the application level media streams are conveyed in an outbound Device model that decomposes each message into a number of packets, according to the length of the message itself, assigning to each of them an incremental identifier. Therefore, each packet is characterized by the message number and packet number that allow receivers to assemble the packets into application messages. Figure 8.9 depicts the SAN model of the device on both Client Side and Server Side: this model can be connected either to a network layer model or to an application layer working in both inbound and outbound configurations. The model interface resides in the *dev_in* (input) and the *dev_out* (output) places. In detail, the two input gates, $IG1$ and $IG2$, verify either the presence of a message (in the output configuration) or the arrival of the last frame of a message (in the input configuration) on the *dev_in* place. These two gates are able to recognize the direction in which the model has been assembled thanks to the packet number: if this value is not specified (or it is set to a meaningless conventional value, like, e.g., 0) it means that the model is working in outbound configuration, inbound otherwise. When an application message is (packets are) ready to be decomposed (assembled), the User Equipment processes it (them) until the related packets are (application message is) put into the output interface.

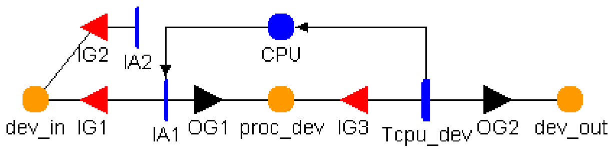


FIGURE 8.9: The Device model

8.3.4 Access Network Level

The network layer is responsible for getting packets from a source peer (Conferencing Server or Participant) and transferring them to the backbone network, when in outbound configuration, or viceversa, when in inbound configuration. The transfer time depends on the nature of the communication medium (Wireless, Wired, DSL, UMTS, etc.) to which the User Equipment is connected. Figure 8.10 depicts the SAN model of the access network: as for the device model, it is generic in the direction of communication (inbound/outbound) and the transfer time actually represents a parameter of the model. The interfaces of this model reside into *net_in* (input) and *net_out* (output) places, respectively.

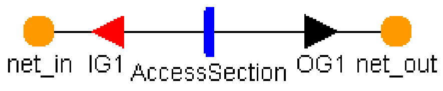


FIGURE 8.10: The Access Network model

8.3.5 Management Level

The Management level is specific to the Conferencing Server and it is not present in the model on the Client Side. This level is responsible for transcoding between the different media codecs adopted by participants, as well as for forwarding messages to them, by relying on stored information about the conference (that the Conferencing Server needs to keep in memory). The overall model of this level instantiates the aforementioned application model templates (inbound and outbound) to transcode messages, in conjunction with another model, described in this paragraph, that deals with the generation and transmission of messages. The overall Management model, like the application models, is specific to the particular media and can hence be instantiated ST times (ST being the number of media streams involved in the conference). Figure 8.11 depicts the SAN representation of this additional model.

Given its function, the additional model offers two interfaces to the device level (*app_in* and *app_out*), as well as four connection points with the inbound application model (*app_in_inbound* and *decodedMessages*) and with the outbound application model (*samples_outbound* and *app_out*), both used for potential messages transcoding. When a new message arrives

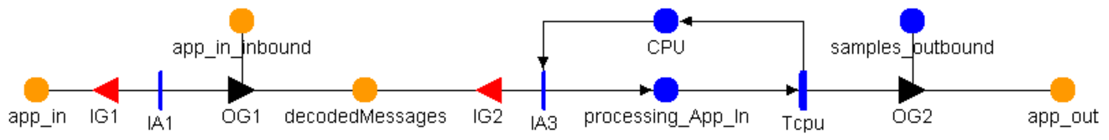


FIGURE 8.11: The Server Side Management additional model

at the server application level on the interface *app_in, IG*1*, IA*1*, OG1* check whether the codec used in the message is the standard codec chosen for the conference. In this case the message is put into the *decodedMessages* place; otherwise, it may be transferred to the inbound application model through the *app_in_inbound* place. The transcoded messages produced by the inbound application model are put into the *decodedMessages* place, where the CPU finds only messages encoded with the standard codec, which are elaborated and sent to all participants through $OG2$. Before putting messages into the *app_out* place, the server checks the codec of choice for each participant and, if not consistent with the standard one, it puts the samples into the *samples_outbound* place (connected to the outbound application model) for the correct message encoding.

8.3.6 Composed Models

The aim of this Subsection is twofold: it first shows how composition has been applied and then how it has been implemented in the Möbius framework. A two-level composition strategy has been implemented: an “intra-stack” composition to create Participant and Conferencing Server composed models, and an “inter-stack” composition at a higher level to create the overall conferencing system model. The “intra-stack” compositions representing the Client Side and the Server Side model stacks are shown in Figure 8.12 and Figure 8.13, respectively.

Starting from the top of the Figure 8.12, the Model Template described in Subsection 8.3.1 is used inside the User level according to the parameter K (the number of involved media). At the Application level, several models described in Subsection 8.3.2 are used. The number of inbound and outbound application model replicas is determined from N and M, representing, respectively, the number of received and transmitted media streams. The two values of N and M can be different (and not equal to K) in those cases when a client is producing but not receiving a specific medium (like, e.g, for a mobile phone, not equipped with a camera, which is not capable to generate its own video stream, but can nonetheless receive video sent by other participants). In the simplest case of an audio conference where all participants can both listen and speak we have $K = M = N = 1$. Both at device and at access network levels two models, according to the direction of the media flows, are instantiated. It is clear that the whole Participant model is itself a Model Template since entire submodels can be replicated. Figure 8.13

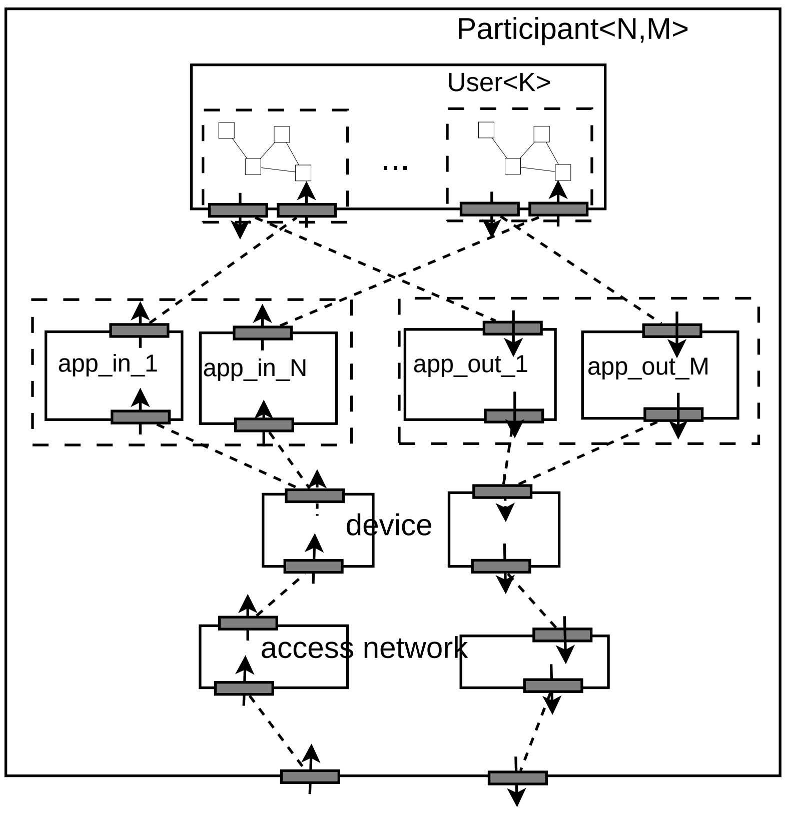


FIGURE 8.12: Client Side stack Model Template

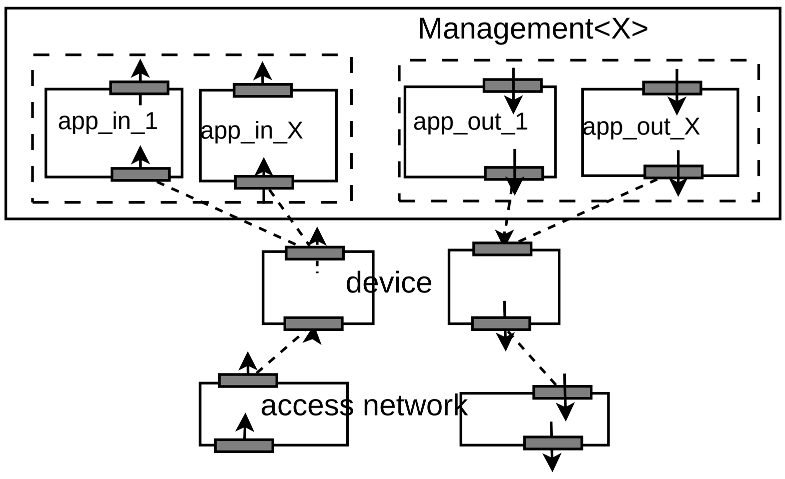


FIGURE 8.13: Server Side stack Model Template

depicts the “intra-stack” composition at Server Side where the different part is represented by the Management model (since Server Side and Client Side use the same Device and Access Network models). The model is in charge of representing the Conferencing Server’s audio and video mixing functions: in order to capture such concepts, the model is built by using hierarchical composition. In fact, transcoding media streams may be necessary depending on the adopted encoding-decoding formats, as well as on the capabilities of the involved devices. Encoding and decoding are modeled by using the application model templates (respectively *app_out* and *app_in*) described in the previous paragraph. These are model templates in X parameter, that is the number of the different encoders and decoders used in the conference.

The “intra-stack” composition is performed by place superpositions, where it is naturally possible, or by using some interconnection SANs: Figure 8.14 shows the Model Template of the interconnection network between device and application models on the inbound branch of both Participant Side and Server Side. Figure 8.15 shows the real instantiation obtained by replicating, for two media (audio and video), the SS_n subgraph. The latter model allows to connect the device level of a Participant/Conferencing Server to the two inbound application models. It basically represents a simple messages separator, working in accordance with the media to which they refer.

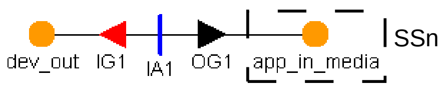


FIGURE 8.14: The Device-to-Application composition network Model Template

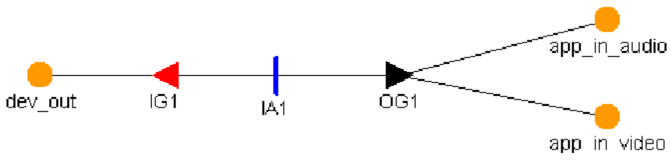


FIGURE 8.15: A Device-to-Application composition network model

The “inter-stack” composition, starting from the definition of conference deployment (in terms of the number of conference participants, as well as overall Conferencing Server configuration), joins all the instances of the peers stack model (Participants and Conferencing Server) through a proper network, on the basis of a provided topology.

The two composition steps described above are implemented into the Möbius framework by means of the Rep and Join operators. The set of participants is created by replication and instantiation of the entire Client Side model stack. Figure 8.16 shows an example of an overall composed model of a centralized audio conferencing system. The “intra-stack” composition has been implemented through the *participant* and *server* Join models; similarly, the“inter-stack” composition is given by the combined action of both the *Rep* (to replicate participants) and the *system* Join model (acting as a top level joining all the involved entities). To identify single participants, by assigning a specific id to each of them, a simple *idAssignment* model has been implemented. This model is similar to the one proposed in [71] and allows to associate an index to each participant instance. Indeed, having non-anonymous participants is critical when, for example, a measure on a specific instance is requested or a different behavior must be modeled for it. The composition SAN is in this case trivial, since different models have to share some communication places.

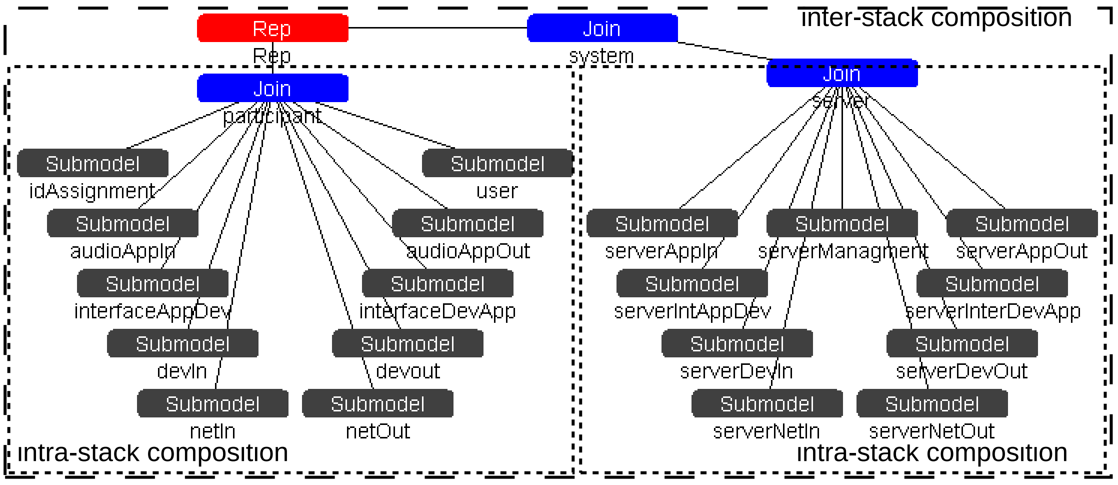


FIGURE 8.16: The overall composed system model

8.4 Stub and Reduced Models

This Section explains how stub and reduced models have been used to solve scalability issues.

Stub models have been implemented given the layered structure of the model. Developed stub models acts as a substitute for the models of the lower levels: for example at application level a stub model must encapsulate the behavior of device and access network levels, exposing the same interface to the upper levels. A trivial SAN can be implemented, where a timed activity can represent the delay as seen by applications; this activity fires with a distribution which approximates the sum of the real ones.

The *reduced* model can be connected directly to the Conferencing Server model stack generating the load related to n participants without instantiating all of them. This model is depicted in Figure 8.17: (i) the distribution function of the *production* transition models the average rate with which messages are produced by a single participant; (ii) the *activeClients* place is used to store, at each instant of time, the number of transmitting participants; (iii) the *verifyClients* input gate checks the presence of at least one token in the associated place and is used to either increase or decrease the number of active participants, for example in case of configurations involving moderation; (iv) the *generateMessages* output gate injects packets into the network. Obviously it does not allow to conduct performance evaluations at the client side.

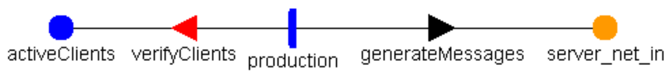


FIGURE 8.17: The client reduced model

8.5 Obtained results

This Section assesses the proposed approach by means of experimental campaigns. Our objectives are: i) validation of the proposed models through a comparison with real data; ii) assessment of the scalability of the modeling methodology; iii) analysis of its power and flexibility when addressing different conference configurations (e.g., with/without transcoding, with/without moderation, etc.).

TABLE 8.1: System parameters

Parameter	**Description**	**Value**
pkts_per_msg	Packets generated at device level for each application level message.	1
sample_len	Duration of a single audio sample (according to G.711 specification).	0.125 ms
samples_per_msg	Samples contained in an application level message (according to G.711 specification).	160
sample_dim	Dimension of an audio sample	8 bit
app_in_Tcpu_mean app_out_Tcpu_mean	Mean CPU time needed to decode/encode a message at the application level.	1.3E-4 ms
app_in_Tcpu_var app_out_Tcpu_var	Variance of CPU time needed to decode/encode a message at application level.	1.3E-5 ms
mngmt_Tcpu_mean	Mean of CPU time spent by the server to generate a single application message	1.3E-4 ms/msg
mngmt_Tcpu_var	Variance of CPU time spent by the server to generate a single application message	1.3E-5 ms/msg
dev_Tcpu_mean	Mean CPU time to decompose/recompose a message at device level.	0.6E-6 ms
dev_Tcpu_var	Variance of CPU time to decompose/recompose a message at device level.	0.6E-7 ms
net_band	Upstream/downstream bandwidth needed for a message	67 Kbit/s

To this purpose and according to related works [76, 77] we choose CPU usage of the Conferencing Server (for an audio conference) as a key performance indicator. Such measure is usually computed with respect to the number of participants connected to the conferencing system. Before showing the results of our analysis, a description of the system configuration parameters used in the trials must be given (Table 8.1). For the sake of coherence, such parameters have been chosen according to the above cited works. The first two objectives are fulfilled in Subsection 8.5.1 and the third one in Subsection 8.5.2

8.5.1 Validation with Real Data

This group of simulations aims at demonstrating the equivalence between our models and real experimental data. In order to achieve this goal, we compared the estimated CPU usage obtained by solving models with the values reported in [76, 77]. This validation requires a high level of scalability in terms of number of participants: for this reason, we introduce at first a comparative analysis of the complete model with *stub* and *reduction* approaches to evaluate their scalability features.

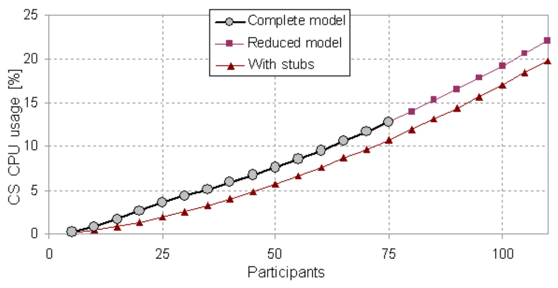


FIGURE 8.18: Scalability analysis: CPU usage

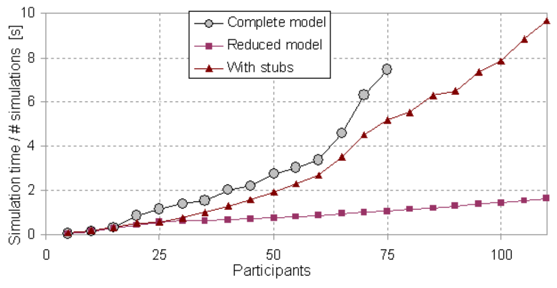


FIGURE 8.19: Scalability analysis: simulation time

Complete vs stub vs reduced model. Based on the specific well-defined reference scenario, the three models are compared: a complete model obtained by composing the server stack with the replicas of the participant stack; a model obtained by substitution of the *device* and *access network* levels with stub SAN models; a model built using reduction techniques as defined in Section 8.4. The accuracy of the *stub* and *reduced* models compared to the complete one, that is the estimated CPU usage of the Conferencing Server against the number of connected participants, is reported in Figure 8.18. The simulation times associated with these studies are instead reported in Figure 8.19. The two analyses have been conducted for a number of participants between 5 and 110: a quick look at the graphs tells us that the reduced model is able to analyze up to 110 participants within reasonable simulation time. The complete model is instead capable to arrive at a maximum of 75 participants on the computer used for the simulation (Intel(R) Core(TM) 2 Duo @ 2.53GHz, 2 GB RAM) due to RAM saturation during the solving phase. The reduced model scales very well in simulation time and, according to the first diagram, it is also as accurate as the complete model for the considered conference scenario. The drawback in its use resides in the potentially unsatisfactory level of detail: some analyses that require non-anonymous participants (e.g., the QoS of the conference as perceived by end-user), cannot be conducted because of its limited description of the participants' model stacks. On the contrary, a stub solution seems to best strike the balance between accuracy, simulation time and level of detail. In particular, the considered stub models superpose communication places without introducing CPU effort at device and network levels: the accuracy can be further improved by constructing more realistic (yet less performing) stub networks.

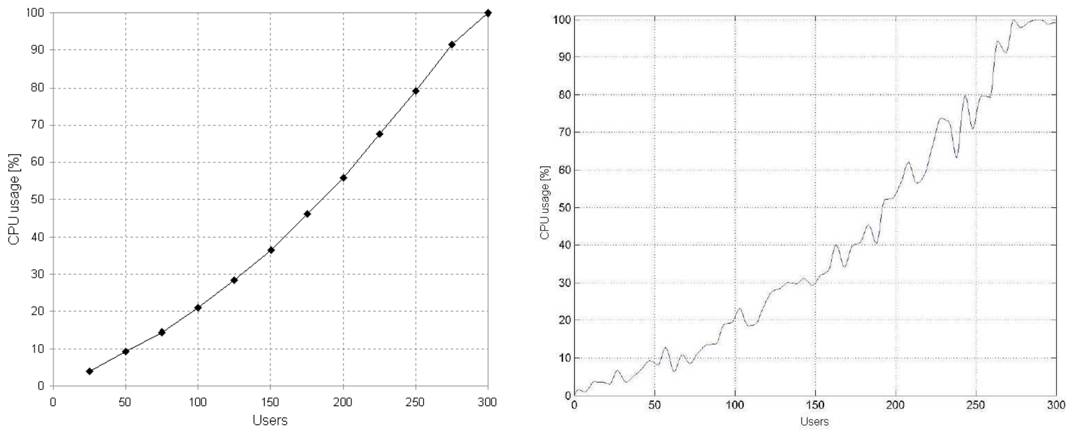


FIGURE 8.20: Reduced model vs real data: centralized scenario

Reduced model vs real data. We compare the solutions obtained with the reduced model with the real experimental data for the two different configurations described in Section 8.1: a) centralized scenario, where a single focus acts as the Conferencing Server for a number of participants varying from 25 and 300; b) distributed scenario, where a "two islands" configuration is used for load balancing purposes (150 participants per focus). The results are reported in Figure 8.20 and Figure 8.21, respectively. In both scenarios the reduced model fits well the real measured data.

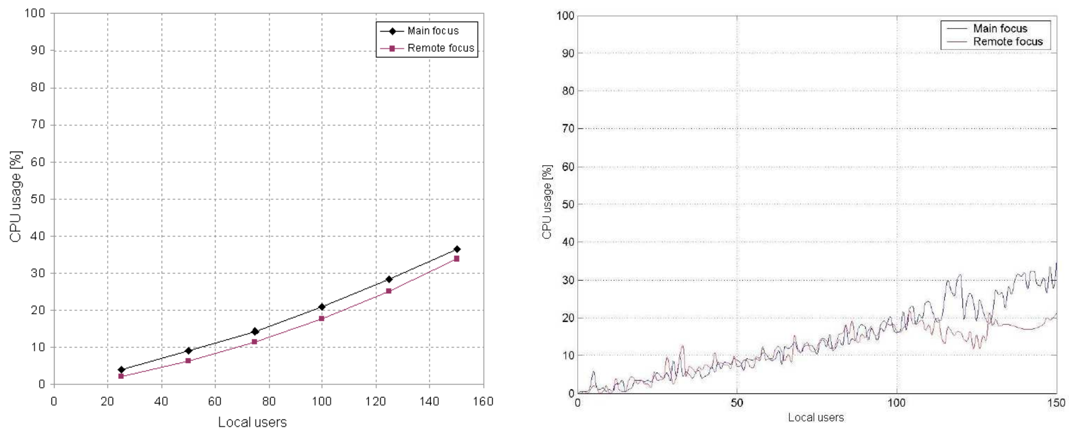


FIGURE 8.21: Reduced model vs real data: distributed scenario

8.5.2 Evaluating Power and Flexibility

So far we have validated our approach both in terms of accuracy of the results and scalability. The next two experiments point out the power and flexibility of the proposed modeling approach by evaluating the system's performance under different conference configurations. More precisely, we evaluate the effects of moderation and transcoding on the CPU usage of the central Conferencing Server for an audio conference. The aforementioned experiments are conducted by varying two key simulation parameters: the probability of being muted by the moderator and the probability of a flow to be transcoded. For both of them, we employed complete models, limiting to a value of 50 the number of participants. In the performed simulations, all participants use identical devices and present the same probability to talk and probability to have their flows be transcoded. However, the adoption of a complete stack model allows us to assign different behaviors and different devices to each user. This paves the way for further analysis that can be realized by introducing heterogeneity in the composition of the participants set. Moreover, this enables future works on performance studies conducted from a user's perspective, by focusing on parameters like, e.g., the performance of user devices, endpoint CPU utilization, perceived delay, etc..

Complete model: moderation-based conference. In this study, starting from a sample conference, we evaluate the difference between the CPU usage of the Conferencing Server either in the presence or in the absence of moderation. According to the User model described in Section 8.3, the evaluation can be characterized by the *Prel* parameter, that is the probability to release a conversation when talking (or, similarly, to be muted by the moderator in a moderation-based conference). Several values of the *Prel* parameter have been used for the analysis. Figure 8.22 and Figure 8.23 show the results of these simulations, by highlighting a reduction on the CPU usage as long as the *Prel* parameter increases. This is clearly due to the reduced number of incoming audio flows to be processed.

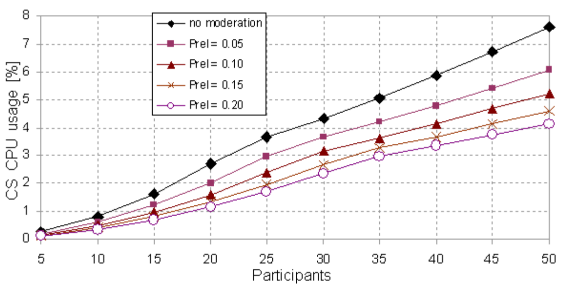


FIGURE 8.22: CPU usage vs number of users for different Prel values

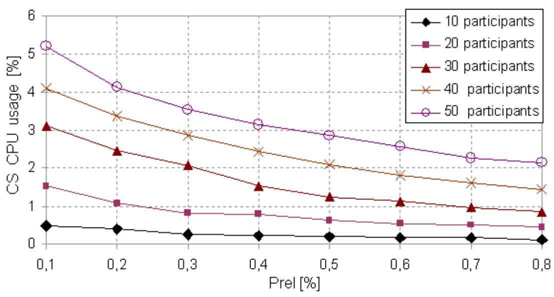


FIGURE 8.23: CPU usage vs Prel for different conference sizes

Complete model: conference with transcoding. In the last experiment we evaluate the potential impact due to the presence of transcoding. We suppose that a portion *Cdtrcd* of participants does not need any transcoding, while the remaining part requires that the Conferencing Server transcodes messages (which increases server CPU usage). Several values of *Cdtrcd* have been used. The results are shown in Figure 8.24 and Figure 8.25. We can notice how the CPU workload is progressively lighter as long as the number of participants needing server-side transcoding decreases (i.e., by increasing the value of the *Cdtrcd* parameter).

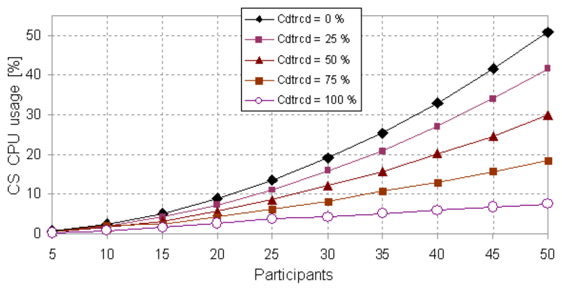


FIGURE 8.24: CPU usage vs number of users for different Cdtrcd

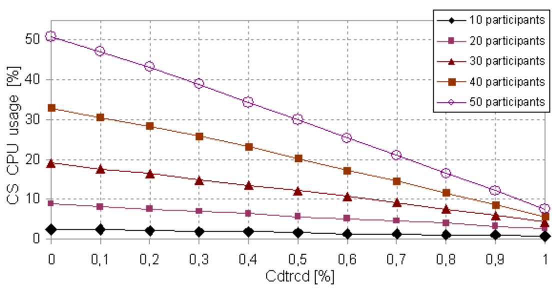


FIGURE 8.25: CPU usage vs Cdtrcd for different conference sizes

Chapter 9

Wireless Sensor Networks

A Sensor Network (SN) is composed of independent nodes that interact with aiming at collecting informations on the surrounding environment. The devices that make up the network are named "sensor nodes" and are equipped with a number of sensors of a heterogeneous nature. Between them, Wireless Sensor Networks (WSNs) are of particular interest; in them the devices are able to communicate without the use of cables. Among their advantages, it is important to remember a more simple installation and a more simple network topology adaptation (consequent to both mobility of individual nodes and nodes failures).

There are several applications of sensor networks, from fire prevention to ocean monitoring. The common feature of the different applications, however, is the design phase, where some basic parameters (density and number of nodes, sampling period, etc.) need to be defined, assessing their impacts on the system properties (i.e. performability, energy consumption). The main problem of sensor networks is the energy consumption: it becomes the first issue on which focus the design of the overall network. In fact, in real application, the recharge or replacement of the batteries is often forbidden, both for accessibility and cost motivations. In addition, sensor networks need to be tolerant with respect to failures of nodes or communication between them.

For this reason these systems can be classified as mission-critical systems since they need to operate for a determined time period, ensuring defined performability levels. This Chapter shows how the formal stage of the proposed approach can be applied also to the performance assessment of these systems. After a brief description of the sensors networks, to provide the reader with the needed background information, the models organization is presented; then the SANs library models are described and finally obtained results are presented.

9.1 System description

In actual SNs, the nodes are not simply transducers of physical quantities, but are more complex systems that integrate also storage, computing capacities, and communication interfaces. A generic sensing application, which requires the monitoring of a specific field, is characterized from the covered area: from a radius of a few centimeters up to entire geographical regions. Furthermore own characteristics of application fields are not known. For example, the application may require the collection of data on the ocean floor at unknown depths: the water pressure makes impossible for a diver to dive and explore the depth. The design of a sensor node have to take into account this characteristics to be able to work also in this condition. The density of sensor nodes in the operating environment is also crucial: it can range from some units to hundreds for a small area. This parameter is strongly dependent on the application: for example, in fire monitoring, the density is a variable parameter, as well as crucial, since there are some areas where firing probability is higher than others. This depends on environmental factors such as ground humidity, wind, etc. Choosing the density according to these parameters makes the dissemination less expensive and increases the effectiveness of the application itself. Both transmissive protocol and sensor nodes shall guarantee a good robustness degree with respect to the environment, physical damages or failures of other nodes. The latter issue gives a measure of tolerance, i.e. the ability of the sensor network to continue its service after failures.

Figure 9.1 depicts the functional architecture of a sensor node. Each sensor node is an embedded circuit in which one or more I/O peripheral are represented by a sensor; the communication is given to another dedicated subsystem. The processing unit, named processing elements, is responsible for executing the user program, opportunely stored on the node. The processing element manages the on-board sensors (also if not natively integrated), controls the memory unit, and drives the communication subsystem.

A sensor is a component that physically performs the conversion of a physical quantity into a signal of any other nature. In most cases the sensor is an electric transducer, i.e. it is a system that receives as input a physical quantity of a process bringing back the value by an electrical signal. The transduction is performed by additional electrical elements, such as signal amplifiers, voltage regulators, filters, etc. A sensor provides an analog or digital interface. In the first case the signal need to be processed by a A/D converter which perform the quantization and the sampling of the source quantity.

The network infrastructure requires that all the sensor nodes, after the sampling, collect data and send them to a central node, named sink. When needed, this node can act as a protocol bridge towards other networks. Conventionally communication between

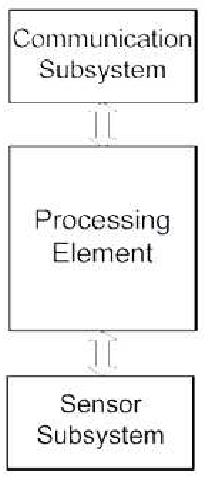


FIGURE 9.1: functional architecture of a sensor node

sensor nodes are performed through wired or wireless networks. The use of the first avoid power limitations, since it is possible to provide each sensor node with a power supply lines; the latter requires nodes to be equipped with power supply (e.g. through batteries), allowing larger network radius without the enormous environmental impacts given by cables. The network topology can be configured, ranging from a star topology (i.e. each sensor node is directly connected to the sink) to a generic tree (i.e. some nodes play also as data forwarders).

The communication and sensor subsystems are controlled by the processing element. Its main objective is to execute a program that contains instructions for managing the sensors, communications, power and other tasks relevant to the specific application. The processing element, typically, consists of a processor, a nonvolatile memory containing the instructions of the program to execute, and a volatile memory. In an abstract way the processing element can be thought as a component that allows communication elements to receive data collected by the sensors.

Currently, various types of sensor nodes have been designed. The most known families are:

- Mica (Mica, Mica2, Mica2dot, MicaZ);
- Telos (Telos, TelosA, TelosB, TmoteSky);
- Intelmote2;
- Eyes (EyesIFX, EyesIFXv1, EyesIFXv2).

The architecture is in the greatest part similar since they are evolution of the Mica node, designed by the Berkeley University in California. The most used node, in recent years, is the Mica2, which, although passed, is still the reference platform for researchers. The reference architecture of the analysis is the TelosB, that is described in the following Subsection.

9.1.1 TelosB node description

The revision B of the Telos architecture, with respect to the previous one, brings an improvement in performance, functionalities and expansions [84]. The Telos architecture exploits standards as USB and IEEE 802.15.4 to interact with other devices. It is produced in two versions: the first is a simple Mote (without embedded sensors), while the other is equipped with an embedded sensor suite (i.e. humidity, temperature and light sensors). The usage of these standards and the native integration of humidity, temperature and light sensors ensures a flexible interconnection with other devices which allows a wide range of applications in mesh networks. With the help of the TinyOS operating system, the Telos manages also wireless protocols for communication. Figure 9.2 shows the internal architecture of the Telos revB.

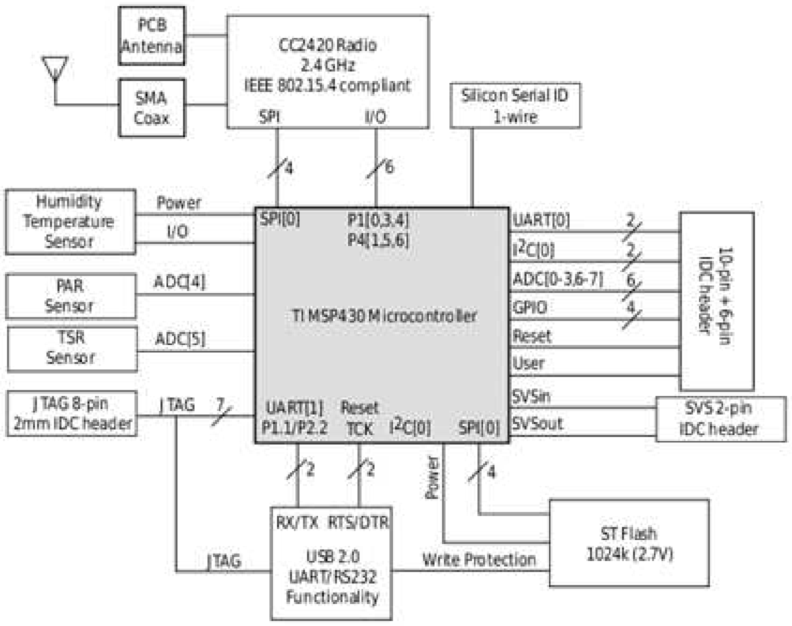


FIGURE 9.2: Telos revB architecture

The TelosB is equipped with the following components:

- *Power*: the power supply of the Telos is given by two AA-size batteries. However, for the programming phase of the microcontroller and the external flash, a voltage of at least 2.7V is necessary. If the Telos node is connected to the USB port, the power supply for programming and for its operation is given by a computer; in this case the operating voltage is 3V and batteries are not needed. Inside the node the input voltage must not exceed 3.6V, otherwise it could damage the radio module or other components.

- *Microcontroller*: the low-power needed by the Telos is mainly due to the low power consumption of the Texas Instruments MSP430 F1611, with a RAM of 10kB and a flash of 48kB. This 16-bit RISC processor allows to operate for years with a single pair of AA batteries. The MSP430 has an internal digital control oscillator (DCO), which can operate up to 8MHz. The DCO can be activated from sleep mode in 6μs, but typically it requires only 292ns at common temperature. When the DCO is off, the MSP430 works with an external clock to 32768Hz. Although the frequency of the DCO changes with the voltage and the temperature, it can be calibrated using an 32kHz oscillator. In addition to the DCO, the MSP430 has an 8-ports Analog Digital Converter (ADC); one of the internal ports can be used to read and monitor the battery voltage through a specific purpose sensor. There are a variety of peripherals including SPI, UART, digital I/O ports, watchdog timer, and Timer. The F1611 includes also a DAC with two input of 12 bits, a voltage supervisor, and 3 ports for the DMA controller. Figure 9.3 shows the main components of the MSP430 controller.

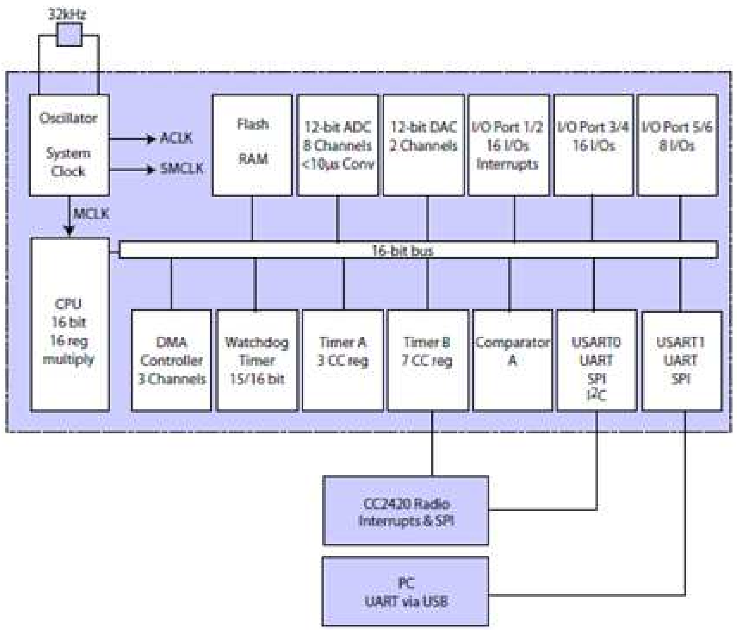


Figure 9.3: MSP430 F1611 microcontroller

- *Radio*: the Telos is equipped with the Chipcon CC2420 radio module. The CC2420 adheres to the IEEE 802.15.4 standard and provides some functions at both PHY and MAC layers. The CC2420 is highly configurable in order to support several applications. The CC2420 is controlled by the MSP430 microcontroller through the SPI port, a set of digital I/O lines and interrupt lines. The radio module can also be switched off by the microcontroller in order to limit energy consumption. At last, the output power is programmable, allowing a variable transmission radius.

- *Antenna*: the Telos has a built-in antenna at 2.4GHz which allows transmission of up to 125m, but can mount an external antenna (SMA connector) to perform those functions that are not supported by default.
- *External Flash*: the Telos Revision B uses the 40MHz ST M25P80 Flash for external data. The flash has a capacity of 1024kB and is separated into 16 segments, each one of 64Kb.
- *Humidity and temperature sensors*: the humidity/temperature module is optional and is produced by Sensirion AG. The SHT11 and SHT15 can be mounted directly on Telos. Both sensors are calibrated and produce a digital output thanks to an integrated AD converter. The sensor is manufactured using a CMOS process and is coupled with a 14-bit A/D converter.
- *Light sensor*: this module is also optional and can be mounted directly on Telos. It has two photodiodes produced by Hamamatsu Corporation. The default diodes are the S1087 for the detection of photosynthetically active radiation, and the S1087-01 to detect the visible spectrum, including infrared.
- *Connectors Expansions*: the Telos has two expansion connectors and a pair of jumpers that can be configured so that additional devices (analog sensors, LCD display and digital devices) can be mounted on the board. On the opposite side of the USB connector, there are a 10-pins and a 6-pins connectors (Figure 9.4). The 10-pin connector, as well as in Telos revision A, provides additional digital and analog inputs; the additional 6-pin connector provides access to exclusive features of the revision B, such as the two additional A/D inputs which can be reconfigured via software.

9.2 Modeling approach

A sensor network shall be properly designed in order to estimate and assess critical characteristics (e.g. dependability, performance, performability, security, etc.) with reference to networks parameters (e.g. node density, node positions, application algorithms, etc.). This leads to two main questions: how to assess such measures? and by what means? the answer is simple. If the system exists and is running, then the easiest way is to apply the direct measurement approach, measuring interest parameters directly on the

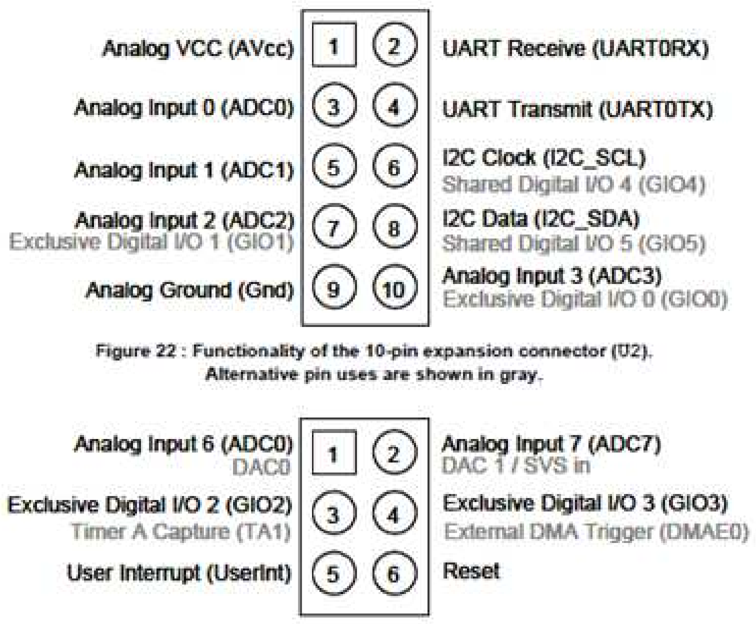


Figure 9.4: additional inputs of the Telos revB

real system. If the system does not exist or is not complete (this happens for example during the design of a new sensor network), it becomes almost necessary to adopt a formal model-based approach able to predict these parameters. This Section apply the proposed approach, realizing a general model of a sensor network, according to the component-base vision adopted by OsMoSys. After the definition of the sensor node the method runs along four phases:

1. decomposition into components: starting from the structure of the sensor network, individual elementary subcomponents have been identified. Three modeling levels have been defined: (1) software level, (2) hardware level, (3) network level. In particular the first can be detailed as on-board software and timer; the second as the composition of bus unit, microcontroller unit, storage unit, radio unit, sensing unit and additional inputs; the last level is given by the network subsystem. At the end 9 subcomponents have been identified, each one is translated into a submodel;

2. characterization of the individual components: for each submodel, the proper interfaces characterizing the components (in terms of interface elements and data types) have been defined;

3. definition of the compositions: the composition relationships between the individual components have been defined in order to carry out the composed system model;

4. definition of the behaviors: on the basis of the model complexity and of the properties to measure, the formal language is chosen.

This modeling approach highlights an important advantage that resides in the possibility of creating a single submodel changing the formalism and the development technique, without modifying interfaces, behaviors and/or compositions. Figure 9.5 shows the 9 subcomponents, highlighting for each one the proper interfaces.

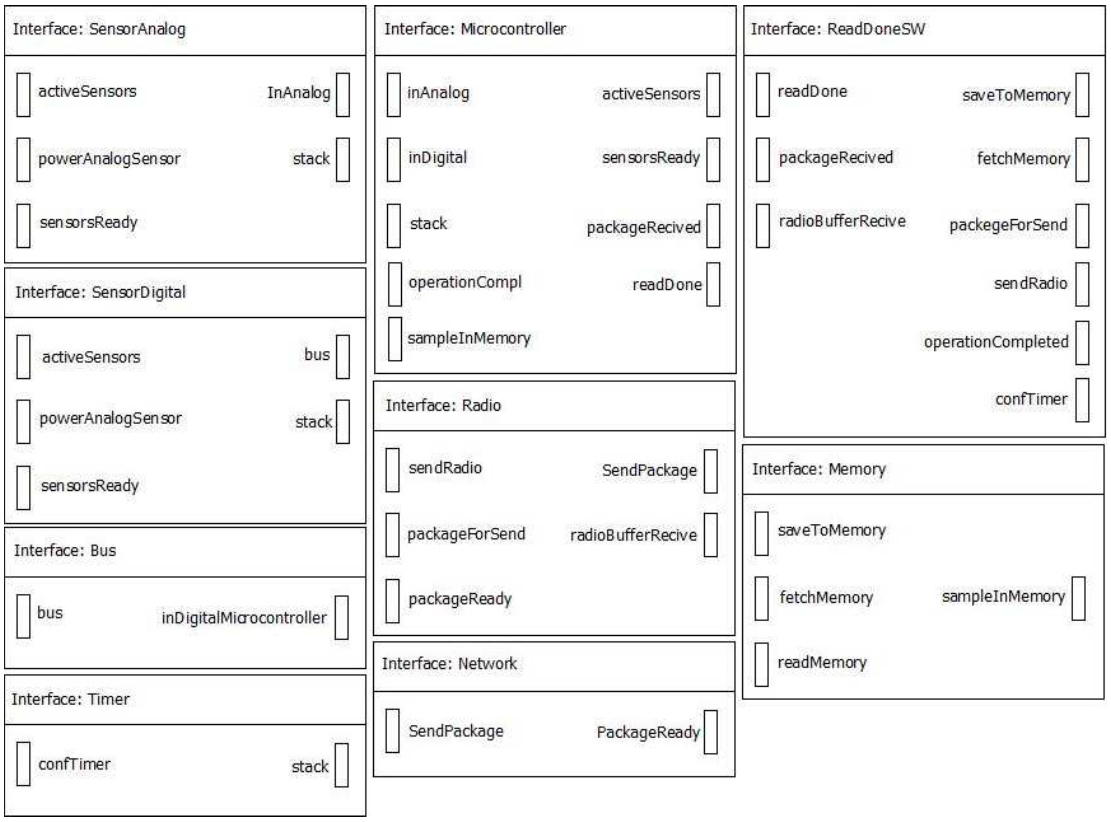


FIGURE 9.5: Reference modeling schema of WSNs

9.3 Generalization techniques in SAN formalism

SAN formalism has been chosen for the great modeling power and for the possibility to introduce code. Furthermore the potentialities of the Möbius modeling frameworks are exploited during model development, in particular introducing extended places which allow to manage complex structured data, even if with some limitations. Several lines of code in gates are necessary to manage complex data structures, this guarantees not only flexibility and scalability increments, but also location of useful information of the sensor node.

The model of a sensor network need to deal with a fundamental issue: the components heterogeneity. In fact, a single network can integrate nodes with different software and hardware, or nodes that have substantially the same hardware but with different configurations. In the following paragraphs, the most important aspects on different hardware and software of a sensor node are addresses and a solution, exploiting the SAN formalism, is given.

9.3.1 Generalization of features

In a sensor node each components is characterized by specific properties that make it dissimilar from another component of the same type. Being a network composed of a large number of sensor nodes, each component of a node need to be "configured" in order to express his particular behavior. The following list give an idea of the parameters needed by each one:

- Analog sensor:
  - mode: synchronous or asynchronous;
  - StartUp delay: time needed to activate the sensor;
  - sampling period: useful if the sensor operates in synchronous mode;
  - power required to generate a sample;
- Digital sensor:
  - in addition to the properties of the analog sensor, it has the following properties
  - buffer size of pending requests, if present;
  - reference to the A/D converter;
  - resolution for the digital conversion, plus any additional fiels (e.g. parity, CRC);
  - conversion delay which depends also on the sensor generating samples;
- bus:
  - transmission delay;
- microcontroller:
  - A/D conversion delay (for operations performed by the integrated A/D converter);
  - resolution of digital samples;
  - CPU frequency;
  - operation stack size;
- memory:

  - number of memories per node;
  - reading and writing delays for each memory;

- radio:
  - size of send buffer;
  - size of receive buffer;
  - reading and writing delays;
  - sending times (different from the link delays);

- ReadDoneSW:
  - number of measures to be stored before the generation of a packet;
  - timers;
  - packet size;
  - type of node (peripheral or sink);

- Network:
  - network topology;
  - listen time (with CSMA/CA protocol);
  - collision probability;
  - link delays;
  - maximum waiting time for acknowledgments;
  - maximum number of transmission attempts.

These informations are assigned to each atomic model using some configuration places: complex data structures have been defined and each atomic model reads its own parameters. The values are initialized using specific custom initializations, and can be dynamically modified to model runtime reconfigurations (e.g. the sampling period of a specific node can be time-dependent, the processing time of a sample can increase after the adoption of a different cryptography algorithm, etc.).

9.3.2 Generalization of behaviors

To avoid the limitation of models to represent specific components, some generalizations have been introduced. Also the configuration of the behaviors can be specified through initialization code and can be dynamically changed. Note that the introduces generalization are configurable at node level: it is not necessary that all nodes share the same behavior. These generalizations have been solved using complex model structures and code. In detail:

- Analog sensor: it can operate in synchronous or asynchronous mode;
- Digital sensor: as analogue, digital sensor also can operate in synchronous and asynchronous mode; digital sensors can have size-variable buffers of pending requests; a certain number of sensors can share one A/D converter;
- Bus: it is characterized by the parallelism, from which transfer time can be evaluated;
- Memory: each memory unit can be structured as a variable set of physical memories, each of them is characterized by its own access times.

9.3.3 Identification of a single replica

The specification of different characteristics for each model need to identify a each atomic in a replica: for this reason a numeric id is necessary. A possible solution, observed in [71], consists in a small model that is able to set the id instantaneously, at the start of the analysis. As an example, Figure 9.6 shows the model developed for digital sensors: *CountDigitalSensors* contains many tokens as the number of replicas; this place is shared with all replicas. A token in the *Start* place means that the id has not been set: the firing of the instantaneous activity *setupIndex* enable the execution of the *setIndex* output gate, which properly sets the id number, storing tokens in the *indexDigitalSensors* place.

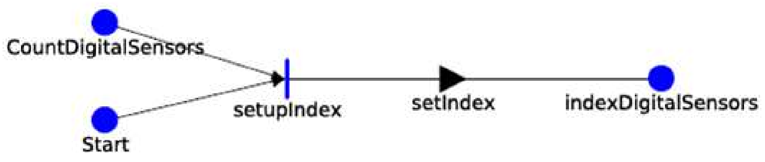


FIGURE 9.6: Identification of a single replica

9.3.4 Submodel connectivity

Any component-based model needs to allow the composition and the communication between atomic models. The basic communication methods, allowed also in the SAN formalism, is the place superposition. The issue becomes more complicated when, instead of connecting two simple components, you must connect a replica of a replica of a component with another component (e.g. replica of sensor model, replicated for all the nodes). In this case the place superposition is not enough. As an example, suppose that we need to model 3 nodes, where the first is equipped with 5 sensors, the second with 2 and the third with 3. And it's not enough: if we have to explain that the fourth sensor of the first node is connected to the third analog input of the microcontroller? It becomes more difficult!

A first solution, adopted in the previous Chapter, consists in the modeling of the most complex structure, after which the portion not needed in the single replica are "switched off". In the described example it would work as: we develop a model of the node with 5 sensors (the maximum number in the example) and, while the first node uses all them, the second uses only two and the third three. This solution implies an excessive resource waste, that can make the solving phase impracticable, given the high heterogeneity of a network.

For this reason, another solution has been implemented. As explained in the previous paragraph, each replica has its own identifier. Taking advantage of this, a model can access to one or more fields of vector (using the id as pointer) and read specific values. In relation to the previous example, between the configuration parameters of a sensor, it have been stored the index of the microcontroller to which it is connected. We need a n-dimensional vector, where n is the number of nodes of the global network. Each field of this vector is x-dimensional vector (where x represent the microcontroller inputs), and each single field is a data structure (Figure 9.7).

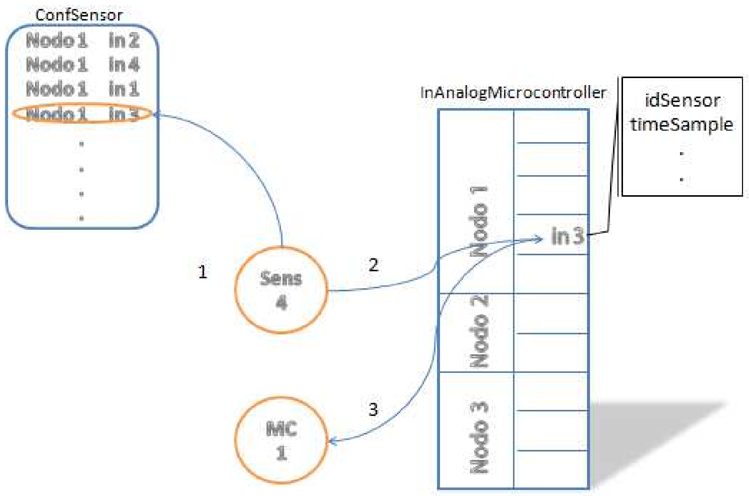


FIGURE 9.7: Reference modeling schema of WSNs

Hence the sensor 4 and the node 1 works as follows:

1. the sensor 4 reads from *ConfSensor* which is its identifier extracting information about the node and the input (Node 1 and In 3);

2. the sensor 4 write on *InAnalogMicrocontroller*, through the pointer taken from *ConfSensor*, the informations of the structure relative to node 1 and input 3;

3. the microcontroller, knowing its identifier, reads from *InAnalogMicrocontroller* in first position and takes the measurement from the third input.

This method, also if presented as a solution to connect analog sensors and microcontroller, has been adopted for any other connections between replicas.

9.4 Modeling WSNs

This Section shows the SAN models developed according to the well-defined interfaces of each component, using techniques and methods showed in the previous Section.

9.4.1 Timer

The timer, presented in Figure 9.8, it works as follows: the input gate IG1 read from the confTimer if it is active or not; in the positive case, the timed activity Sampling is invoked after each period defined in confTimer; the output gate GenerateRead generates a Read operation, corresponding to the request for a sample, which is stored in the stack. Each timer generates a Read for the device with the same id.

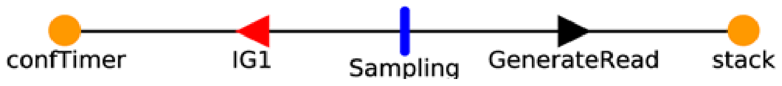


FIGURE 9.8: Timer model

9.4.2 Microcontroller

The model of microcontroller can be structured as two distinct sections that work independently: the stack management and the read done generation. The first is showed in the Figure 9.10. The input gate *operation* checks if there are request for sampling on the stack. In the positive case the activity *delayinit* locks the CPU and fetches the next operation to be performed from the stack. Within the stack you can find three types of operations: (1) request for generation of a sample, in which case the specific sensor is activated through the extended place *activateSensors*; (2) a sensor is ready to send a measurement to the microcontroller, in which case an enabling signal is sent through the *sensorsReady* place; (3) a packet have been reached from the network.

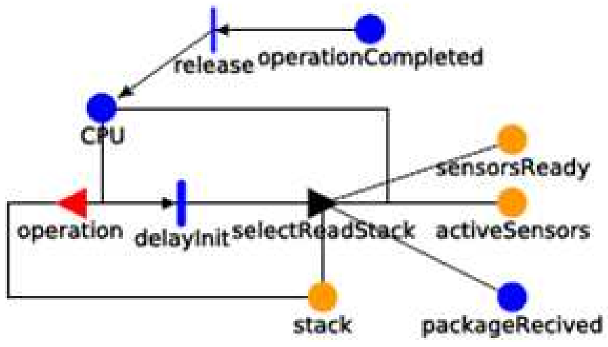


FIGURE 9.9: Microcontroller model (2)

The second portion of the model (Figure 9.10), is able to generate read done. If a digital sample has been received it shall be only pre-processed, while in case of analog samples the process is identical with the exception of a digital conversion (and additional time). Of course, once converted, the sample is stored in the *SampleDigital* place. The output Gate *generateReadDone* puts sensor identifier in the *readDone* place.

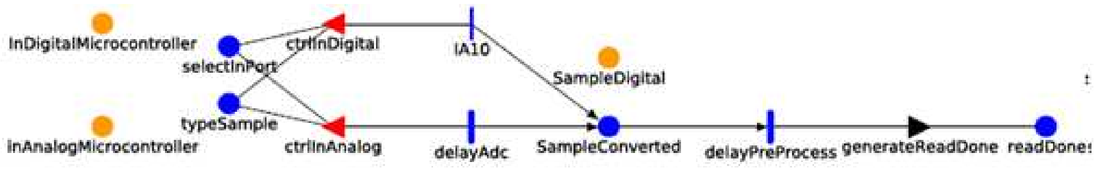


FIGURE 9.10: Microcontroller model (2)

9.4.3 ReadDoneSW

The *ReadDoneSW* sends a packet to the next hop after the sampling of a given number of samples. This model is structured as two independent sections. A first section (Figure 9.11), checks how many samples have been generated by a device after receiving a read done: when the microcontroller sect *readDone*, input gate *ctrlReadDone* activates the transition *IA2* can fire. After its fire, the output gate *selectOP* can insert a token in *SaveToMemory* (asking for the storage of the sample) or insert in *fetchNumSample* the number of samples that shall be sent. In the latter case, after the complete read, the input gate *endFetch* activate the transition *IA4* which ask radio for sending.

The second section is the management of packets receivedfrom the network (Figure 9.12). When a token is present in *packageReceived*, the output gate *ctrlRadioIn* occurs if the board is a sink node or not. If it is not a sink, the packet can be memorized or forwarded. In the other case, the packet is handled and eliminated from the model.

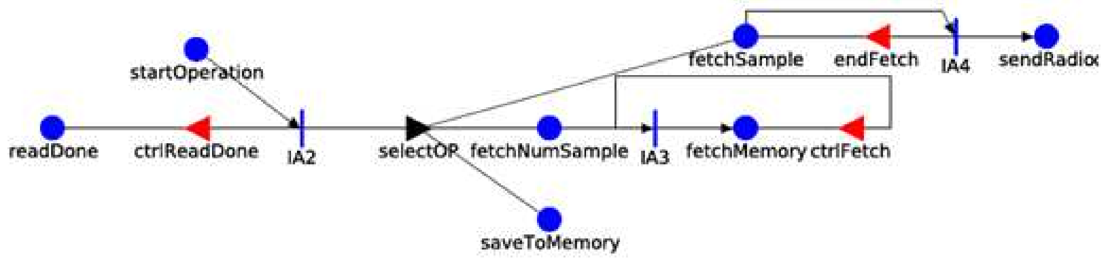


FIGURE 9.11: ReadDoneSW model (1)

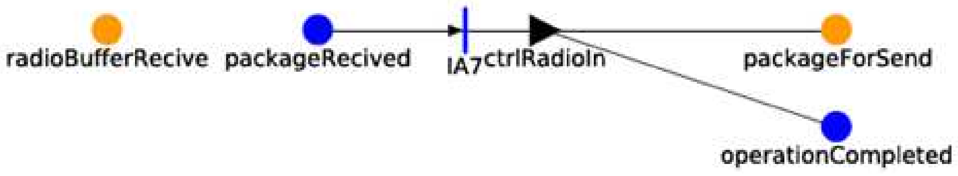


FIGURE 9.12: ReadDoneSW model (2)

9.4.4 SensorAnalog

The component SensorAnalog, depicted in Figure 9.13, is modeled to operate in two ways: synchronous (generating samples each period, defined in *cycleSampling*); asynchronous (waiting to receive a request from the microcontroller through *activeSensors*). In both cases a startup delay represents the time required to activate the hardware component, when it is in the sleep state. The output gate *pendingRequests* brings the sensor in the active state and inserts a request in *bufferPendingRequests*. After the sampling delay it (1) generates an analogue measurement and stores it in *SampleAnalogGenerated*; (2) inserts in the stack operation the signal of a ready measure; (3) waits to receive a signal from the microcontroller on *sensorsReady*; (4) moves the measure to *inAnalogMicrocontroller*; (5) shut off the device if there are no more pending requests.

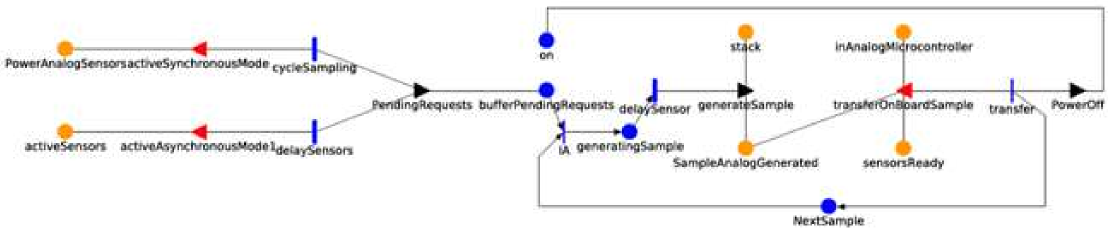


FIGURE 9.13: SensorAnalog model

9.4.5 Memory

Memory model (Figure 9.14) allows the operations on physical memories: write, read and fetch (read and delete). In the first case, after the firing of *delayCpuStored*, *save* verifies which is the free memory to store the sample; if no space is available the sample is eliminated, properly logging this information. In the fetch and read cases, a sample is transferred into *sampleInMemory* if present.

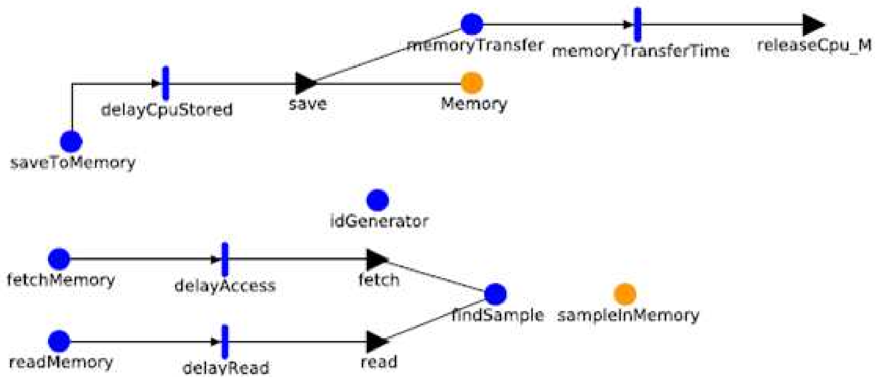


FIGURE 9.14: Memory model

9.4.6 Radio

The Radio component shown in Figure 9.15, performs radio functions: when a token is present in *sendRadio*, the output gate *releaseCPU_R* puts the complete package inside the *radioBufferSend* and finally releases the CPU. Then the radio module send the packet to network elements through *SendPackage*. If a packet is received from the network within *PackageReady*, it is taken and inserted directly within *radioBufferReceive* generating an entry within the stack.

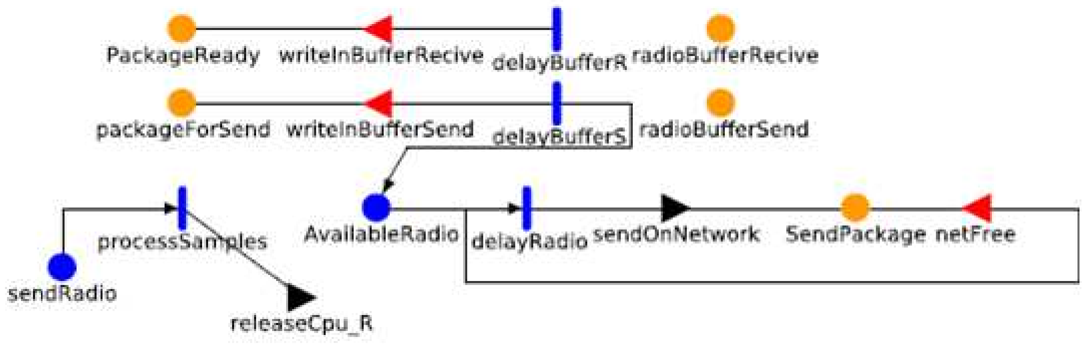


FIGURE 9.15: Radio model

9.4.7 Bus

The bus model is shown in Figure 9.16: when a digital measure is present in the *bus* place, *ctrlBus* activates the transition *delayTransfer*. When it fres, *ctrlBus* moved the sample in *inDigitalMicrocontroller*.

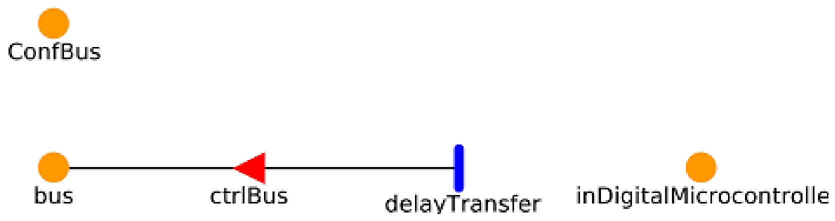


FIGURE 9.16: Bus model

9.4.8 Network

The networks protocol adopted in a WSN is the ZigBee, which requires low power consumptions allowing low transfer rates. The network module, depicted in Figure 9.17, models this protocol. When an incoming packet is present, *ctrlNetworkIn* and *AccessChannel* model the CSMA/CA protocol. If nearest nodes are transmitting, the transmission is delayed since the channel is busy. If the next hop is receiving but the transmitting node does not sense this, it tries to send the package and, with a probability *CollisionProbability* the message is lost, otherwise it is correctly delivered. When the next hop receives the packet in *networkOut* it sends an ack to the transmitting node; otherwise the transmitting node retries to send the packet for a finite number of attempts.

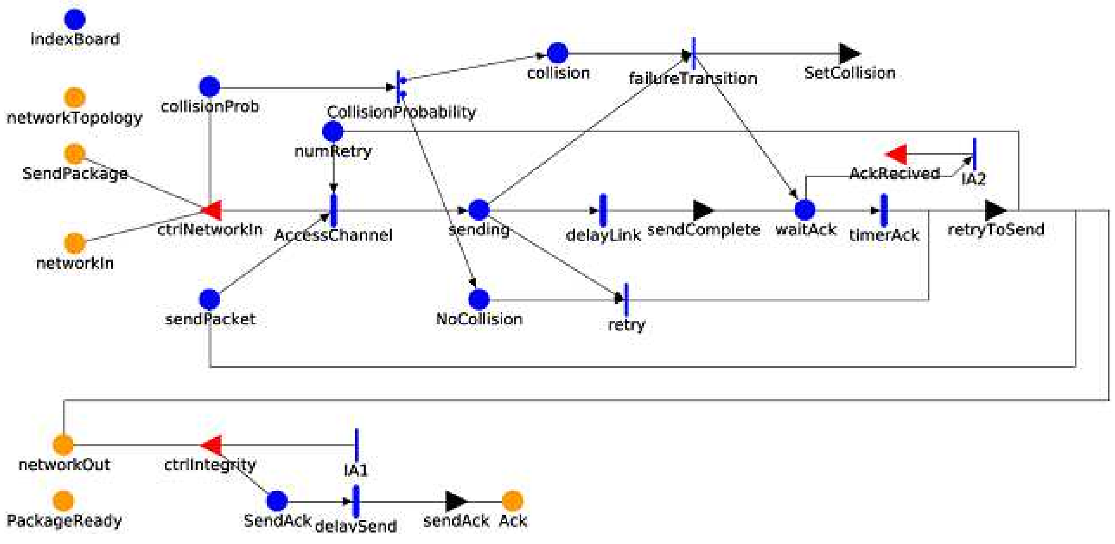


FIGURE 9.17: Network model

9.4.9 SensorDigital

The component SensorDigital presented in Figure 9.18, has a similar behavior to SenasorAnalog. The substantial difference is in the generation of the measure. In fact, while for the analog measurement device is generated automatically after a given delay, in SensorDigital, it must wait for an A/D conversion. Multiple devices can share the same A/D converter: *inADC* store a n-dimensional vector, where n is the total number of A/D converters.

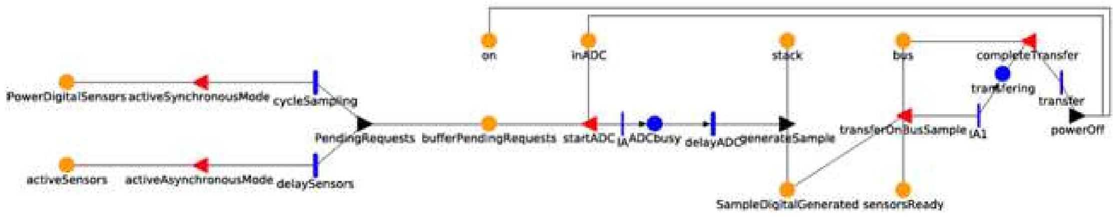


FIGURE 9.18: SensorDigital model

9.4.10 Composed model

The composed model cannot be the replication of a board since, as said previously, this solution lead to an excessive resource waste in the solution phase. Suppose, for example, to model a network of 100 nodes, of which 99 are composed of one device analog, one digital and one bus, and the last node is composed of 50 analog devices, 100 digital and 30 bus. It would mean a composition of 100 nodes each consisting of 50 analog devices, 100 digital and 30 bus. The model would continue to operate but during the simulation, the software should allocate data structures in memory and submodels, risking that it runs out very quickly to the point of not being able to even start the simulation. A solution adopted for this purpose, has been to exchange Rep and Join operators as shown in Figure 9.19:

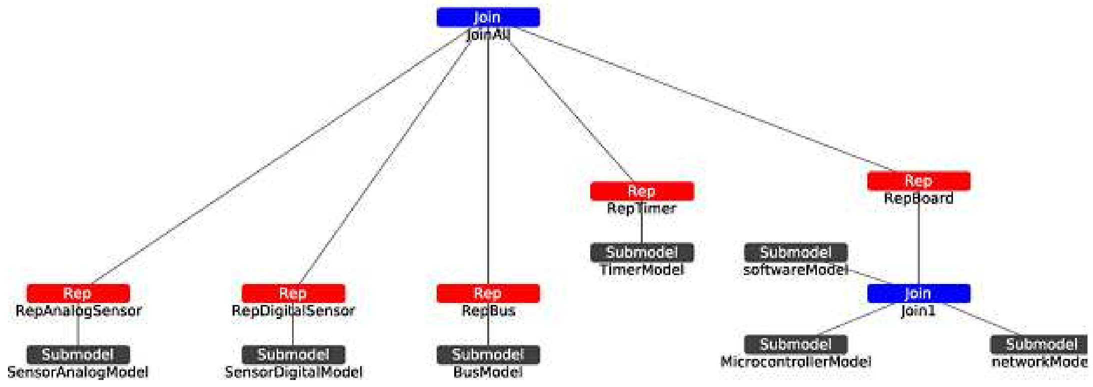


FIGURE 9.19: Composed Model

In this way all devices and buses, are replicated only as many times as necessary. For the specific example we will have exactly 149 analog device models, 199 digital device models and 129 bus models instead of 5000 analog device models, 10000 digital device models and 3000 bus models.

9.5 Obtained results

After the successful model validation, conducted with real data coming from real systems, the models have been used to assess the performability of the following case study: we want to model a monitoring system that need to constantly control the temperature in many points of a surface. It is necessary to consider two performance measures such as the number of packets that the sink node is able to process and, the physical delay that elapses between the instant in which the measurement is sampled and the one in which it is processed.

Three different network topologies have been evaluated; results are combined with the aim to evaluate the following QoS formula:

$$QoS = \alpha \cdot (1 - \%LP) + \beta \cdot (1 - \frac{PD - PD_{min}}{PD_T}) + \gamma \cdot (1 - \frac{SP - SP_{min}}{SP_T})$$

where:

- $\%LP$ is the percentage of lost packets;
- PD is the physical delay;
- PD_{min} is the technological delay;
- PD_T is the tolerable physical delay;
- SP is the sampling period;
- SP_{min} is the technological sampling period;
- SP_T is the tolerable sampling period;
- α, β, γ are multiplicative factors.

The reference system is the Telos revB node. The number of nodes varies from 10 to 60 and the sampling period varies from 1 to 15 seconds. The transmission delay is 1ms, while the processing time at sink has been assumed equal to 100ms.

Topology 1: all nodes are directly connected to the sink

In the Figure 9.20 the percentage of packets processed by the sink node (ratio of those received by the sink divided by the total number of generated packets) is expressed varying the sampling period of each temperature sensor located on each node. As can be seen from the graph, with a short sampling period, the percentage of packets processed by the sink node is relatively low (for a period of 1s it is 16%), while increasing sampling periods increases until reaching approximately 94% (at a sampling period of 15s).

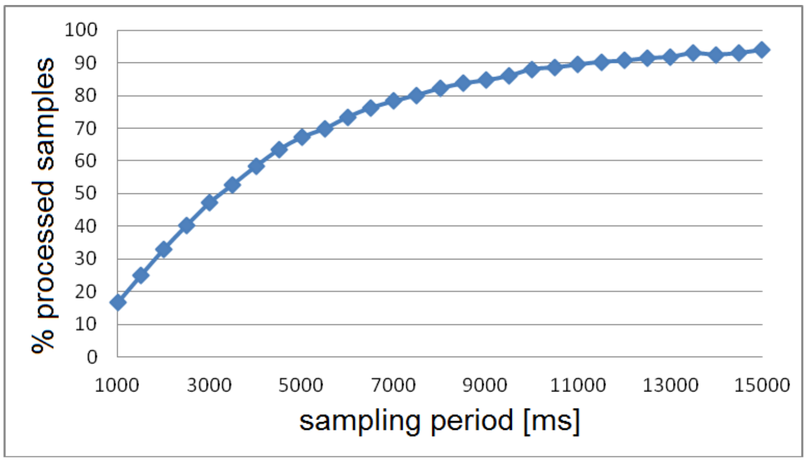


FIGURE 9.20: Topology 1: %processed samples vs sampling period

This phenomenon is justified by the limited size of the buffer of the radio module. When packets are generated with a frequency that does not give the possibility to insert all them in the buffer, they are discarded. In fact, for a short sampling period, the sink node is busy, increasing also physical delays (mean time elapsed between the instant of measure generation and the processing at sink). In Figure 9.21 this trend is depicted. From this graph, we observe an increase of 13ms of the physical delay with respect to the one showed with a sampling period of 1 and 15 seconds. In fact at a sampling period of 1 second a lot of packets are discarded, while those sent are characterized by a short temporal delay.

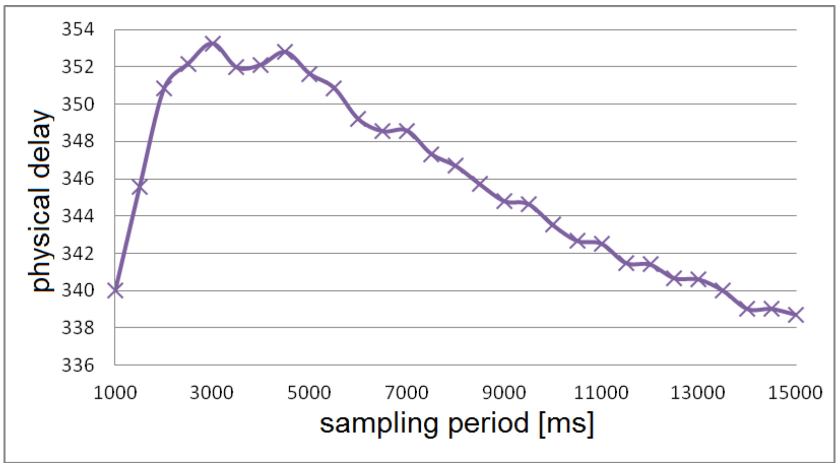


FIGURE 9.21: Topology 1: physical delay vs sampling period

A further analysis is carried out by setting the sampling period equal to 1s and varying the number of active sensors (from 10 to 60). The trend of processed samples and delays are showed respectively in Figure 9.22 and Figure 9.23.

Topology 2: unbalanced tree

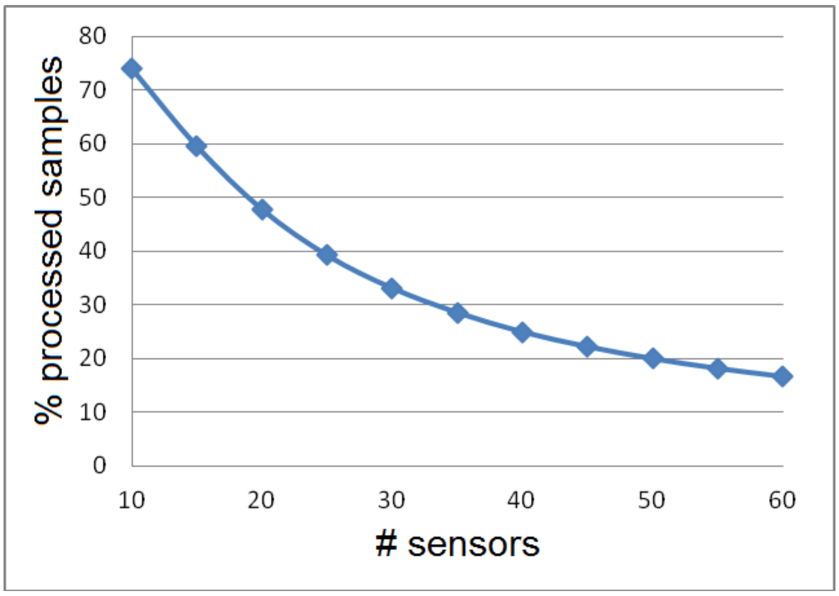


FIGURE 9.22: Topology 1: %processed samples vs #sensors

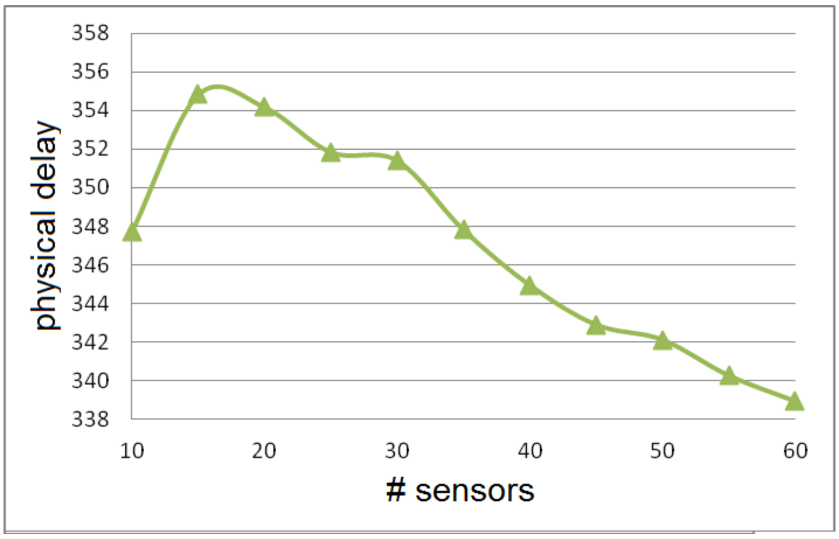


FIGURE 9.23: Topology 1: physical delay vs #sensors

In this case, 30 nodes are connected to a forwarder node and the remaining 30 are connected directly to the sink. Each forwarder node accumulates 5 measures, evaluate their average, send this value to the sink node. In this way, a lesser number of measures arrives at sink node.

Figure 9.24 shows the percentage of packets processed by the sink while Figure 9.25 shows the physical delay, varying the sampling period. An overall performance improvement is highlighted: with a sampling period equal to 4s the percentage of received packets reaches 80%. In the worst case the physical delay increases linearly with the increasing of the sampling period: this is due to the fact that the forwarder node needs to wait 5 samples before sending the measure to the sink.

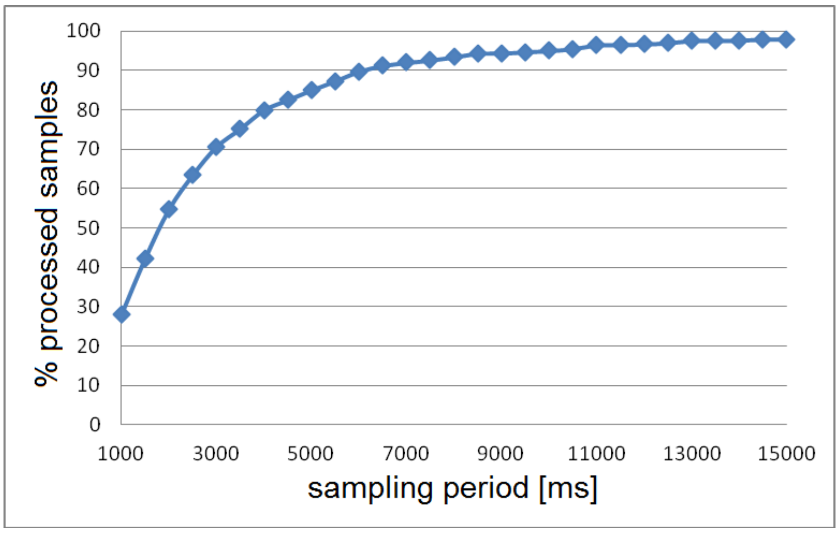


FIGURE 9.24: Topology 2: %processed samples vs sampling period

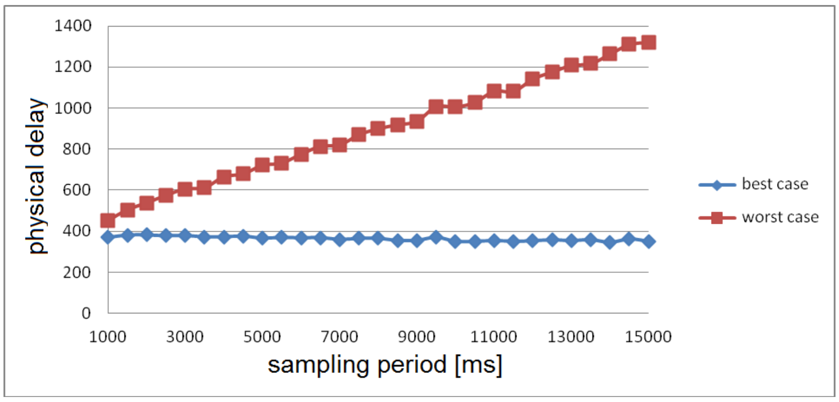


FIGURE 9.25: Topology 2: physical delay vs sampling period

The percentage of processed samples with respect to the number of sensors is higher than that coming from the other topology: Figure 9.26 shows this trend. With 60 active sensors, for example, the 28% of processed packets is measured while, with the previous topology, this value does not exceed the 16%. Figure 9.27 shows that the upper limit of physical delay decreases with the number of sensors until it reach an asymptotic trend.

Topology 3: balanced tree

In this case, 12 groups of 5 nodes are connected to 12 forwarders, and each forwarder sends packets to the sink node. Also in this case each forwarder node accumulates 5 measures, evaluates their average, sends this value to the sink node.

Figure 9.28 shows the percentage of packets processed by the sink while Figure 9.29 shows the physical delay, varying the sampling period. This topology does not increase the overall performances, indeed the worst case of physical delay is worse than the previous one.

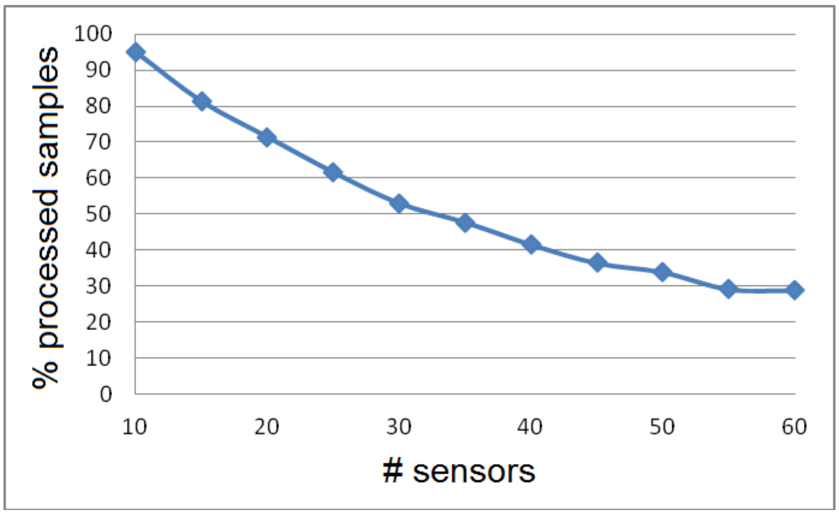


FIGURE 9.26: Topology 2: %processed samples vs #sensors

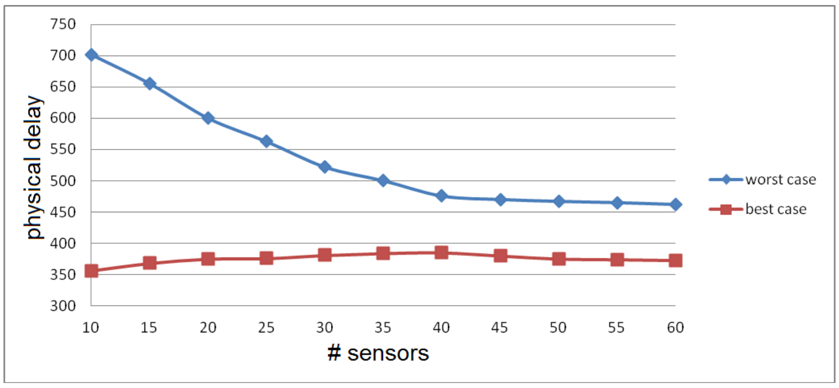


FIGURE 9.27: Topology 2: physical delay vs #sensors

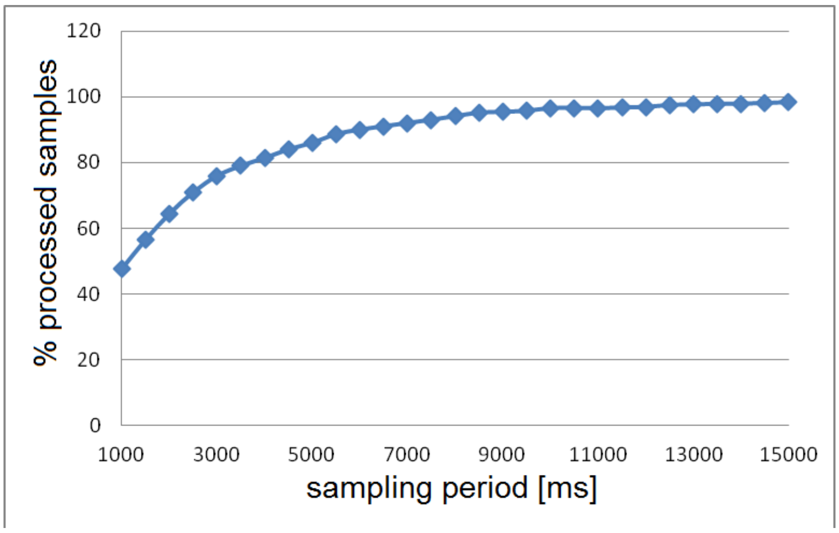


FIGURE 9.28: Topology 3: %processed samples vs sampling period

Figure 9.30 and Figure 9.31 show the percentage of processed samples and physical delay with respect to the number of sensors

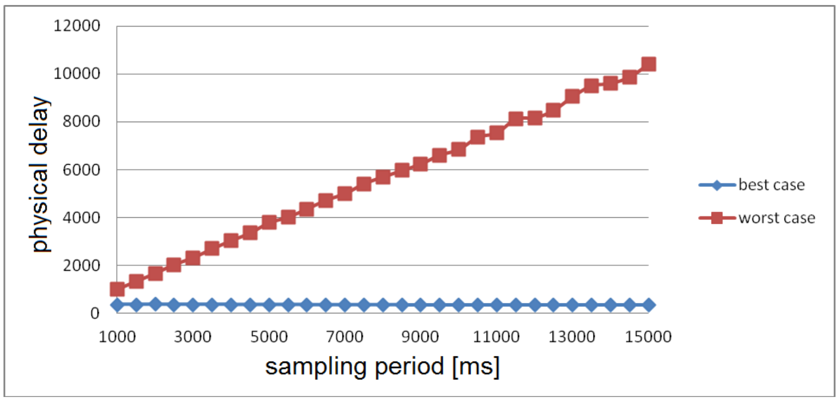


FIGURE 9.29: Topology 3: physical delay vs sampling period

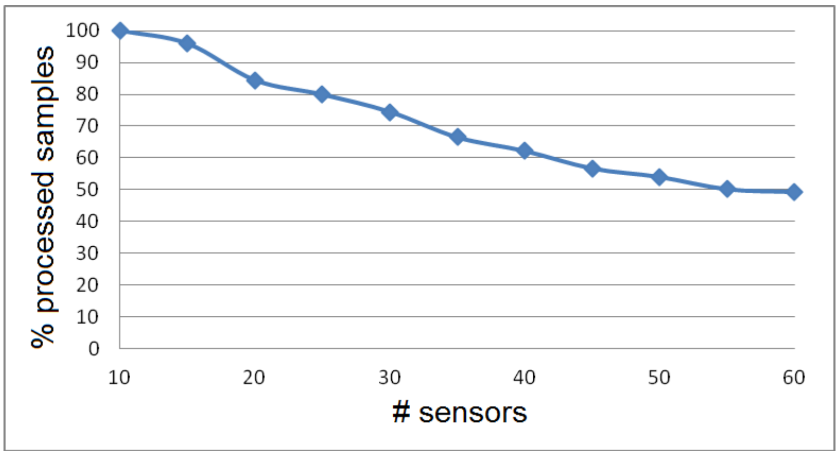


FIGURE 9.30: Topology 3: %processed samples vs #sensors

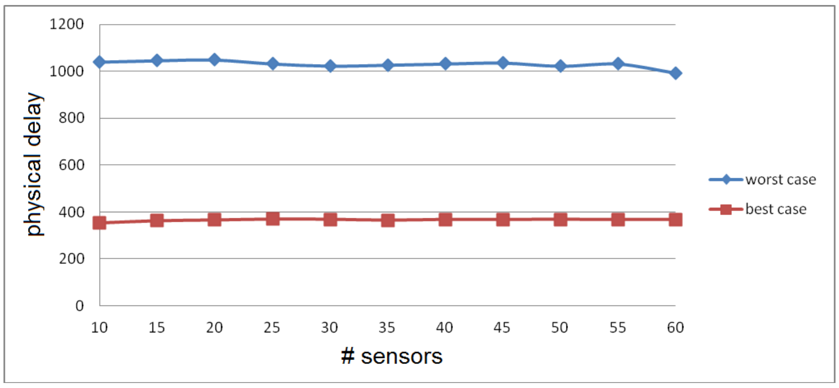


FIGURE 9.31: Topology 3: physical delay vs #sensors

QoS evaluation

This paragraph evaluates the QoS coming from the topologies before presented. Figures from 9.32 to 9.37 depict the trends of the QoS varying the network topology, the sampling period and the number of sensors.

From these graphs it is clear that the star topology suffers of the known bottleneck limit. So the best topology is a layered network. The proposed approach is able to quantitative

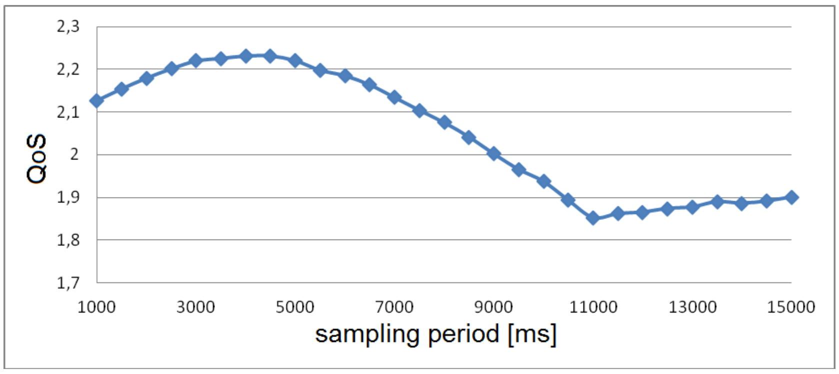


FIGURE 9.32: Topology 1: QoS vs sampling period

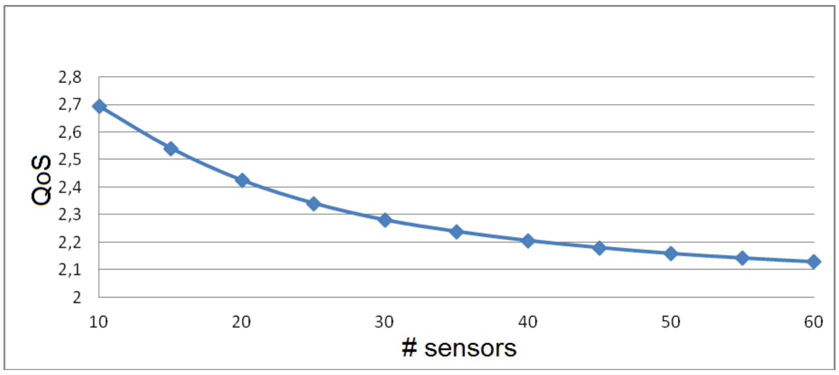


FIGURE 9.33: Topology 1: QoS vs #sensors

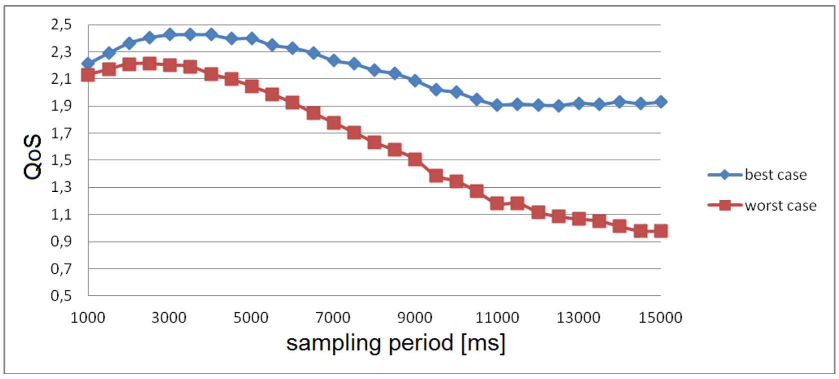


FIGURE 9.34: Topology 2: QoS vs sampling period

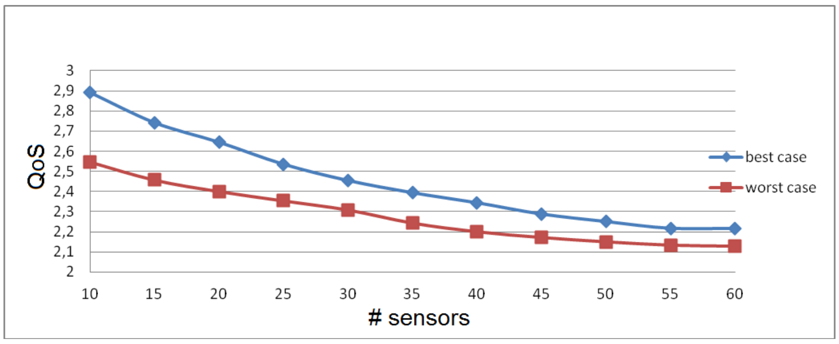


FIGURE 9.35: Topology 2: QoS vs #sensors

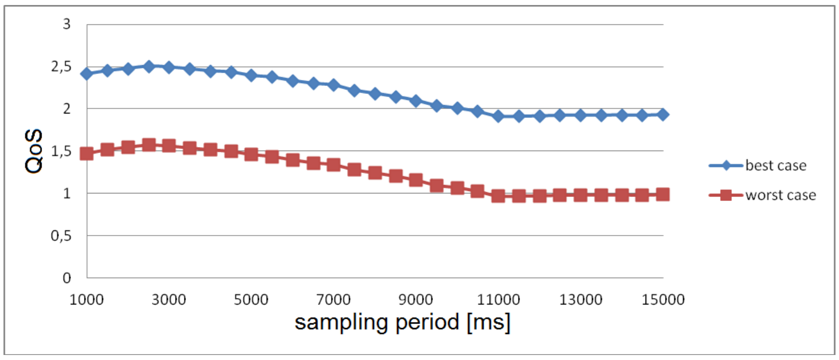


FIGURE 9.36: Topology 3: QoS vs sampling period

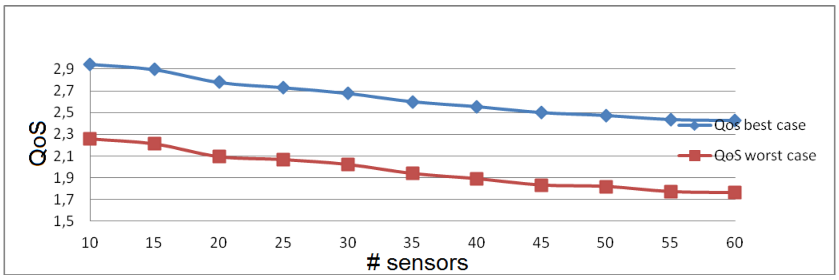


FIGURE 9.37: Topology 3: QoS vs #sensors

evaluate different topologies, allowing to discriminate between different alternatives, such as balanced and unbalanced trees. If we analyze the best cases of Figure 9.34 and 9.36, we appreciate that the highest quality is reached with a sampling period equal to 3s. In this case the balanced tree (Topology 3) assures a QoS of 2.5, higher than that observed with the unbalanced tree that is 2.42 (Topology 2). But if we consider also the worst case, we observe that the balanced tree guarantees a lower QoS than that of the unbalanced tree (1.6 and 2.2 respectively). At last we can say that the choose of the best performing topology depend on a set of parameter (e.g. number of nodes,sampling period, etc.), the evaluation of the best case does not imply the finding of the optimal solution since the worst case can be critical. In fact, unbalanced trees can guarantee, in certain conditions, QoS levels higher that those obtained with balanced trees.

Chapter 10

Conclusions

This thesis has presented a novel approach for quantitative analysis of critical systems which integrates Model-Based analysis (performed through formal models) with Model-Driven paradigm and techniques, exploiting benefits of multiformalism, submodel composition and automatic model generation. Quantitative analysis may be performed by generating formal models from high-level specifications, through the definition of Model-to-Model and Model-to-Text transformations. An important point is related to the enabling techniques that make the methodology more attractive to the industrial settings: MDE approaches are rapidly evolving since they promise modelers reduction of inconsistencies that can occur between the artifacts created in a manual development and analysis process. The methodology has been applied to three different critical systems in order to show its effectiveness: railway systems, conferencing systems and wireless sensor networks.

The application of the proposed approach to the real world systems make necessary the development of two specific high-level modeling language. The first is DAM-Rail, that is a specialization of the DAM language, and is able to model dependability aspects of railway systems. The second is CIP_VAM that want to model vulnerability of critical infrastructures, belonging to different application domains. Both of them have been developed, into a cost-effective way, using the UML profiling technique. Furthermore new methods, formal operators and techniques are addressed in order to support not only the high-level modeling but also the formal stage.

The proposed technique have been applied to different situation coming from three different applicative domains. In detail performance and performability studies have been conducted for railway systems, conferencing systems and wireless sensor networks; the availability quantitative evaluation of a modern metro driverless vehicle is shown as well as the vulnerability modeling of a railway station.

Future research, will be focused on the extension of the propose languages: DAM-Rail can be extended for safety modeling and analysis while CIP_VAM can be adopted for cyber security analysis. Furthermore automatic generation of Model Classes from Model Templates through the definition of *instancing functions* will be fully implemented.

From a more theoretical aspect the foundation of DSML definition and the relationships between Model-Driven Architecture and Model-Driven Engineering will be investigated: since DSML development is still an emerging discipline with few established guidelines and patterns, we experienced that the relationships between the standard OMG's Model-Driven Architecture and the Model-Driven Engineering meta-modelling stacks have not been clarified enough.

Future research efforts will investigate also on the application of the same methodology to other case studies coming from the addressed or from other domains.

Bibliography

[1] J. Rushby. Critical system properties: survey and taxonomy. *Reliability Engineering & System Safety*, 43(2):189 – 219, 1994.

[2] J.C. Laprie. *Dependability: Basic Concepts and Terminology, Dependable Computing and Fault-Tolerance.* Springer-Verlag, 1992.

[3] A Avizienis, J. C. Laprie, and B. Randell. Fundamental concepts of dependability, 2001.

[4] A. Bondavalli. *L' analisi quantitativa dei sistemi critici.* Esculapio, 2011.

[5] W. E. Vesely, F. F. Goldberg, Roberts N. H., and Haasl D. F. *Fault Tree Handbook, Technical report.* 1981.

[6] T. Murata. Petri Nets: Properties, Analysis and Applications. In *Proceedings of the IEEE*, pages 541–580, April 1989.

[7] M. A. Marsan. Stochastic petri nets: An elementary introduction. In *In Advances in Petri Nets*, pages 1–29. Springer, 1989.

[8] G. Balbo. Introduction to generalized stochastic petri nets. In *Formal Methods for Performance Evaluation*, volume 4486 of *Lecture Notes in Computer Science*, pages 83–131. Springer Berlin Heidelberg, 2007.

[9] J. K. Muppala, G. Ciardo, and K. S. Trivedi. Stochastic Reward Nets for Reliability Prediction. *Communications in Reliability, Maintainability and Serviceability*, 1994.

[10] W. H. Sanders and J. F. Meyer. Stochastic activity networks: formal definitions and concepts. In *Lectures on formal methods and performance analysis*, pages 315–343. Springer-Verlag New York, Inc., 2002.

[11] T. Courtney, S. Gaonkar, K. Keefe, E. W. D. Rozier, and W. H. Sanders. M´obius 2.3: An extensible tool for dependability, security, and performance evaluation of large and complex system models. In *39th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2009)*, pages 353–358, 2009.

[12] G. Ciardo and A. S. Miner. Smart: the stochastic model checking analyzer for reliability and timing. In *Quantitative Evaluation of Systems, 2004. QEST 2004. Proceedings. First International Conference on the*, pages 338–339, 2004.

[13] K.S. Trivedi. Sharpe 2002: Symbolic hierarchical automated reliability and performance evaluator. In *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*, 2002.

[14] J. Lara and H. Vangheluwe. Atom3: A tool for multi-formalism and meta-modelling. In R. D. Kutsche and H. Weber, editors, *Fundamental Approaches to Software Engineering*, volume 2306 of *Lecture Notes in Computer Science*, pages 174–188. Springer Berlin Heidelberg, 2002.

[15] F. Bause, P. Buchholz, and P. Kemper. A toolbox for functional and quantitative analysis of deds. In R. Puigjaner, N. Savino, and B. Serra, editors, *Computer Performance Evaluation*, volume 1469 of *Lecture Notes in Computer Science*, pages 356–359. Springer Berlin Heidelberg, 1998.

[16] V. Vittorini, M. Iacono, N. Mazzocca, and G. Franceschinis. The osmosys approach to multi-formalism modeling of systems. *Software and Systems Modeling*, 3(1):68–81, 2004.

[17] David M. Nicol, William H. Sanders, and Kishor S. Trivedi. Model-based evaluation: From dependability to security. *IEEE Trans. Dependable Secur. Comput.*, 1:48–65, January 2004. ISSN 1545-5971.

[18] Simona Bernardi, José Merseguer, and Dorina C. Petriu. A dependability profile within MARTE. *Software and System Modeling (SoSyM)*, 10(3):313–336, 2011.

[19] A. D'Ambrogio, G. Iazeolla, and R. Mirandola. A method for the prediction of software reliability. In *Proc. of the 6-th IASTED Software Engineering and Applications Conference (SEA2002)*, SEA2002, 2002.

[20] G.J. Pai and J.B. Dugan. Automatic Synthesis of Dynamic Fault Trees from UML System Models. In *Proceedings of the 13th International Symposium on Software Reliability Engineering*, pages 243–254, Washington, DC, USA, 2002. IEEE CS. ISBN 0-8186-1763-3.

[21] Andrea Bondavalli, Mario Dal Cin, Diego Latella, István Majzik, András Pataricza, and Giancarlo Savoia. Dependability analysis in the early phases of uml-based system design. *Comput. Syst. Sci. Eng.*, 16(5):265–275, 2001.

[22] Gábor Huszerl, István Majzik, András Pataricza, Konstantinos Kosmidis, and Mario Dal Cin. Quantitative analysis of uml statechart models of dependable systems. *Comput. J.*, 45(3):260–277, 2002.

[23] K. Tadano, J. Xiang, M. Kawato, and Y. Maeno. Automatic synthesis of SRN models from system operation templates for availability analysis. In *Proceedings of the 30th international conference on Computer safety, reliability, and security*, SAFECOMP'11, pages 296–309, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-24269-4.

[24] S. Bernardi, F. Flammini, S. Marrone, J. Merseguer, C. Papa, and V. Vittorini. Model-driven availability evaluation of railway control systems. volume 6894 of *Lecture Notes in Computer Science*, pages 15–28, 2011.

[25] E.C. Andrade, F. Machida, K. Dong Seong, and K.S. Trivedi. Modeling and analyzing server system with rejuvenation through sysml and stochastic reward nets. In *Availability, Reliability and Security (ARES), 2011 Sixth International Conference on*, pages 161–168, 2011.

[26] 50126: Railway applications - the specification and demonstration of reliability, availability, maintainability and safety (rams).

[27] Afnor association, www.afnor.com.

[28] Leonardo Montecchi, Paolo Lollini, and Andrea Bondavalli. Towards a MDE transformation workflow for dependability analysis. In *ICECCS*, pages 157–166, 2011.

[29] SATURN: SysML bAsed modeling, architecTUre exploRation, simulation and syNthesis for complex embedded systems, 2009. http://www.saturn-fp7.eu/.

[30] CHESS: Composition with guarantees for High-integrity Embedded Software components aSsembly, 2009. http://www.chess-project.org.

[31] C. Haskins, K. Forsberg, and M. Krueger. Systems engineering handbook. *INCOSE Version*, 3, 2006.

[32] Douglas C Schmidt. Model-driven engineering. *COMPUTER-IEEE COMPUTER SOCIETY-*, 39(2):25, 2006.

[33] David Ameller, Xavier Franch, and Jordi Cabot. Dealing with non-functional requirements in model-driven development. In *Requirements Engineering Conference (RE), 2010 18th IEEE International*, pages 189–198. IEEE, 2010.

[34] Model driven architecture, http://www.omg.org/mda/, .

[35] Model driven architecture denition, http://ormsc.omg.org/mda_guide_working_page.htm., .

[36] Stephen J Mellor, Tony Clark, and Takao Futagami. Model-driven development: guest editors' introduction. *IEEE software*, 20(5):14–18, 2003.

[37] Steven Kelly and Juha-Pekka Tolvanen. *Domain-specific modeling: enabling full code generation.* Wiley-IEEE Computer Society Press, 2008.

[38] Antero Taivalsaari. On the notion of inheritance. *ACM Comput. Surv.*, 28:438–479, September 1996. ISSN 0360-0300.

[39] Mikaël Barbero, Frédéric Jouault, Jeff Gray, and Jean Bézivin. A practical approach to model extension. In *Proceedings of the 3rd European conference on Model driven architecture-foundations and applications*, ECMDA-FA'07, pages 32–42, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-72900-6.

[40] A. Ledeczi, G. Nordstrom, G. Karsai, P. Volgyesi, and M. Maroti. On metamodel composition. In *Proceedings of the 2001 IEEE International Conference on Control Applications*, 2001.

[41] Jess Snchez Cuadrado and Jess Garca Molina. A model-based approach to families of embedded domain specific languages. *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, 2010.

[42] OMG-UML2. *Unified Modeling Language: Infrastructure and Superstructure.* OMG, May 2011. Version 2.4, formal/11-08-05.

[43] OMG-OCL2. *Object Constraint Language.* OMG, January 2012. Version 2.3, formal/12-01-01.

[44] GIOVANNI Giachetti, BEATRIZ Marin, and OSCAR Pastor. Integration of domain-specific modeling languages and uml through uml profile extension mechanism. *International Journal of Computer Science and Applications*, 6(5):145–174, 2009.

[45] Bran Selic. A systematic approach to domain-specific language design using uml. In *Object and Component-Oriented Real-Time Distributed Computing, 2007. ISORC'07. 10th IEEE International Symposium on*, pages 2–9. IEEE, 2007.

[46] François Lagarde, Huáscar Espinoza, François Terrier, Charles André, and Sébastien Gérard. Leveraging patterns on domain models to improve uml profile definition. In *Fundamental Approaches to Software Engineering*, pages 116–130. Springer, 2008.

[47] Frédéric Jouault and Ivan Kurtev. Transforming models with atl. In *Satellite Events at the MoDELS 2005 Conference*, pages 128–138. Springer, 2006.

[48] UML-MARTE. UML Profile for MARTE: Modeling and Analysis of Real-time Embedded Systems, June 2011. Version 1.1, formal/11-06-02.

[49] Ebrahim Bagheri and Ali A Ghorbani. Uml-ci: A reference model for profiling critical infrastructure systems. *Information Systems Frontiers*, 12(2):115–139, 2010.

[50] M. S. Lund, B. Solhaug, and K. Stølen. Foundations of security analysis and design vi. chapter Risk analysis of changing and evolving systems using CORAS, pages 231–274. Springer-Verlag, Berlin, Heidelberg, 2011. ISBN 978-3-642-23081-3.

[51] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl E. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Sec. Comput.*, 1(1):11–33, 2004.

[52] C. Papa. *Una metodologia per la modellazione formale di sistemi critici basata su metodi e tecniche di Model Driven Engineering. Doctoral Thesis.* 2011.

[53] Anneke Kleppe. *Software Language Engineering: Creating Domain-Specific Languages Using Metamodels.* Addison-Wesley Professional, 2008.

[54] S. Marrone, C. Papa, and V. Vittorini. Multiformalism and transformation inheritance for dependability analysis of critical systems. In *Proceedings of 8th Integrated formal methods*, IFM'10, pages 215–228, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-16264-9, 978-3-642-16264-0.

[55] Dennis Wagelaar, Ragnhild Van Der Straeten, and Dirk Deridder. Module superimposition: a composition technique for rule-based model transformation languages. *Software and Systems Modeling*, October 2009. ISSN 1619-1366.

[56] D. Codetta Raiteri, M. Iacono, G. Franceschinis, and V. Vittorini. Repairable fault tree for the automatic evaluation of repair policies. In *Proceedings of the 2004 International Conference on Dependable Systems and Networks*, pages 659–668, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2052-9.

[57] Eugene Charniak. Bayesian networks without tears: making bayesian networks more accessible to the probabilistically unsophisticated. *AI Mag.*, 12(4):50–63, November 1991. ISSN 0738-4602.

[58] F. Fleurey, J. Steel, and B. Baudry. Validation in model-driven engineering: testing model transformations. In *First International Workshop on Model, Design and Validation*, 2004.

[59] Márk Asztalos, László Lengyel, and Tihamér Levendovszky. Towards automated, formal verification of model transformations. In *Proceedings of ICST '10*, pages 15–24, Washington, DC, USA, 2010. IEEE Computer Society. ISBN 978-0-7695-3990-4.

[60] A. Bobbio, L. Portinale, M. Minichino, and E. Ciancamerla. Improving the analysis of dependable systems by mapping fault trees into bayesian networks. *Reliability Engineering and System Safety*, 71(3):249–260, 2001.

[61] Soheib Baarir, Marco Beccuti, Davide Cerotti, Massimiliano De Pierro, Susanna Donatelli, and Giuliana Franceschinis. The greatspn tool: recent enhancements. *SIGMETRICS Perform. Eval. Rev.*, 36(4):4–9, 2009. ISSN 0163-5999.

[62] G. Cozman. JavaBayes - User Manual. version 0.346.

[63] E. Bagheri and A. A. Ghorbani. UML-CI: A reference model for profiling critical infrastructure systems. *Information Systems Frontiers*, 12(2):115–139, 2010.

[64] J. Jürjens. *Secure systems development with UML*. Springer, 2005. ISBN 978-3-540-00701-2.

[65] F. Lagarde and et al. Improving UML profile design practices by leveraging conceptual domain models. In *ASE 2007, Atlanta (USA)*, pages 445–448. ACM, November 2007.

[66] G. Franceschinis, M. Gribaudo, M. Iacono, S. Marrone, F. Moscato, and V. Vittorini. Interfaces and binding in component based development of formal models. In *Proceedings of the 4th International ICST Conference on Performance Evaluation Methodologies and Tools*, VALUETOOLS '09, pages 44:1–44:10. ICST, 2009. ISBN 978-963-9799-70-7.

[67] F. Moscato, V. Vittorini, F. Amato, A. Mazzeo, and N. Mazzocca. Solution workflows for model-based analysis of complex systems. *IEEE T. Automation Science and Engineering*, 9(1):83–95, 2012.

[68] W. J. Zhang, Q. Li, Z. M. Bi, and X. F. Zha. A generic Petri net model for flexible manufacturing systems and its use for FMS control software testing. *International Journal of Production Research*, 38(50):1109–1131, 2000.

[69] A. Bondavalli, P. Lollini, and L. Montecchi. Analysis of user perceived QoS in ubiquitous UMTS environments subject to faults. In *Proceedings of the 6th IFIP WG 10.2 international workshop on Software Technologies for Embedded and Ubiquitous Systems*, SEUS '08, pages 186–197, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-87784-4.

[70] K. Tiassou, K. Kanoun, M. Kaâniche, C. Seguin, and C. Papadopoulos. Modeling aircraft operational reliability. In *Proceedings of the 30th international conference on Computer safety, reliability, and security*, SAFECOMP'11, pages 157–170, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-24269-4.

[71] S. Chiaradonna, F. Di Giandomenico, and P. Lollini. Definition, implementation and application of a model-based framework for analyzing interdependencies in electric power systems. *International Journal of Critical Infrastructure Protection*, 4(1):24 – 40, 2011. ISSN 1874-54820.

[72] Ph. Kotler. *Marketing Management, Analysis, Planning, Implementation and Control*. Prentice Hall, 1991.

[73] E. Cascetta. *Transportation System Analyses, models and applications*. Springer, 2009.

[74] I. Chieh Yu P. Enger A. M. Hagalisletto, J. Bjork. Constructing and refining large-scale railway models represented by petri nets. *Trans. On System, Man and Cybernetics-Part C: Applications and Reviews*, 37(4):444 – 460, 2007.

[75] G. Camarillo and M. A. Garcia-Martin. *The 3G IP Multimedia Subsystem (IMS): Merging the Internet and the Cellular Worlds*. III edition, September 2008. ISBN 978-0-470-51662-1.

[76] A. Amirante, T. Castaldi, L. Miniero, and S.P. Romano. Improving the scalability of an IMS-compliant conferencing framework through presence and event notification. In *Proceedings of the 1st International Conference on Principles, Systems and Applications of IP Telecommunications (IPTComm)*, July 2007.

[77] A. Amirante, T. Castaldi, L. Miniero, and S.P. Romano. Improving the scalability of an IMS-compliant conferencing framework part ii: Involving mixing and floor control. In *Lecture Notes in Computer Science - IPTComm 2008*, pages 174–195. Springer-Verlag, 2008.

[78] P. Truchly, M. Golha, F. Tomas, G. Radoslav, and M. Legen. Simulation of IMS using current simulators. In *ELMAR, 2008. 50th International Symposium*, volume 2, pages 545 –548, sept. 2008.

[79] 3GPP. Conferencing using the IP multimedia (IM) core network (CN) subsystem; stage 3. Technical report, 3GPP, March 2006.

[80] M. Barnes, C. Boulton, and O. Levin. Rfc 5239 - a framework for centralized conferencing. Request for comments, IETF, June 2008.

[81] A. Buono, S. Loreto, L. Miniero, and S.P. Romano. A distributed IMS enabled conferencing architecture on top of a standard centralized conferencing framework. *IEEE Communications Magazine*, 45(3):152–159, March 2007.

[82] C. Bo, C. Junliang, and D. Min. Petri net based formal analysis for multimedia conferencing services orchestration. *Expert Systems with Applications*, 39:696–705, January 2012.

[83] P. Owezarski and M. Boyer. *Modeling of Multimedia Architectures: The Case of Videoconferencing with Guaranteed Quality of Service*, pages 501–525. ISTE, 2010. ISBN 9780470611647.

[84] telos revB datasheet. *http://moss.csc.ncsu.edu/ mueller/rt/rt11/readings/projects/g4/datasheet.pd*