Journal of Software and Systems Modeling manuscript No.
(will be inserted by the editor)

A Dependability Profile within MARTE

Simona Bernardi[1], **José Merseguer**[2], **Dorina C. Petriu**[3]

[1] Dipartimento di Informatica, Università di Torino, Italy
e-mail: `bernardi@di.unito.it`

[2] Departamento de Informática e Ingeniería de Sistemas,
Universidad de Zaragoza, Spain
e-mail: `jmerse@unizar.es`

[3] Department of Systems and Computer Engineering,
Carleton University, Ottawa, Canada
e-mail: `petriu@sce.carleton.ca`

Received: date / Revised version: date

Abstract The importance of assessing software non-functional properties (NFP) beside the functional ones is well accepted in the software engineering community. In particular, dependability is a NFP that should be assessed early in the software life-cycle by evaluating the system behaviour under different fault assumptions. Dependability-specific modeling and analysis techniques include for example Failure Mode and Effect Analysis for qualitative evaluation, stochastic Petri nets for quantitative evaluation, and fault trees for both forms of evaluation. Unified Modeling Language (UML) may be specialized for different domains by using the profile mechanism. For example, the MARTE profile extends UML with concepts for modeling and quantitative analysis of real-time and embedded systems (more specifically, for schedulability and performance analysis). This paper proposes to add to MARTE a profile for dependability analysis and modeling (DAM). A case study of an intrusion-tolerant message service will offer insight on how the MARTE-DAM profile can be used to derive a stochastic Petri net model for performance and dependability assessment.

1 Introduction

The dependability of a system, as defined in Laprie et al. [7] is the ability to avoid failures that are more frequent and more severe than acceptable. The dependability encompasses a set of non-functional properties, called attributes of dependability, such as: a) availability, the readiness for correct service; b) reliability, the continuity of correct service; c) safety, the absence of catastrophic consequences on the users and environment; d) maintainability, the ability to undergo modifications and repairs.

The importance of assessing software non-functional properties (such as dependability, performance, security) throughout the software development process is well accepted in the software engineering community. In particular, dependability-specific modeling and analysis techniques include: Failure Mode and Effect Analysis (FMEA) for qualitative evaluation, stochastic Petri nets for quantitative evaluation, and fault trees for both. A general approach for analyzing a certain NFP of a software model expressed in UML is to: a) annotate the UML specification with information needed for that particular analysis, b) extract an analysis-domain model from the annotated UML model, and c) determine the properties by using tools from that domain.

Although there are several proposals in literature for extending UML models with dependability annotations, as reviewed in Section 3 of this paper, each covers only a subset of dependability aspects. The terminology and concepts used in different publications are sometimes inconsistent with each other, as there was no attempt to define a unified domain model for dependability analysis. Compared with the performance and schedulability analysis domains, which are supported by standard UML profiles such as "Schedulability, Performance and Time Specification" (SPT) [37] and "Modeling and Analysis of Real-Time Embedded Systems" (MARTE) [38], there is no similar standard profile for the dependability analysis of UML-based models yet. There is another OMG standard specifying UML extensions for a variety of non-functional properties, named the "Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms" (QoS&FT) [39]. QoS&FT provides a flexible but heavy-weight mechanism to define properties such as performance, security or reliability by means of specific QoS catalogs. The annotation mechanism is supported by a two-step process, which implies catalog binding and either the creation of extra objects just for annotation purposes, or the specification of long OCL expressions.

The main objective of this paper is to propose a UML profile for quantitative dependability analysis of software systems modeled with UML, with particular focus on the following facets of dependability: reliability, availability, maintainability and safety. One of our own requirements for defining such a profile was to reuse the best practices reported in literature, and to unify the terminology and concepts for different dependability aspects under a common dependability domain model (which will be presented in Section 2). According to [44], the UML profile mechanism can be used to define expressive domain-specific modeling languages (DSMLs). This means that UML extended with the dependability profile proposed in this paper becomes, in fact, a domain specific modeling language for the dependability analysis domain.

We realized that the recently adopted MARTE profile [38] contains many features that would be also useful for the dependability profile, so we decided to make the latter compliant with the former. More specifically, we used MARTE's Non-Functional Properties (NFP) framework and its corresponding Value Specification Language (VSL) for defining dependability-specific data types necessary for the profile definition, and specialized general concepts from MARTE's generic quantitative analysis model (GQAM) for the dependability analysis domain. We adopted a systematic approach for the definition of the dependability profile according to the recommendations from Selic [44] (see also Lagarde et al. [29]). First we defined the domain model for dependability analysis and modeling, which covers different dependability aspects, and reuses and unifies the concepts from previous work. Secondly we mapped the domain concepts to elements of a UML profile. The new proposed stereotypes either extend UML meta-classes or specialize MARTE stereotypes, while the stereotype attribute definitions use dependability-specific types defined in the DAM library. The paper applies the proposed dependability profile to a Message Redundancy Service (MRS) case study and gives insight on how to derive from the annotated UML model a Deterministic and Stochastic Petri Net (DSPN) [4] model, which can be used for performance and dependability analysis and assessment.

This paper is an extended version of [12], offering an in-depth description of the dependability domain model and of the DAM profile, as well as the derivation of the latter from the former. Moreover, this paper introduces a new case study, MRS, derives a dependability model from its DAM-annotated design, then performs the quantitative dependability analysis and assessment of the MRS based on the derived model.

The paper is organized as follows: the next subsection presents briefly the background of the paper, MARTE and stochastic Petri nets. Section 2 introduces the approach we followed for the dependability profile definition and Section 3 gives a survey of related work from literature. Section 4 presents the dependability domain model and Section 5 describes the proposed DAM profile and library. Section 6 gives an example of profile application to a Message Redundancy Service (MRS); Section 7 describes the quantitative analysis and assessment of the MRS case study with the help of a DSPN model, and Section 8 presents the conclusions.

1.1 Background

We briefly introduce the background for this paper: the MARTE profile, which we propose to extend for dependability analysis, and the stochastic Petri nets formalism, which is used in Section 7 as the dependability-specific technique for the analysis and assessment of the MRS case study.

MARTE profile. The "UML Profile for Modeling and Analysis of Real-Time and Embedded systems" [38] is a recently adopted standard profile that specializes UML by adding domain-specific concepts for modeling and analysis of real-time and embedded systems. MARTE extends UML in a lightweight fashion, i.e. through the use of stereotypes, tagged-values and constraints. Two MARTE features are important for this paper. The first is that MARTE enables the specification of quantitative and qualitative nonfunctional properties (NFP) in UML models through its Value Specification Language (VSL). VSL allows developers to attach annotations to UML model elements, by providing the ability to express basic types, data types, values (such as time and composite values), as well as variables, constants and expressions. The second important feature is that MARTE provides a general analysis framework called the General Quantitative Analysis Model (GQAM) sub-profile. Although analysis domains have different terminology, concepts, and semantics, they also share some foundation concepts which are expressed in GQAM. The intent is to specialize GQAM for different kind of quantitative analyses. However, MARTE addresses concretely only the schedulability and performance analysis through two sub-profiles, SAM and PAM, that are specializing GQAM. In this work we propose the sub-profile DAM that specializes GQAM for dependability analysis.

Stochastic Petri Nets. Deterministic and Stochastic Petri Nets (DSPN) which are used in this paper as a formal dependability model are an extended version of a well-known class of stochastic Petri nets named Generalized Stochastic Petri Nets (GSPN) [3]. GSPNs are Petri nets with priority in which two types of transitions are distinguished: immediate (that fire in zero time) and timed (that fire with a delay that is an exponentially distributed random variable). A GSPN system is a 8-tuple $S = (P, T, \Pi, I, O, H, W, M_0)$, where P is the set of places, T is the set of immediate and exponential transitions, $P \cap T = \emptyset$; $\Pi : T \rightarrow \mathbb{N}$ is the priority function that maps transitions onto natural numbers representing their priority level (by default timed transitions have priority equal to zero); $I, O, H : P \times T \rightarrow \mathbb{N}$ are the input, output, inhibition functions, respectively, that assign to each arc a multiplicity; $W : T \rightarrow \mathbb{R}^+$ is a function that defines the rate of the exponential distribution for timed transitions and the weight for immediate transitions. Transition weights are used to solve probabilistically the conflicts among immediate transitions with the same priority. Finally, $M_0 : P \rightarrow \mathbb{N}$ is the initial marking function. DSPN is an extended version of GSPN, where timed transitions can be either deterministic (i.e., characterized by a constant firing delay), or exponential. DSPN is a well-suited formalism for the analysis of systems in which events occur either after a constant duration (like time-outs) or random durations, like in the MRS case study.

2 Approach Overview

In this section, we introduce the process followed to define the proposed profile. The goal of the process, shown in Figure 1, was to produce a technically

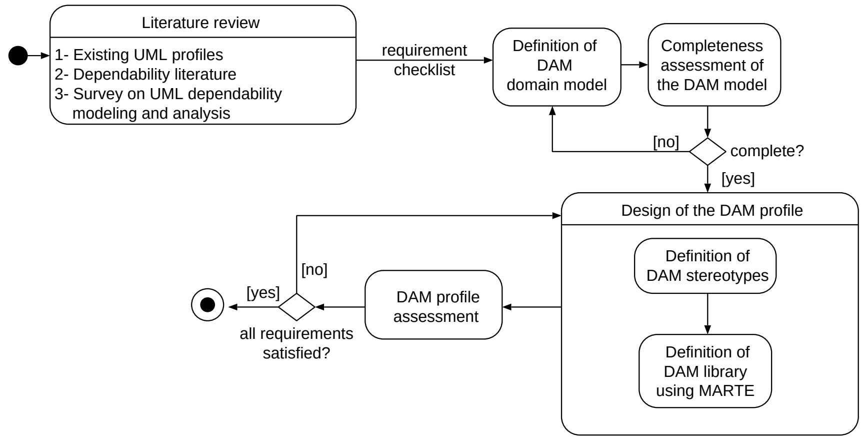


Fig. 1 Approach for Profile definition

correct high-quality UML profile that covers the necessary concepts according to the best practices reported in literature. The steps of the process are described below.

Literature Review The existing standard UML profiles for the analysis of non-functional properties of software systems have been analyzed, in particular SPT [37], QoS& FT [39] and MARTE [38]. None of them provides a comprehensive support for dependability analysis, especially from a quantitative point of view. We investigated the literature on dependability main concepts and taxonomy (e.g., Avizienis et al. [7], Leveson [30], Lyu [31, 32]) as well as on standard methods used for the quantitative assessment of dependability (e.g., [15,16]). We also surveyed the works from the literature proposing dependability modeling and analysis of UML system specifications (about twenty papers). The output of this preliminary step is a checklist of information requirements that a UML profile for dependability analysis and modeling should satisfy, reported in detail in [11].

Definition of DAM Domain Model. We defined a DAM domain model to represent the main dependability concepts from the literature. Its construction required several refinement steps to consider all the surveyed works. The final domain model is described in Section 4.

Completeness Assessment of the DAM Domain Model. We verified whether each requirement of the checklist has been included in the DAM domain model. The verification task has been a manual process. If we found that a requirement was not included, we repeated the refinement step.

Design of the DAM Profile. The DAM profile was defined by mapping the concepts from the DAM domain model to UML and MARTE. Using the DAM domain model we designed: a) the DAM extensions (stereotypes and tags), and b) the DAM library containing dependability specific types. The DAM library has been defined by importing the MARTE library and consists of basic and complex dependability types. We followed an iterative process for the profile definition, in which each domain class was examined, together with its attributes, associations and constraints, to identify the most suitable UML base concepts for it, as suggested in [44]. The objective was to introduce a set of stereotypes small yet expressive, that can be easily used by the software analyst. We then used guidelines from [29] to select the subset of the domain classes that eventually were mapped to stereotypes. Moreover, several patterns proposed in [29] were applied (e.g., the *reference association* pattern), that enable the creation of a profile from the domain model while keeping it consistent with the UML meta-model. Finally, to keep track of the mapping between the domain model and the DAM profile, we adopted the best practice from MARTE to name each introduced extension with the name of the mapped domain class prefixed by *Da*, namely Dependability Analysis.

DAM Profile Assessment. We verified manually whether 1) the requirements from the checklist are satisfied, and 2) the extensions proposed by each surveyed work on dependability analysis and modeling of UML-based systems have been mapped to DAM extensions. If either a requirement was not met or an extension was not mapped to a DAM extension, we went back to the previous step in order to refine it. This activity enabled both a completeness and a consistency checking of dependability extensions found in the different surveyed works from literature.

3 Literature Review

We have selected the literature on dependability concepts and taxonomy, focusing on the following facets of dependability: reliability, availability, maintainability and safety. In particular, for the definition of the firsts three concepts, we have used as references the papers [7,31,32], which precisely provides the terminology according to a component-based view of a software system. The Leveson book [30] has been used, instead, for the definition of safety related concepts. The concepts have been collected in a draft of information requirements checklist for the DAM profile (Table 1).

Table 1: Information requirement checklist

ID	Requirement Description
R1	Identification of the dependability analysis and modeling context: reliability, availability, maintainability or safety.
R2	Specification of dependability requirements in terms of upper/lower bounds.
R3	Specification of dependability metrics to be estimated and properties to be verified (to assess **R2**).

Continued on next page

Table 1 – ***continued from previous page***

ID	Requirement Description
R4	Characterization of threats (fault, error, failure, hazard, accident) that may affect both hw/sw resources and their relationships (fault-error-failure chain, error propagation, hazard-accident).
R5	(*For repairable systems*) Characterization of repair/recovery processes that remove basic or derived threats from the system.
R6	Specification of incorrect behaviour of the system affected by threats as well as the recovery actions that restore the system state.
R7	(*For fault tolerant (FT) systems*) Specification of hw/sw redundant structures.

We have surveyed the existing works from literature providing support for dependability analysis of UML-based designs. In particular, we have selected those works that: 1) address one or more of the considered dependability facets, 2) focus on quantitative analysis, through metrics estimation, and, possibly, 3) propose transformation techniques from UML-based design to dependability formal models. The survey from this section focuses on the works selected according to the above criteria. (A more detailed survey can be found in [11]). The considered works have provided further inputs for the refinement of the checklist in Table 1. In the following, we pinpoint the requirements from the checklist addressed by each work.

Pataricza [41] extends the General Resource Modeling package of the SPT profile with the notion of faults and errors to support the analysis of the effect of local faults to the system dependability. The work includes permanent and transient faults in the resources and uses error propagation to estimate which fault may lead to a failure (**R4**). Fault injection behavioural models are also proposed to represent faults as special virtual clients; the effect of their request causes a change of state in the system (**R6**).

Addouche et al. [1,2] define a profile for dependability analysis of real-time systems compliant with the SPT resource modeling. The input parameters of system resources, i.e., reliability and maintainability, are specified as tags (**R1**). The UML extensions are used to derive probabilistic timed automata for the verification of dependability properties via temporal logic formulas (**R3**). The static model of the system is enriched with new stereotyped classes associated with resources to specify state-based conditional failures (**R4**) (but has the disadvantage that new classes are introduced in the system model for dependability analysis purposes).

Mustafiz et al. [36] extend UML use cases to model all possible exceptional situations that can interrupt the system normal behaviour (**R4**). Use cases are mapped into a kind of probabilistic state-charts to derive a Markov chain for the quantitative assessment of safety and reliability (**R1**), i.e., probability of system success. Their state-charts introduce a probability annotation in the transitions that may lead the system to *success* or *failure* states (**R6**).

Bernardi et al. [8,9] propose a set of UML Class Diagrams for collecting dependability and real-time requirements and properties of automated embedded systems with the use of COTS fault-tolerance mechanisms (**R2-4**). The approach was conceived within the TIRAN and DepAuDE projects[1], and provides support for a semi-automatic derivation of dependability analysis models, such as stochastic Petri nets and temporal logic.

In [10] a method is proposed to assess the quality of service of fault tolerant (FT) distributed systems by deriving performability models from UML+SPT models. State-machines are used to model faults behaviour (**R4**).

The most comprehensive approach so far for reliability and availability analysis (**R1**) of UML specifications has been proposed in [14,34]. A profile for annotating software dependability properties compliant with the taxonomy and basic concepts from [7] is proposed (**R3**). A model transformation process derives timed Petri net models via an intermediate model from the annotated UML models. The approach supports the specification of error propagation between components, as well as independent and dependent failures (**R4**). In particular, it is possible to discriminate between normal and failure states and events (**R6**), and to assign common failure mode occurrence tags to redundant structures (**R7**). The main drawback of this work is the introduction of unnecessary redundant information in the UML model, as sometimes the joint use of more than one stereotype is needed.

Dal Cin [18] proposes a UML profile for specifying dependability mechanisms, aimed at supporting the quantitative evaluation of the effectiveness of a fault tolerance strategy. The approach provides support for capturing reliability and availability requirements of such mechanisms (**R1-2**). However, the profile lacks support for modeling the interactions between dependability mechanisms and system components.

Pai and Dugan [40] present a method to derive dynamic fault trees from UML system models. The method supports the modeling and analysis of sequence error propagations that lead to dependent failures, reconfiguration activities and redundancies (**R4-5,R7**).

The papers [19,17,23,24] address specifically the reliability analysis of UML-based design (**R1**). D'Ambrogio et al. [19] define a transformation of UML models into fault tree models to predict the reliability of component-based software. Cortellessa and Pompei [17] propose a UML annotation for the reliability analysis of component-based systems (**R4**), within the frameworks of the SPT and QoS&FT profiles. The annotations defined in [17] are used by Grassi et al. [23,24] where a model-driven transformation framework for the performance and reliability analysis of component-based systems is proposed. The method uses an intermediate model that acts as bridge between the annotated UML models and the analysis-oriented models. In particular, discrete time Markov process models can be derived for the computation of the service reliability (**R3**).

[1] TIRAN (Tailorable fault tolerance framework for embedded applications), DepAuDE (Dependability for embedded Automation systems in Dynamic Environment with intra-site and inter-site distribution aspects).

Jürjens et al. define a safety [28] and reliability [27] check list, based on UML extension standard mechanisms, to support the specification of dependability requirements and the identification of failure-prone components in the software design (**R1-2,R4**).

The approaches [42,22,26] support the safety analysis of UML-based system models (**R1**). Pataricza et al. [42] use UML stereotypes to identify erroneous states and error correcting transitions in state machine diagrams, integrating the normal and the faulty behaviour of a system component in a single state machine (**R6**). Goseva et al. [22] devise a methodology for the risk assessment of UML models at architectural level; a Markovian model is constructed to estimate the scenario risk factor from risk factors associated to software components and connectors (**R3-4**). Hassan et al. [26] introduce a methodology for the severity analysis of software systems modeled with UML, which integrates different hazard analysis techniques (Functional Failure Analysis, Failure Modes and Effects Analysis and Fault Tree Analysis) to identify system level and component/connector level hazards (**R4**) and to evaluate the cost of failure of system execution scenarios, software components and connectors (**R3**).

Observe that each surveyed work provides some contributions to a subset of requirements from the checklist given in Table 1, and the whole literature survey covers all the information requirements. The checklist was used in our approach to test for the completeness of both the DAM domain model and the DAM profile. As a final result, the proposed DAM profile addresses all the information requirements from the checklist. Hence, it becomes possible to conceptually reuse the transformation approaches for generating dependability analysis models proposed in the surveyed works, under the common UML extension framework proposed in the paper.

4 DAM Domain Model

The DAM domain model has been constructed considering the main dependability concepts from the literature as well as standard methods used for dependability assessment. It is organized into a set of packages, as shown in Figure 2, where the top-level package includes:

- The *System* model provides a description of the system to be analyzed, according to a component-based view [7,31]. The model includes also additional concepts to describe redundancy structures that may characterize a fault tolerant (FT) system [32].
- The *Threats* model introduces the threats [7,31,30] that may affect the system at different levels as well as the relationships between threats.
- The *Maintenance* model introduces repair/recovery actions undertaken in case of repairable systems [7,32].

Figures 3, 4, 5, and 6 show the DAM domain model; the different gray scale used for the classes will be explained in Section 5. The *Core* model

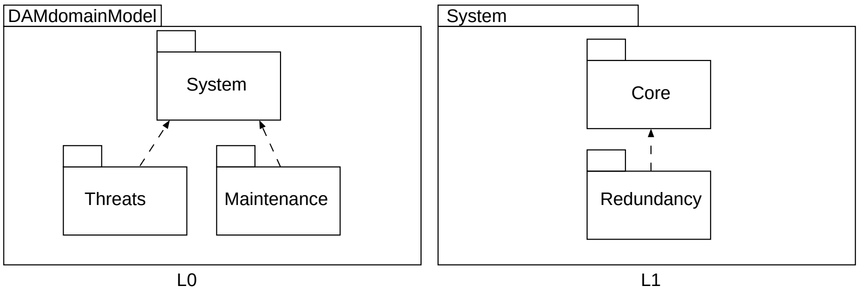


Fig. 2 DAM domain model: Top-level package (L0), System Core package (L1)

(Figure 3) represents the *context* for carrying out dependability analysis. Actually, it is a component-based description of the system to be analyzed that includes both structural and behavioural views. From the structural point of view, the system consists of a set of hardware and software *components* that are bound together through *connectors*, i.e., logical/physical links, in order to interact. A component can be a sub-system consisting of sub-components. The structure of the system is what enables it to generate the behaviour. The system delivers a set of high-level *services*, in response to user *service requests*. Each high-level service is a part of the system behaviour as perceived by its users and it is carried out by the interaction of system components, which provide and request basic services to each other. A component must either provide or request at least one basic service. A service is implemented by a sequence of *steps* that may represent component states and actions.

The *Core* model acts as a bridge between the DAM concepts and the concepts introduced in MARTE for general quantitative analysis and modeling (GQAM). Indeed, several classes of the *Core* model will be mapped to stereotypes that specialize the ones from the GQAM profile.

Observe that some classes of the DAM domain model have attributes that represent requirements, metrics, properties or input parameters. A detailed description of the attributes is given in [11]. Table 2 shows an excerpt of the definition of such attributes, along with the references to the surveyed works from literature and to the standard dependability analysis methods which use them.

A system may be characterized by redundancy structures. Software and hardware redundancy are typical means used to increase the fault tolerance (FT) of software systems, e.g., by eliminating single points of failure. The *Redundancy* model (Figure 4) represents *FT components* [32], which can play different roles in a *redundant structure* [34]. In particular, a redundant structure may consist of several *variants*, i.e, modules with different design that provide the same services, allocated over different *spares*, a *controller*, that is responsible for the co-ordination of the variants, an *adjudicator*, that either looks for a consensus of two or more outputs among the variants ("N-version software" scheme) or applies an acceptance test to the variant

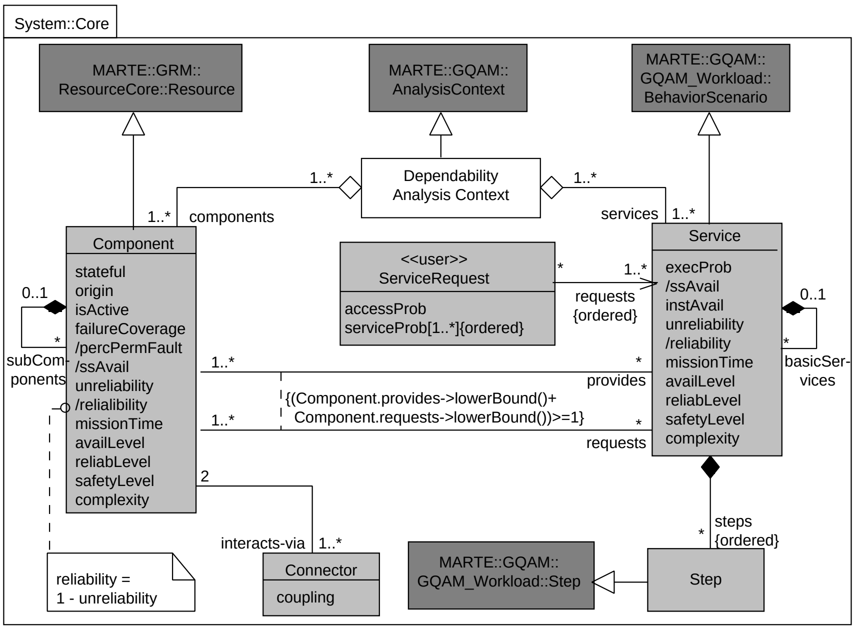


Fig. 3 DAM Domain Model: Core package

outputs ("recovery block" scheme). A spare may substitute for one or more components and, depending on the adopted FT strategy, can be unpowered until needed (*cold*), continually powered (*hot*) or partially powered in the standby mode until it is needed (*warm*). The type of spare can be specified by its *dormancyFactor*, which is the ratio between the spare failure rates in standby mode and in operational mode, respectively. (For instance, a cold spare has a dormancy factor equal to zero, since it cannot fail in standby mode).

Note that it is out of scope in this work to provide full support for the modeling of FT architectures. Rather, the introduction of the *System Redundancy* model is motivated by the objective of providing specific support for the quantitative dependability analysis of the FT systems characterized by redundant structures.

The *Threats* model (Figure 5) includes the threats that may affect the system, namely the *faults, errors, failures* [7,31] and *hazards* [30]. We have introduced an abstract concept of *impairment*, that can be specialized depending of the type of analysis domain, i.e., *failure* for reliability/availability analysis and *hazard* for safety. The model represents also the cause-effect relationships between the threats and the relationships between the threats and the system core concepts. Then, a fault is the original cause of errors and impairments, and affects system components. A *fault generator* concept is added to represent a mechanism, used for example in [10,41], to inject faults in the system and to specify the max number of concurrent faults.

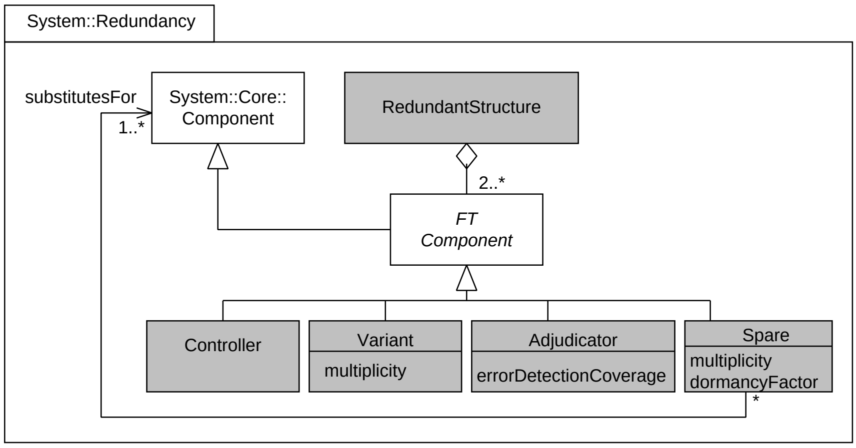


Fig. 4 DAM Domain Model: System Redundancy package

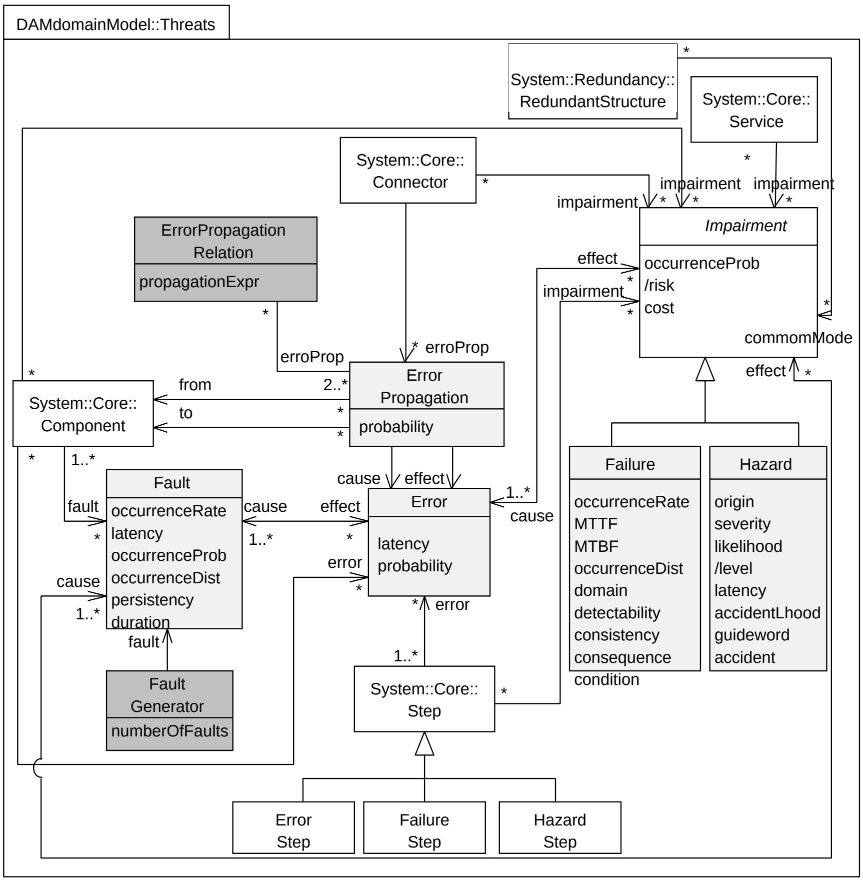


Fig. 5 DAM Domain Model: Threats package

Errors are related to steps (i.e., states or actions) of the basic services provided by the faulty components. When an error affects an external state of a faulty component, that is the service interface of that component, then *error propagations* may occur from the faulty component to other components it interacts with, via the corresponding connectors. A series of error propagations may be inter-related, for example they may occur in a given order [40]; ordered sequences of error propagations as well as non trivial *error propagation relations* can be specified as logical expressions (attribute *propagationExpr*). Errors may cause impairments at different system level: 1) at service step level, leading to *failure/hazard steps*, when the service provided by the component becomes incorrect; 2) at component level, when the component is unable to provide any basic service; 3) at system level, when the impairment is perceived by the system users. Finally, multiple dependent impairments can affect a redundant structure, such as when several redundant components fail in the same mode (i.e., *common mode failures* [32]).

Table 2: DAM domain model: Example of attribute description.

Component	DAMdomainModel::System::Core
stateful	(true) Faulty components can be characterized by an error latency, so they can be restored before failure. (false) Faulty components are considered as failed [14,34].
origin	Hardware/software component [14,34].
isActive	(true) The component can perform its behaviour autonomously and trigger behaviour of other components [17].
failureCoverage	Percentage of failure coverage [40].
/percPermFault	Percentage of permanent faults [14,34]. Derived from *fault* association-end and *persistency* attribute of Threats::Fault class.
/ssAvail	Steady state availability (percentage) [8,9,43]. Derived from *MTTF* (Threats::Failure) and *MTTR* (Maintenance::Repair) attributes.
unreliability	Probability that the time to failure random variable is less or equal than time t (time dependent) [43].
/reliability	Survival function [21]: probability that the component is functioning correctly during the time interval $(0, t]$ [43] (time dependent).
missionTime	Time interval in which the component unreliability is lower than a preassigned threshold.
availLevel	Availability level associated to the nines of availability. E.g., very high corresponds to 99,9% of ssAvail, etc. (application specific).
reliabLevel	Reliability level (application specific).
safetyLevel	Safety level (application specific).
complexity	Complexity metric [8,9,28,27,22] quantifies the component failure proneness.
Failure	DAMdomainModel::Threats
	Continued on next page

Table 2 – continued from previous page

occurrenceRate	Number of failures per unit time [19,40,23,24,8,9].
MTTF	Mean Time To Failure [8,9,34,14].
MTBF	Mean Time Between Failures [8,9].
occurrenceDist	Failure occurrence distribution (time dependent) [34,14,19,40,23,24].
domain	Content, early timing, late timing, halt or erratic [7]. Used in [41,8–10,27,28].
detectability	Signaled or unsignaled [7].
consistency	Consistent or inconsistent [7].
consequence	Minor, marginal, major, or catastrophic [7]. Used in [8,9,22,26].
condition	Logical condition that leads to the failure. Used to express relationships among component failure states [1,2,13].
Fault	DAMdomainModel::Threats
occurrenceRate	Number of faults per unit time [34,14,10].
latency	Time elapsing between a fault occurrence and the instant in which it is perceived by the component(s) [10].
occurrenceProb	Fault occurrence probability (time independent) [17].
occurrenceDist	Fault occurrence distribution (time dependent).
persistency	Transient or permanent [7]. Used in [41,10].
duration	Fault duration from its occurrence. It can be used to discriminate the fault persistency [8,9].

The *Maintenance* model (Figure 6) concerns repairable systems and includes concepts that are necessary to support the evaluation of system availability, that is the *maintenance actions* undertaken to restore the system affected by threats. Indeed, during the execution of maintenance actions, the services provided by the system are either partially or entirely not delivered to the user, so the time to perform maintenance has an impact on the system availability. The execution *rate* and the execution time *distribution* characterize, from a quantitative point of view, the maintenance actions. According to [7,32], we distinguish *repairs* of system components, that involve the participation of external agents (e.g., repairman, test equipment, etc) and *recovery* strategies, usually implemented in FT systems, that aim at transforming the system anomalous states into correct states. In particular, the reconfiguration steps imply the use of spare components [40]. The model represents *replacement* and *reallocation steps.* The former consist in actions in which a set of faulty components are replaced with a set of spares. In *reallocation steps*, a set of software components is reallocated onto a collection of spares. To represent the new system reconfiguration, after the system recovery, the set of replaced/reallocated components and the spares must be ordered and have the same size.

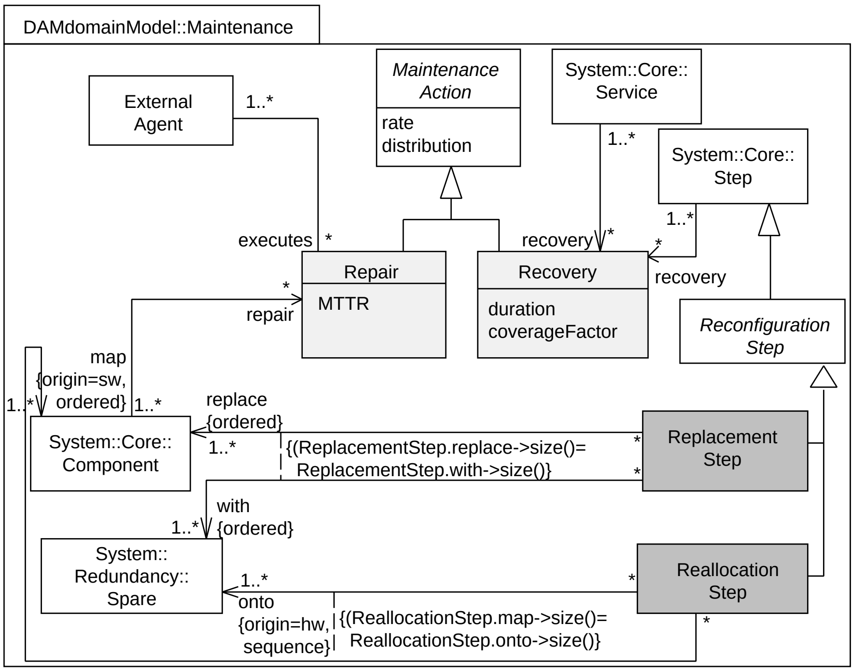


Fig. 6 Maintenance model

5 Design of the DAM Profile

Once the DAM domain model has been defined, the process of mapping it to a concrete profile could be addressed. We have followed an iterative process, in which each class has been examined, together with its attributes, associations and constraints, to identify the most suitable UML base concepts for it, as suggested in [44]. Moreover, following the general guidelines in [44], we aim at designing a technically correct and effective profile which takes advantage of the new UML2 profile mechanisms.

Figure 7(a) gives the big picture of the DAM profile proposal. It consists of a UML package stereotyped as `<<profile>>`, which includes the set of UML extensions and a model library. The first is a package containing a set of dependability stereotypes, their attributes (also called tags) and constraints. Most of the DAM stereotypes specialize the ones of MARTE [38]. The library, detailed in Figure 7(b), is made of basic and complex dependability types used to define the stereotype attributes. In the design of the profile, we have also applied several suggestions and patterns proposed in [29] that enable the creation of a profile from the domain model, while keeping it consistent with the UML meta-model. Moreover, we adopted the best practice from MARTE to keep track of the mapping between the domain model and the DAM profile.

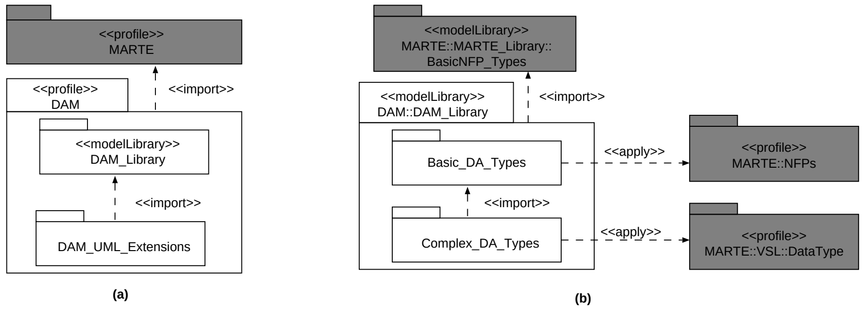


Fig. 7 (a) DAM profile and (b) DAM library

5.1 DAM UML extensions

The DAM extensions package provides the domain expert with a set of stereotypes to be applied at model specification level, i.e., the stereotypes necessary to represent the dependability system view in a concrete UML model. Domain classes are natural candidates for becoming stereotypes. However, as in [29], we aim at providing a small yet sufficient set of stereotypes to be actually used in practical modeling situations. Eventually, only a subset of the domain classes have been mapped to stereotypes (i.e., the pale gray classes in Figures 3, 4, 5 and 6). In order to maintain a traceable mapping, the name of each stereotype will be the name of the domain class prefixed by *Da*, namely Dependability Analysis, as shown in Figures 8 and 9. Table 3 gives as an example the description of the *DaComponent* stereotype, while the complete stereotypes list can be found in [11].

Table 3: Example of stereotype description

DaComponent maps the System::Core::Component domain class	
Generalization	MARTE::GRM::Resource
Extension	none
Attributes	
stateful	Boolean[0..1]
origin	Origin[0..1]
isActive	Boolean[0..1] - Inherited from Resource
failureCoverage	NFP_Percentage[*]
percPermFault	NFP_Percentage[*]
ssAvail	NFP_Percentage[*]
unreliability	NFP_CommonType[*]
reliability	NFP_CommonType[*]
missionTime	NFP_CommonType[*]
availLevel	DaLevel[*] - Application specific
reliabLevel	DaLevel[*] - Application specific
safetyLevel	DaLevel[*] - Application specific
complexity	NFP_Real[*]
	Continued on next page

Table 3 – continued from previous page

fault	DaFault[*] - Faults affecting the component
error	DaError[*] - Errors affecting the component
failure	DaFailure[*] - Failures affecting the component
hazard	DaHazard[*] - Hazards affecting the component
repair	DaRepair[*] - Repairs undergone by the component

In the process of selecting a minimum set of stereotypes, we visited each package in the domain model, characterizing their classes as *abstract, firm, uncertain* or *parametric*, according to [29]. This criterion aims to clearly characterize the role of each DSML concept. *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. For example, in the Core model package, we identified the classes *Component, Connector, Service, ServiceRequest* and *Step* as *firm*, and mapped them to stereotypes (Figure 8, pale grey), while the class *DepAnalysisContext* was regarded as *uncertain.* As for the classes regarded as *abstract*, e.g., *FT Component* (Figure 4), they don't have to be mapped to stereotypes, as indicated in [29].

In a second stage, for each stereotype obtained from a *firm* class, we searched for suitable *extensions*, i.e., the actual UML meta-classes to be extended by the stereotype. To facilitate the extension process, we consulted the proposals in the surveyed literature, identifying the UML model elements annotated with the same dependability properties as the ones characterizing the stereotype. In other words, we applied the general guidelines from [44], based on the semantic similarity between the UML meta-classes and stereotypes. Finally, if a semantically equivalent stereotype did exist in MARTE, then we defined the DAM stereotype as a sub-stereotype of the MARTE one (depicted in dark grey in Figures 8 and 9). For the final result of this stage, see Table 3 showing as an example the definition of the stereotype *DaComponent.* It specializes the MARTE stereotype shown in the "Generalization" row, does not extend directly any UML meta-class (the "Extensions" row is empty) and extends indirectly the same UML meta-classes as the stereotype from which it inherits.

The stereotypes attributes, which characterize their properties, are obtained from the original class in the domain model, either from its attributes or from navigable association ends. The type of the first kind of attributes is a basic UML type, a MARTE NFP type or a basic dependability type (described in Section 5.2.2). To define the type of the attributes obtained from association ends, we introduced the complex dependability types (described in Section 5.2.1). Since the mapping of association ends is less trivial than that of attributes, we often applied the *reference association* pattern in [29]. An example of such pattern is given in Figure 5, where the *Component* class is associated with the *Fault* class through the association-end *fault.* The latter is used to define the attribute *fault* of *DaComponent* (with complex type

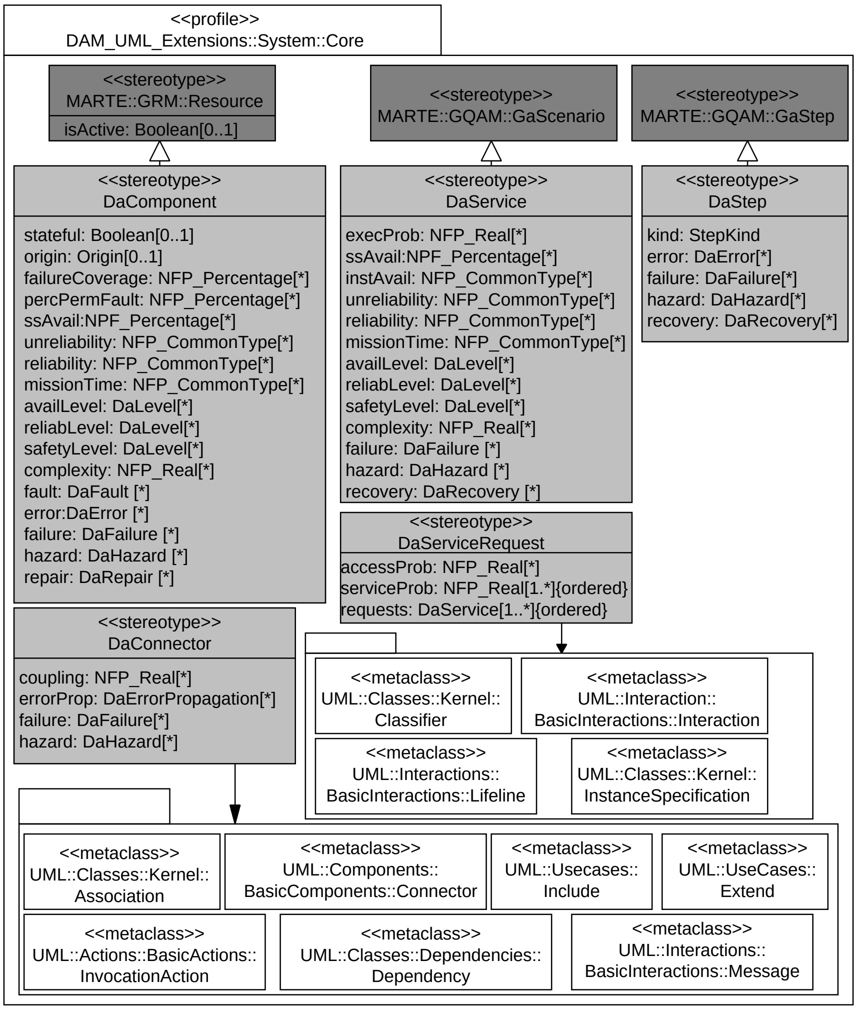


Fig. 8 DAM stereotypes from Core

DaFault, see Table 3). On the other hand, when an abstract class, e.g., *Impairment* in Figure 5, is the target of associations ends, e.g., *impairment* from *Component*, then all the specialized classes (*Failure* and *Hazard*) inherit the associations ends but renamed. Thus, Table 3 shows *failure* and *hazard* as attributes of *DaComponent*. Finally, when defining the attribute multiplicity, we retained for the attributes obtained from association ends their multiplicity values from the domain model.

It is worth noting that the domain model is characterized by several constraints, which have been assigned to the DAM extensions using OCL. They represent constraints for the use of the profile at model specification level. Some constraints already expressed in OCL in the domain model (Figure 6)

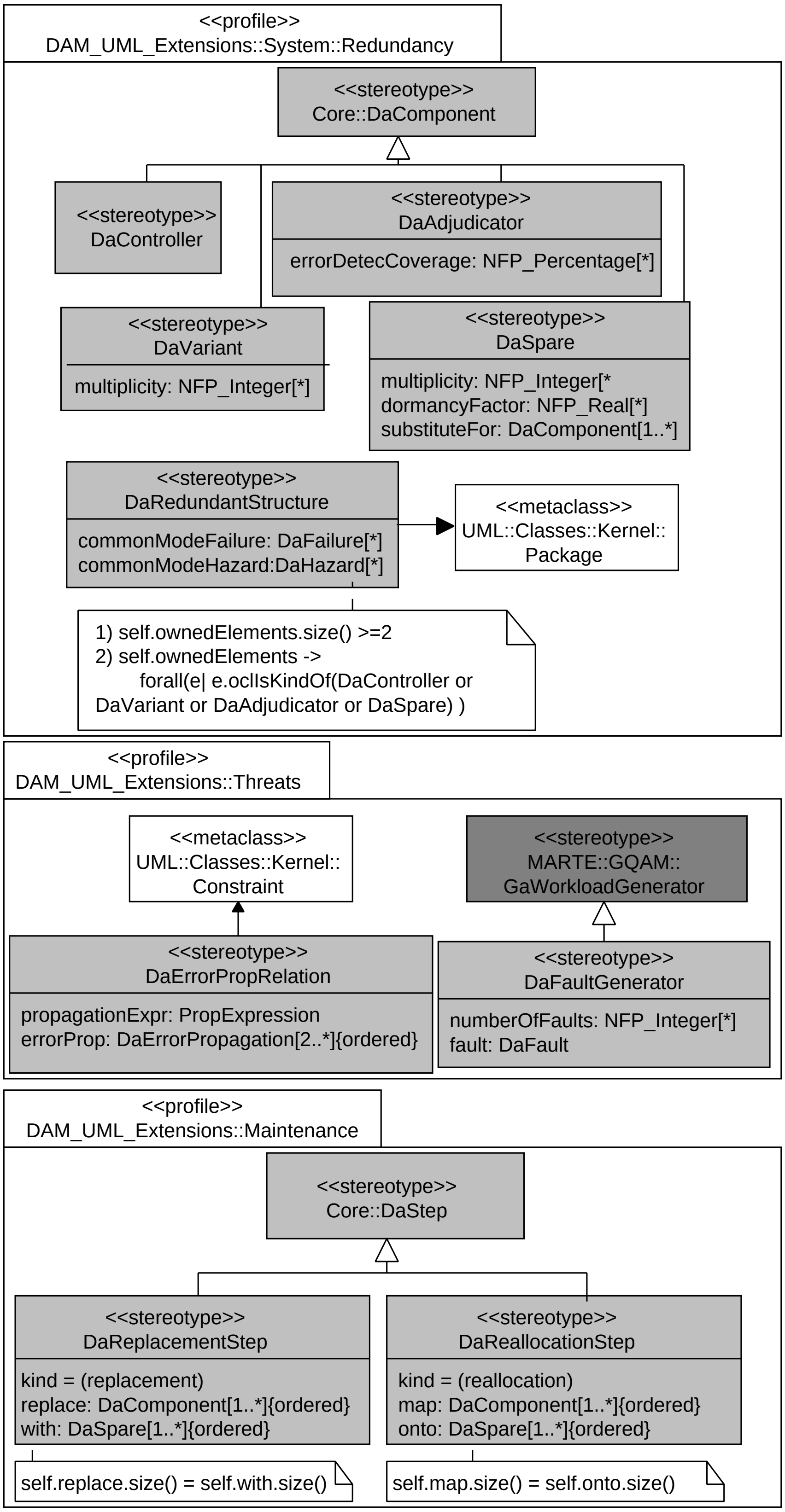


Fig. 9 DAM stereotypes from System Redundancy, Threats and Maintenance

are directly inherited in the profile, while others have been extracted from the domain model. For example, the *DaRedundantStructure* stereotype has two OCL constraints attached (Figure 9), one for mapping (Figure 4) the cardinality of the *RedundantStructure* aggregation, and the other for forcing the types of the elements in the Package.

5.2 DAM library

Figure 7(b) depicts a high-level view of the DAM library, which contains basic and complex dependability types. We have imported the basic NFP types from the MARTE library for the definition of these types. In particular, the MARTE NFPs sub-profile is applied to the definition of new basic dependability types and the VSL sub-profile to the definition of the complex ones.

5.2.1 Complex dependability types The complex dependability types are MARTE data-types used to type DAM extensions. They are characterized by attributes, whose type can be a basic NFP type from the MARTE library, a basic dependability type, or a complex dependability type. The set of complex types has been obtained by mapping classes which model threats or maintenance actions (i.e., the faint grey classes of Figures 5 and 6). A complex dependability type is prefixed by *Da*. Table 4 describes the *DaFailure* complex dependability type, the rest can be found in [11]. As for stereotypes, their attributes can map either an attribute of the (mapped) domain class (e.g., *occurrenceRate* in *DaFailure*), or an attribute inherited from an abstract class (e.g., *occurenceProb*) or an association-end. Also the association-ends from the abstract classes are inherited and renamed, *causeF* in *DaFailure* maps the *cause* association-end between *Impairment* and *Fault* in Figure 5.

Table 4: Complex dependability type description.

DaFailure maps the Threats::Failure domain class	
Attribute	
occurrenceRate	DaFrequency[*]
MTTF	NFP_Duration[*]
MTBF	NFP_Duration[*]
occurrenceProb	NFP_Real[*]
occurrenceDist	NFP_CommonType[*]
domain	Domain[0..1]
detectability	Detectability[0..1]
consistency	Consistency[0..1]
consequence	DaCriticalLevel[*]
risk	NFP_Real[*]
cost	DaCurrency[*]
	Continued on next page

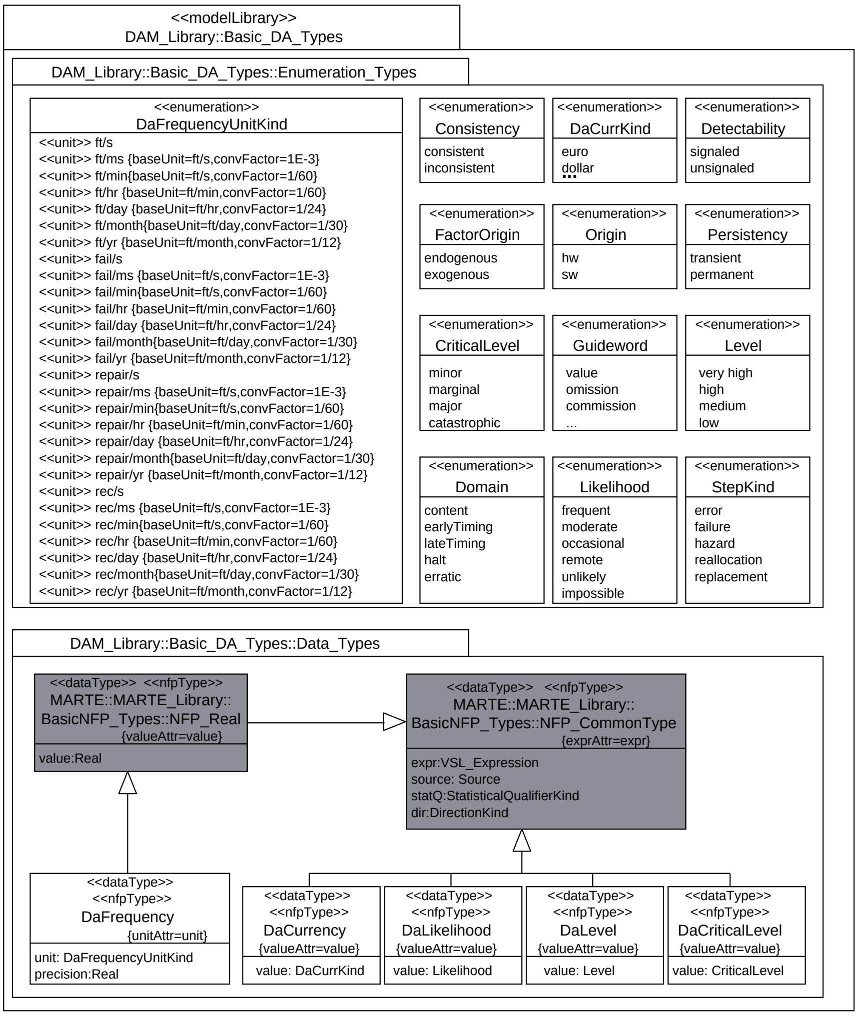


Fig. 10 DAM basic types

Table 4 – continued from previous page

condition	FailureExpression[0..1]
causeF	DaFault[1..*]
causeE	DaError[1..*]

5.2.2 Basic dependability types Basic dependability types, see Figure 10, can be either simple enumeration types or data-types. An example of enumeration type is the *DaFrequencyUnitKind*, which includes a set of frequency units of fault/failure occurrence rates and of repair/recovery rates. An example of data-type is the *DaFrequency* introduced to specify, e.g., a failure occurrence rate as a real value together with a failure frequency unit (e.g., 0.1^{-2} failures per hour).

The data-types specialize the *NFP_CommonType* from MARTE library, which allowed us to reuse several properties of the super-type that enrich the annotation capabilities at system model level. In particular, the *expression* property supports the specification of expressions using the Value Specification Language (VSL). The *source* property can be used to define the origin of the specification (*required*, a requirement to be satisfied; *estimated*, a metric to be computed; or *assumed*, an input parameter). The *statQ* property defines the type of statistical measure (e.g., maximum, minimum, mean). Finally, the *direction*, defines the type of the quality order relation in the allowed value domain of the NFP, for comparative analysis purposes.

5.3 Usage of the DAM profile

At model specification level, a software analyst may apply a DAM stereotype provided that the target model element belongs to a meta-class *extended* by that stereotype. For example, in Figure 12 the *DaService* stereotype is applied to a Use Case model-element. This is possible because *DaService* specializes the MARTE::GQAM::GaScenario stereotype (Figure 8), which in turns specializes the MARTE::TimeModels::TimedProcessing stereotype. Since TimedProcessing *extends* the UML::CommonBehaviours::Behaviour meta-class, then *DaService* can be applied to a wide set of behaviour-related elements such as Use Cases. Although this is the "normal" way of usage, the DAM profile also provides support for the specification of non trivial threat assumptions. In particular, two such examples deserve special attention: the state-based failure conditions and the common mode impairments of a set of redundant components.

State-based failure conditions State-based failure conditions can be specified for either components or services. Note that both classes have an association with the *Impairment* abstract class (Threats model, Figure 5). As shown in Table 3, we have converted association-end *impairment* of component class into two attributes (*failure* and *hazard*) of complex types (*DaFailure* and *DaHazard*). Regarding *DaFailure* type (Table 4), its *condition* attribute (*FailureExpression* type) let us specify a logical expression that accounts for the state-base failure condition. The syntax is given in Table 5. For example, let us assume that the failure of component A depends on the state of component B, in particular when component B is either in state *degraded* or *failed*. Then, we can stereotype both components as *DaComponent* and annotate the failure condition on component A, as shown in Figure 11(a).

Common mode failures/hazards Stereotype *DaRedundantStructure* in Figure 9 is used to characterize the impairments affecting simultaneously the

```
condition-value ::=   '(' failure-body ')'
failure-body ::=      fail-term | 'not' fail-term |
                      'not' '(' failure-body ')' |
                      failure-body logical-op fail-term
logical-op ::=        'and' | 'or' | 'xor' | 'implies'
fail-term ::=         '(' 'component' '=' component,
                       'state' '='state)
component ::=         string
state ::=             string
```

Table 5 BNF syntax for the specification of state-based failure conditions

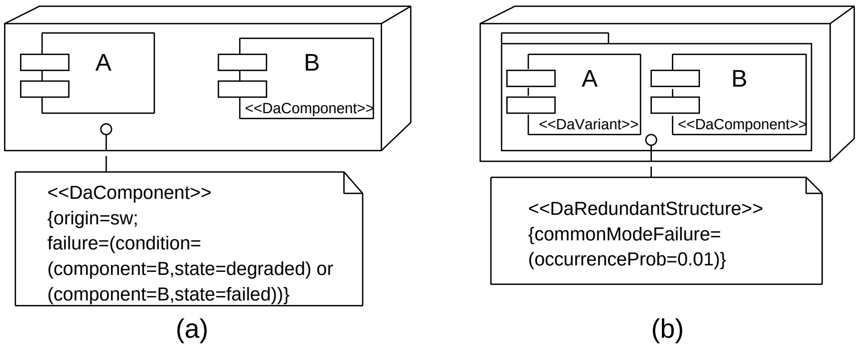


Fig. 11 DAM profile usage

set of FT components belonging to a redundant structure (that is components stereotyped as *DaVariant*, or *DaController*, or *DaAdjudicator* or *DaSpare*). These impairments account for the failures and the hazards. The association-end between the *RedundantStructure* and the *Impairment* classes (Threats model, Figure 5) is mapped onto two attributes (*commonModeFailure* and *commonModeHazard*) belonging to the *DaRedundantStructure* stereotype. They will allow to specify among others the common mode failure/hazard probability. The annotation is carried out by including the set of FT components into a package stereotyped as *DaRedundantStructure* and then specifying the desired attributes. See Figure 11(b) for an example.

6 Modeling with MARTE-DAM

In this section we consider the case study of a Message Redundancy Service (MRS), which aims at improving the dependability of distributed systems that have to provide their services even in the presence of malicious attacks. In particular, the goal of the MRS is to enhance a system with *intrusion-tolerance* capabilities by delivering only uncorrupted messages to the target

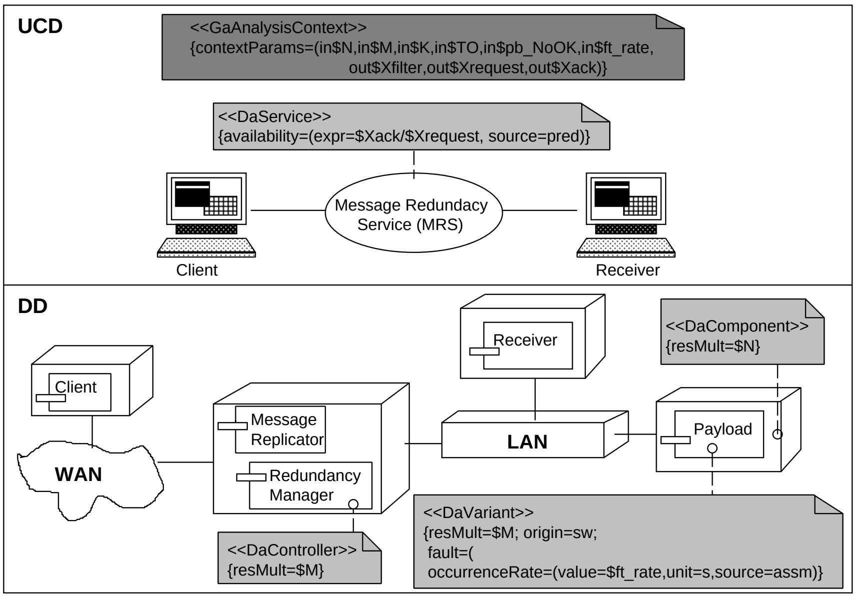


Fig. 12 Message Redundancy Service overview (UCD) and architecture (DD).

destination. For this purpose, as well as to attain a fault-tolerant MRS, some well-known fault-tolerance mechanisms are applied, such as hardware and software redundancy and voting [32]. By using them, MRS should be able to mask software faults that otherwise could lead to service failure. Usually, these mechanisms are implemented along with *recovery* strategies that restore the system services. For the sake of simplicity, we will not consider *recovery* in this example.

The UML specification of MRS is shown in Figures 12, 13 and 14. The Use Case Diagram (UCD) shows the main use case realized by the service scenario given in the Sequence Diagram (SD) in Figure 13. MRS receives messages from Clients, specifying the target receiver and the file to deliver. The Message Replicator (MR) is an interface agent of MRS, which creates for each message another agent, the Redundancy Manager (RM), which is in charge of the actual delivery. RM creates N replicas (*software redundancy*), called Payloads, which scan and decipher the file. Each Payload sends back to RM a result, that can be of approval (if the file was found clean) or of rejection otherwise. RM ends up killing the Payloads and deciding with a majority *voting* algorithm, to deliver or not the message to its final receiver. In any case, MR is informed about the service outcome: the message has been correctly delivered (`value=OK`) or it has been detected as corrupted (`value` with others values), or a time-out exception occurred and no decision was taken by the RM then producing a `noResult` message.

The Deployment Diagram (DD) specifies the system architecture. The Payloads, when created, will be deployed on N different nodes to improve

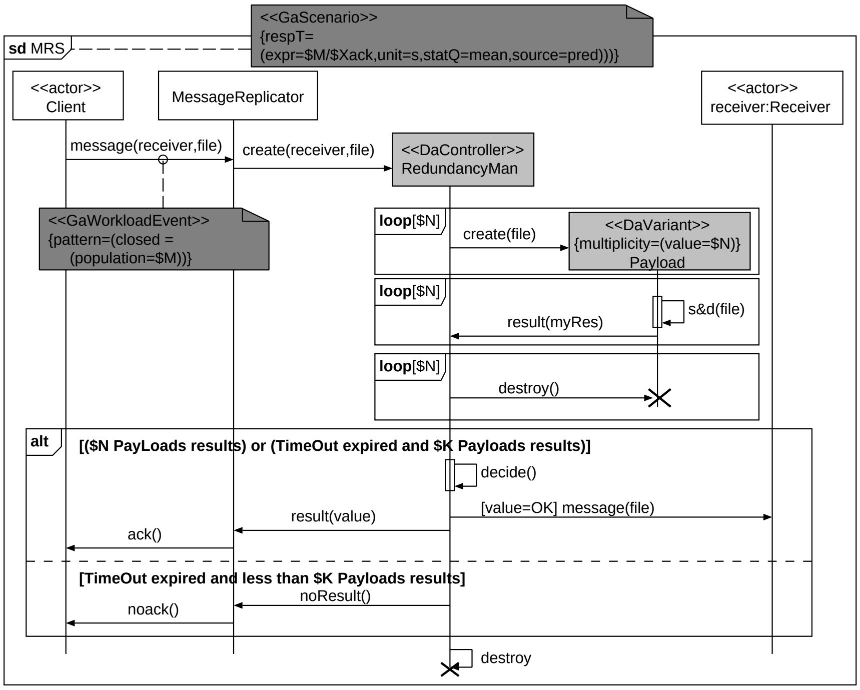


Fig. 13 MRS scenario.

dependability (*hardware redundancy*), while the unique MR and the RMs execute on the same node to avoid transmission delays. The local area network (LAN) is a secured one, so the messages can be trustily delivered.

We can get a better understanding of the MRS behaviour with a closer look at the State Machines (SMs) specification (Figure 14). MR processes the Client requests (`message(receiver,file)` transition) and receives the service outcomes from RM, both when the service success (`result(value)`) and when the service fails due to a time-out exception (`noResult()`). MR ends up acknowledging (`ack()`) or not (`noack()`) the Client about the delivery of the message. Observe that MR sends an `ack` to the Client even if the file had been considered corrupted. After processing the `message`, MR creates an RM that, in turn, creates the N Payloads and sends the file to them for analysis. RM sets then a time-out and, concurrently, waits for the Payloads to reply. Just in case the N Payloads answer before the time-out expires, the RM destroys them, aborts the time-out and starts deciding. On the other hand, if the time-out is raised while RM is still waiting for results from the Payloads, then the latters are destroyed and RM checks if at least K Payloads have sent the results (`[count>=$K]`). In a positive case, RM is allowed to decide about the integrity of the file. Therefore, the MRS can mask at most $N-K$ concurrent software faults, i.e., the service can tolerate up to $N-K$ Payloads down per message. When the time-out expires and the

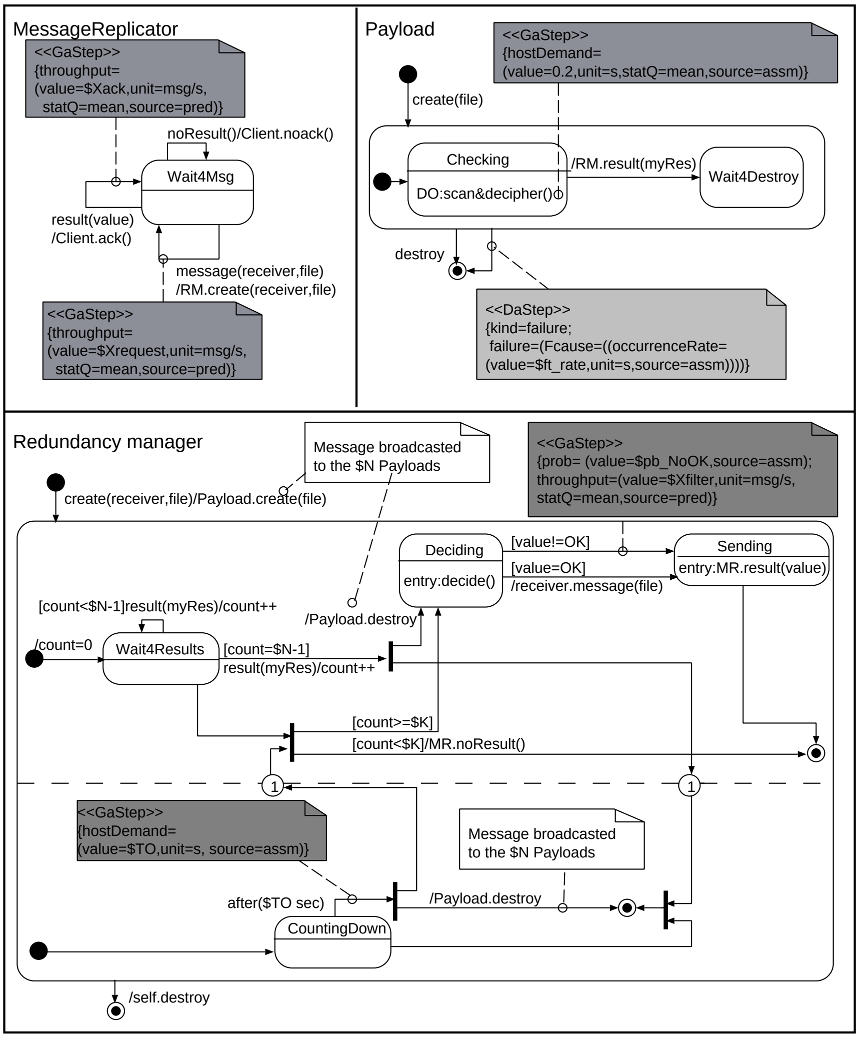


Fig. 14 MRS state machines.

number of results received from the Payloads is less than K (`[count<$K]`), RM does not take any decision and sends a message `noResult()` to MR.

6.1 The annotated model

So far we have modeled the MRS behaviour as a set of UML diagrams, now we can annotate them with the dependability system properties by using the MARTE-DAM profile. The annotated model will allow us to perform both dependability and performance analysis. Indeed, the MARTE-DAM profile enables UML to define both performance metrics, (e.g., *scenario* *response time, activity throughput*), and dependability metrics (e.g., *reliability, availability, safety*). In this example, we are interested in computing the *steady state availability* of the MRS (see the pale grey annotation in Figure 12(UCD)). We interpret the *steady state availability* in the problem domain as the percentage of messages that the system can process (not only those delivered, also those detected as corrupted) out of all messages requested by the clients. So, the *steady state availability* is expressed as a ratio of two SM transition throughputs (`$Xack` and `$Xrequest`). These two parameters, being quantitative, are annotated using the GQAM, concretely the `GaStep` stereotype (see the MR state-machine). Note that these throughputs are output parameters, i.e., metrics to be computed, as indicated by the modifier `source=pred`. The DD defines the type of the resource, variant or controller, and its multiplicity. Note that some multiplicities are equal to the number of messages `$M`, while others define the software redundancy `$N`.

On the other hand, we need to define the system fault assumption in terms of 1) which MRS components can be affected by faults and in which states, 2) the fault occurrence rate, and 3) the maximum number of faults that can concurrently affect the MRS. The Payloads are the identified components where the faults can occur, so they are annotated in the DD with the fault occurrence rate (`$ft_rate`). Moreover, in the Payload SM, we identified the states where the fault can lead to a failure. Consequently, we introduced a new transition, the one stereotyped as `DaStep`. Note that its occurrence rate annotation is just a duplicate of the one in the DD, since they refer to the same value. Finally, we do not need an explicit annotation for the maximum number of faults, since the design *per se* allows MRS to mask up to $N - K$ software faults.

The rest of the annotations (dark grey ones) concern the quantitative MARTE stereotypes needed for carrying out the analysis. The SD supports the MRS workload definition, which is closed with population `$M`, which is the number of requests from the clients. The SD is stereotyped as `GaScenario` and the scenario mean response time is specified. The response time accounts for the time elapsed from the client request until a positive ack is replied. The timing duration of the activities are annotated in the SCs with the `hostDemand` tag (`GaStep` stereotype). For the `GaStep` activities, we have specified either constant durations, e.g., the time-out, or mean duration, e.g., scan & decipher. Moreover, to evaluate the robustness of the MRS, we annotate in the MR state-machine the throughput (`$Xfilter`) of the `GaStep` transition `[value != OK]`. This metric corresponds to the number of corrupted files per second the service can detect. All the input and output parameters are gathered in the analysis context of the UCD.

Finally, it is worth to note that some input parameters, such as the fault occurrence rate (`$ft_rate`), will allow us to carry out sensitivity analysis by assuming different values during the analysis.

7 Analysis and Assessment with MARTE-DAM

In this section we describe the quantitative analysis and the assessment of the MRS case study. In general, the analysis aims to evaluate numerically the dependability and performance metrics, specified in the UML annotated model, and to interpret the metric values in the application domain. On the other hand, the assessment provides an indication on how to change the design or set the service parameters to guarantee that the system meets its non-functional requirements. In MRS, the metrics of interest are the service steady state availability and response time, and the rate of filtered messages. The latter used to evaluate robustness of the MRS under different payloads fault assumptions. The objective of the MRS assessment is to find a (range of) value(s) to be assigned to the time-out duration parameter to ensure a good trade-off between the service availability level (dependability metric) and the service response time (performance metric).

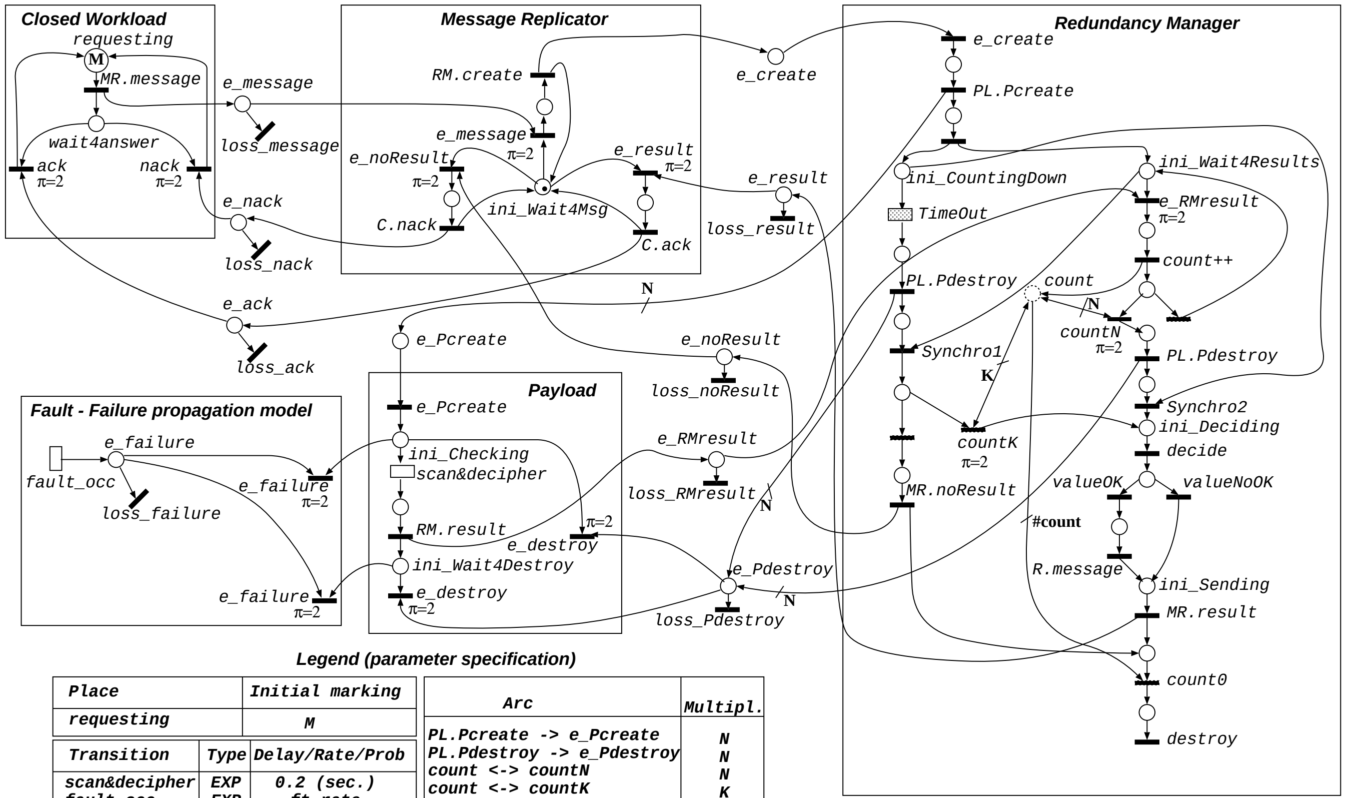
7.1 Getting a formal model from the UML specification

The analysis of the MRS has been carried out customizing the approach in [35] to the dependability analysis domain. In [35], a transformation method from UML SMs to Generalized Stochastic Petri Nets (GSPN) [3] was proposed. The method aims at obtaining a performance model, amenable to be analysed with GSPN solution techniques, from the annotated UML specification and provides a formal semantics of UML SMs in terms of GSPNs. The proposed semantics is compositional: the GSPN model of the system is obtained by composing the GSPN sub-models of the single SMs, by using standard Petri net operators.

The detailed translation of UML SMs into GSPN sub-models is beyond the scope of this paper. Instead, we focus on the most interesting aspects of the previous approach, and on their customization, in order to get a Deterministic and Stochastic Petri Net (DSPN) [4] model for the MRS case study. DSPN is an extended version of GSPN, where timed transitions can be either deterministic (i.e., characterized by a constant firing delay), or exponential (i.e., with firing delay represented by a random variable with negative exponential distribution).

The DSPN model of the MRS is shown in Figure 15, where the DSPN sub-models representing the three SMs in Figure 14 (i.e., Message Replicator, Redundancy Manager and Payload), can be identified. Besides, there are DSPN sub-models representing the closed workload and the fault-failure propagation in the payloads. All DSPN sub-models are characterized by interface places, labeled as `e_ev`, that represent mailboxes of events `ev`. The DSPN model has been obtained by merging the interface places with matching labels of the different DSPN sub-models.

The Message Replicator, Redundancy Manager and Payload of the DSPN model have been automatically derived using the ArgoSPE [6] tool, which



Legend (parameter specification)

Place	Initial marking
requesting	M

Transition	Type	Delay/Rate/Prob
scan&decipher	EXP	0.2 (sec.)
fault_occ	EXP	ft_rate
TimeOut	DET	TO (sec.)
valueOK	IMM	1-pb_NoOK
valueNoOK	IMM	pb_NoOK

Arc	Multipl.
PL.Pcreate -> e_Pcreate	N
PL.Pdestroy -> e_Pdestroy	N
count <-> countN	N
count <-> countK	K
count -> count0	#count

Fig. 15 DSPN model of the message redundancy service.

implements the approach in [35]. Considering that by default ArgoSPE creates exponentially distributed transitions, according to the GSPN definition, only one minor manual change was necessary to set the (dotted) transition *Timeout* as deterministic. However, the rest of the DSPN model was manually derived since it contains the MARTE-DAM extensions (i.e., closed workload and fault-failure propagation sub-models) which are not currently implemented in ArgoSPE. Some details of interest regarding this manual derivation are discussed in the following paragraph.

The closed workload DSPN sub-model, for example, is derived from the MARTE annotation *GaWorkloadEvent*, in the SD in Figure 13: the initial population variable `$M` has been used to define the initial marking M of place *requesting*. On the other hand, the fault-failure propagation DSPN sub-model represents the fault occurrence in the payloads together with its effect (i.e., failure) in the affected components. This sub-model results from the mapping of the DAM annotations in the DD and in the payload SM. The fault occurrence (DaVariant `fault.occurrenceRate` tagged-value, DD) is represented by the timed transition *fault_occ*. Its firing produces a token in the *e_failure* place, which may enable the *e_failure* transitions when the payload is either scanning & deciphering the message or waiting for a destroy event (DaStep `kind` and `failure` tagged-values, payload SM).

Observe that, the DSPN model is characterized by several parameters, summarized in the legend in Figure 15. In particular, transition firing delays are defined by the MARTE *hostDemand* tagged-values attached to the SM do-activities GaSteps: mean values are mapped to mean firing delays of exponential transitions (e.g., *scan&decipher* in the payload DSPN sub-model) and constant values are mapped to firing delays of deterministic transitions (e.g., *TimeOut* in the RM DSPN sub-model). In [35], a probabilistic translation of SM guarded transitions is proposed: e.g., the SM guarded transitions `[value!=OK]` and `[value=OK]` in the RM SM, are mapped onto a free-choice conflict between DSPN immediate transitions *valueNoOK* and *valueOK* (RM DSPN sub-model), whose weights (*pb_NoOK* and *1-pb_NoOK*, respectively) are defined considering the MARTE GaStep tagged-value `prob` (i.e., the variable `$pb_NoOK`).

According to the objective of the analysis, the metrics of interest are the service steady state availability and response time, and the rate of filtered messages. The *steady state availability* is a dependability metric and represents the percentage of times the system is able to provide the MRS when requested. The service is correctly performed when either the message is eventually delivered to the receiver or the message is filtered by the redundancy manager (since it is considered corrupted). The service downtime corresponds to each time RM is not able to decide after a time-out, due to an insufficient number of results (less than three) provided by the Payloads. The metric can be computed as the ratio between the throughput of the GaSteps `result(value)/Client.ack()` and `message(receiver,file)/RM.create (receiver,file)` (see the MARTE annotations in Figure 12 - UCD - and in Figure 14 - Message Replicator SM). In the DSPN model (Figure 15 -

MR DSPN sub-model), this metric is mapped onto the ratio between the throughputs of transition *e_result* - that represents the reception of the service outcome from the RM, i.e., either delivered or filtered - and transition *e_message* - that models the reception of the service request from the client.

The *response time* is a performance metric and corresponds to the mean time from client request to the reception of a positive ack. It is defined, by applying Little's operational law [20], as the ratio between the workload M and the throughput of the GaStep `result(value)/Client.ack()` (see the MARTE annotations in Figure 13 and in Figure 14 - Message Replicator SM). In the DSPN model (Figure 15), the metric is mapped onto the ratio between the initial marking of place *requesting* and the throughput of transition *e_result* (MR DSPN sub-model).

Finally, the *rate of filtered messages* is a performance metric that gives the number of incorrect messages per second detected by the RM, and then not delivered to the final destination. It provides an indication on the robustness of the MRS and it is specified as the throughput of the GaStep `[value != OK]` of the RM SM (Figure 14). In the DSPN model (Figure 15 - RM DSPN sub-model), the metric is mapped onto the throughput of transition *valueNoOK*.

7.2 Simulation of the DSPN model and initial assessment of the MRS

The DSPN model has been used to compute the metrics of interest via simulation: in particular, we run the steady-state discrete-event simulator implemented in the GreatSPN tool [25] and all the metrics have been computed setting a confidence interval of 99% and an accuracy of 15%. The simulated DSPN model assumed an implementation of the service with a 4-redundancy level ($N = 4$), a 1-fault tolerance level ($K = 3$) and one client requesting the MRS ($M = 1$).

Considering first, the steady state availability and response time metrics, we carried out sensitivity analysis by varying the values of the fault occurrence rate ($ft\_rate \in [1ft/s, 1ft/yr]$) and the time-out duration ($TO \in [0.1s, 1s]$). The probability of invalid message detection has been set to a fixed value $pb\_NoOK = 0.5$; indeed, this parameter does not affect the metrics above. In the light of the simulation results obtained for these metrics, we will assess certain aspects of the system. This assessment is now a manual process and requires knowledge about the problem domain where the results are interpreted (MRS in this case). However, specific assessment techniques able to automate decisions about setting parameters or design changes will be addressed in future work.

Figure 16(A) shows the steady state availability of the MRS versus the time-out duration and the payloads fault occurrence rate. Considering that the mean time to scan and decipher the message by a payload is set to 0.2 seconds, we can observe that the longer the time-out duration is, the greater the system availability; and it becomes closer to 100% when $TO \geq 0.5$ seconds. Indeed, the longer the time-out duration is, the lower the probability

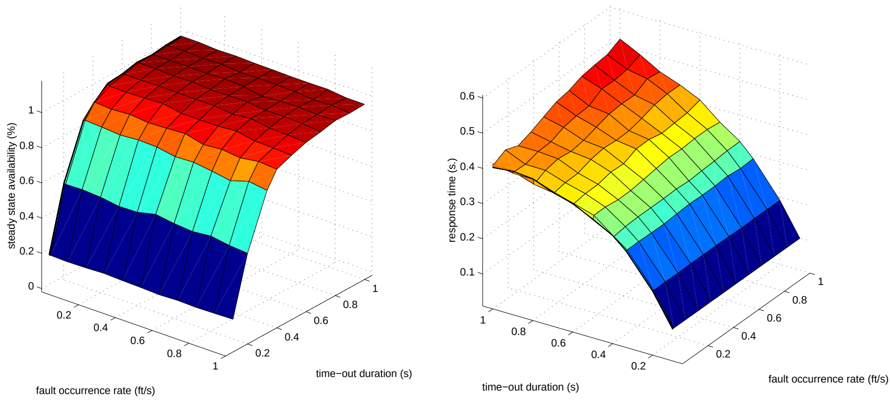


Fig. 16 (A) availability, (R) response time.

of false alarms becomes, that is, a time-out expires but no failure has actually occurred in the payloads. On the other hand, when fixing a time-out value and moving along the fault occurrence rate axis in Figure 16(A), the curve is basically a constant line. This fact leads to the conclusion that the fault occurrence rate in the payloads does not significantly affect the service availability. Certainly, this suggests to the software analyst to set the TO parameter to a value greater than 0.5 seconds, in order to guarantee a service availability of at least 91%.

Figure 16(R) shows the response time of the MRS versus the time-out duration and the payloads fault occurrence rate. At first sight, the result seems to be counter-intuitive: the longer is the time-out duration w.r.t. the time required by the payloads to scan and decipher a message (0.2 seconds), the smaller should be the response time: indeed, no false alarms should occur and the response time should be influenced only by the time to send a positive ack. Nevertheless, the computed results can be explained considering the possibility of payload failures. Indeed, when several payloads concurrently fail, the system can send a (negative) ack to the client only after the time-out expiration. So, under payload fault assumption, the longer is the time-out duration the longer it takes to send a final response to the client, therefore increasing the overall (positive) response time. On the other hand, the observed metric is influenced by the payloads fault occurrence rate when the time-out duration is greater than 0.6 seconds. The assessment suggests to set the TO parameter to a value less than 0.6 seconds to guarantee a reasonable response time (i.e., less than 0.4 seconds), independently of the assumption on payload fault occurrence rate. Then, considering both the dependability and performance requirements (under the stated assumptions for redundancy and fault-tolerance levels, and the system workload), the optimal assessed values for the time-out duration should fall in the interval $[0.5s, 0.6s]$.

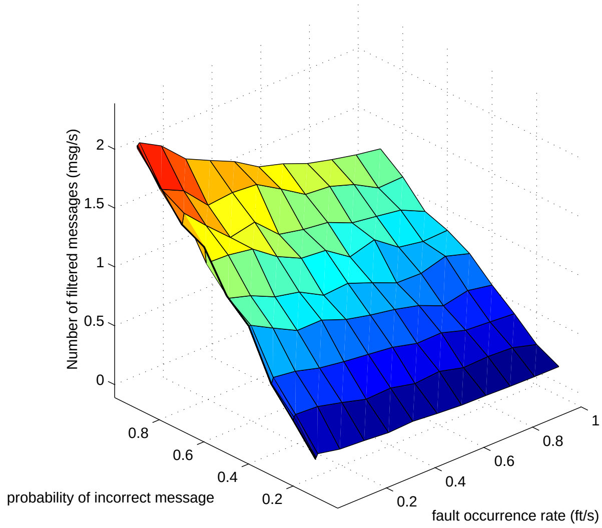


Fig. 17 Rate of filtered messages

Unlike the aforementioned metrics, the rate of filtered messages is affected by the *pb_NoOK* parameter: so we have conducted sensitivity analysis with *pb_NoOK* ranging between $[0.1, 0.9]$ and under different fault occurrence rate assumptions. As a result of the time-out duration previously assessed, the TO parameter has been set to 0.5 seconds, since this value provides a good trade-off between the service availability and response time. Figure 17 shows the metric versus the probability of incorrect message detection and the payloads fault occurrence rate. As expected, the rate of filtered messages is in direct proportion to the *pb_NoOK* parameter, i.e., it increases when the probability of detecting an incorrect message becomes higher. We can observe, also, that the higher the fault occurrence rate is, the lower the rate of filtered messages: the number of filtered messages/second decreases by about 45% when the fault occurrence rate changes from one fault/year to one fault/s. This result is due to the impossibility of deciding on the message integrity because of payload failures; indeed, since the payloads fail more frequently, Redundancy Manager does not receive enough results from the payloads to formulate a decision. In this case, it is impossible to assess how to set the parameters for optimizing the number of filtered messages, because such parameters (i.e., probability of incorrect message and fault occurrence rate) are not managed by the software analyst.

8 Conclusion

In this paper we have proposed a profile to support dependability modeling and analysis of UML designs. The proposed profile is compliant with the standard MARTE profile, and has been built considering current dependability standards. We have defined the profile by following the approach proposed by Selic [44] and applying the patterns from Lagarde [29]. We have applied the MARTE-DAM profile to the modelling, analysis and assessment of a message redundancy system (MRS). Such a case study gives an insight on how the MARTE-DAM profile can be used to derive a Deterministic and Stochastic Petri Net (DSPN) model for quantitative performance and dependability assessment.

The case study showed why dependability annotations for UML are needed, and illustrated the usefulness of the proposed DAM profile. In order to assess the availability of MRS, we had to consider fault and failure assumptions, as well as software and hardware redundancies. This kind of metrics and input parameters were defined using DAM extensions, while MARTE worked for performance extensions. As expected, both notations easily fit together into the UML design. No current standard profile would have been able to carry out the case study annotations: MARTE because of its lack of dependability concepts and QoS&FT because of its complex heavy-weight annotation mechanism.

To the best of our knowledge, this is the first attempt to provide a common domain model for different dependability communities. We consider the profile as an open proposal, subject to future refinements and extensions to address particular issues for the different dependability aspects.

We envision a great potential for our proposal, which offers a common modeling support intended to cover many existing approaches that have been developed separately. For instance, the existing UML-based approaches for analyzing dependability aspects of software systems surveyed in Section 3 can benefit directly from the proposed profile.

8.1 Future work

Among the challenges raised by a rich profile such as MARTE, one could mention the large amount of stereotypes and tags, and the richness of VSL for expressing non-functional properties. Obviously, extending MARTE by DAM only increases such challenges. From our point of view, tools are absolutely necessary in order to manage the complexities of the profiles and of the analysis techniques. We have started to develop a prototype with the tool [33] to manage our profile. From this implementation, we hope to learn lessons about real world applications, and to gain insight on how to effectively manage a UML model with a large number of NFPs, and to discern among annotations with different analysis purposes.

Future work also includes the implementation of model transformations taking as input UML+DAM models and producing different dependability models. We will follow some of the existing approaches from those surveyed in literature. On the other hand, there are works that although focussed exclusively on modeling dependability, can also benefit from the profile by using its annotations or even by extending it. Among such works, there are methodologies for collecting dependability requirements, (e.g., safety domain requirements in [5]), or others targeting certification according to standard software requirements [45]. Finally, there exist traditional works in dependability analysis of software systems outside the UML umbrella. In this case, the challenge is not how to integrate the DAM profile, but rather how to integrate the UML within the respective methodologies.

Acknowledgment

Authors would like to thank the anonymous referees and the associate editors, who reviewed the paper and helped to improve it. This research was supported by the project DPI2006-15390 of the Spanish Ministry of Science and Technology, the Natural Sciences and Engineering Research Council of Canada (NSERC) through its Discovery and Strategic grants, and by the project PaCo (Performability-Aware Computing: Logics, Models, and Languages) of the Italian Ministry of the University and Research.

References

1. N. Addouche, C. Antoine, and J. Montmain. UML models for dependability analysis of real-time systems. In *Proc. International Conference on Systems, Man and Cybernetics*, volume 6, pages 5209 – 5214. IEEE CS., Oct. 2004.
2. N. Addouche, C. Antoine, and J. Montmain. Methodology for UML modeling and formal verification of real-time systems. In *International Conference on Computational Intelligence for Modelling Control and Automation (CIMCA 2006), International Conference on Intelligent Agents, Web Technologies and Internet Commerce (IAWTIC 2006)*, page 17. IEEE Computer Society, 2006.
3. M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets.* J. Wiley, 1995.
4. M. Ajmone Marsan and G. Chiola. On Petri nets with deterministic and exponentially distributed firing times. In *Advances in Petri Nets 1987, covers the 7th European Workshop on Applications and Theory of Petri Nets*, pages 132–145, London, UK, 1987. Springer-Verlag.
5. K. Allenby and T Kelly. Deriving safety requirements using scenarios. In *5th IEEE International Symposium on Requirements Engineering (RE 2001)*, pages 228–235. IEEE Computer Society, 2001.
6. ArgoSPE. `http://argospe.tigris.org`. University of Zaragoza, 2006.
7. A. Avizienis et al. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. on Dependable and Secure Computing*, 1(1):11–33, 2004.
8. S. Bernardi, S. Donatelli, and G. Dondossola. Methodology for the generation of the modeling scenarios starting from the requisite specifications and its application to the collected requirements. IST Project 25434 DepAuDE - Deliverable D1.3b, 2002.
9. S. Bernardi, S. Donatelli, and G. Dondossola. A class diagram framework for collecting dependability requirements in automation systems. In *Proc. of 1^{st} Int.l Symposium on Leveraging Applications of Formal Methods*, Cyprus, October 2004.

10. S. Bernardi and J. Merseguer. QoS Assessment via Stochastic Analysis. *IEEE Internet Computing*, pages 32–42, May-June 2006.
11. S. Bernardi, J. Merseguer, and D. Petriu. A UML profile for Dependability Analysis and Modeling of Software Systems. Technical Report RR-08-05, Universidad de Zaragoza, Spain, 2008. http://www.di.unito.it/∼bernardi/DAMreport08.pdf.
12. S. Bernardi, J. Merseguer, and D.C. Petriu. Adding dependability analysis capabilities to the MARTE Profile. In Krzysztof Czarnecki, Ileana Ober, Jean-Michel Bruel, Axel Uhl, and Markus Völter, editors, *Proc. of 11^th International Conference on Model Driven Engineering Languages and Systems (MoDELS)*, volume 5301 of *Lecture Notes in Computer Sciences*, pages 736–750, Toulouse, France, 2008. Springer.
13. A. Bobbio, E. Ciancamerla, G. Franceschinis, R. Gaeta, M. Minichino, and L. Portinale. Sequential application of heterogeneous models for the safety analysis of a control system: a case study. *Reliability Enginering and System Safety*, 81:269–280, 2003.
14. A. Bondavalli et al. Dependability analysis in the early phases of UML-based system design. *Int. Journal of Computer Systems Science & Engineering*, 16(5):265–275, 2001.
15. International Electrotechnical Commission. IEC-60300-3-1 standard: Dependability management.
16. International Electrotechnical Commission. IEC-61508 standard: Functional Safety of Electrical/ Electronic/ Programmable Electronic safety related problems.
17. V. Cortellessa and A. Pompei. Towards a UML Profile for QoS: a contribution in the reliability domain. In *Proceedings of the Fourth International Workshop on Software and Performance (WOSP'04)*, pages 197–206, January 2004.
18. M. Dal Cin. Extending UML towards a Useful OO-Language for Modeling Dependability Features. In *Proc. of 9th Int.l Workshop on Object-Oriented Real-Time Dependable Systems*, pages 325–330, Capri Island, Italy, October 2003. IEEE CS.
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)*, Cambridge, MA, USA, November 2002.
20. P.J. Denning and J.P. Buzen. The Operational Analysis of Queueing Network Models. *ACM Comput. Surv.*, 10(3):225–261, 1978.
21. M. Evans, N. Hastings, and B. Peacock. *Statistical distributions.* Wiley, New York, 2000.
22. K. Goseva-Popstojanova et al. Architectural-level risk analysis using UML. *IEEE Transactions on Software Engineering*, 29(10):946–960, 2003.
23. V. Grassi, R. Mirandola, and A. Sabetta. From design to analysis models: a kernel language for performance and reliability analysis of component-based systems. In *Proceedings of the Fifth International Workshop on Software and Performance (WOSP'05)*, pages 25–36, July 2005.
24. V. Grassi, R. Mirandola, and A. Sabetta. Filling the gap between design and performance/reliability models of component-based systems: A model-driven approach. *Journal of Systems and Software*, 80(4):528–558, 2007.
25. GreatSPN. `http://www.di.unito.it/∼greatspn`. University of Torino, 2002.

26. A. Hassan, K. Goseva-Popstojanova, and H. Ammar. UML Based Severity Analysis Methodology. In *Proc. of Annual Reliability and Maintainability Symposium (RAMS 2005)*, Alexandria, VA, January 2005.
27. J. Jürjens and S. Wagner. Component-based Development of Dependable Systems with UML. In Atkinson et al., editor, *Component-Based Software Development*, volume 3778 of *LNCS*, pages 320–344. Springer-Verlag, 2005.
28. J. Jürjens. Developing safety-critical systems with UML. In *Proc. of UML 2003, San Francisco*, volume 2863 of *LNCS*, pages 360–372. Springer-Verlag, October 2003.
29. F. Lagarde et al. Improving UML profile design practices by leveraging conceptual domain models. In *22nd Int.l Conf. on Automated Software Engineering, Atlanta (USA)*, pages 445–448. ACM, November 2007.
30. N.G. Leveson. *Safeware*. Addison-Wesley, 1995.
31. Michael R. Lyu, editor. *Handbook of Software Reliability Engineering*. IEEE Computer Society Press, 1996.
32. M.R. Lyu. *Software Fault Tolerance*. John Wiley & Sons, Ltd., 1995.
33. MagicDraw. `http://www.magicdraw.com/`. No Magic, Inc., 2009.
34. I. Majzik, A. Pataricza, and A. Bondavalli. Stochastic Dependability Analysis of System Architecture Based on UML Models. In *Architecting Dependable Systems*, LNCS 2677, pages 219–244. Springer-Verlag, 2003.
35. J. Merseguer, S. Bernardi, J. Campos, and S. Donatelli. A compositional semantics for UML State Machines aimed at performance evaluation. In Silva M., Giua A. and Colom J.M., editor, *WODES02: 6^{th} International Workshop on Discrete Event Systems*, pages 295–302, Zaragoza, Spain, October 2002. IEEE Computer Society.
36. S. Mustafiz, X. Sun, J. Kienzle, and H. Vangheluwe. Model-driven assessment of system dependability. *Journal of Software and Systems Modeling*, 7(4):487–502, 2008.
37. Object Management Group. *UML Profile for Schedulability, Performance and Time Specification*, January 2005. V1.1, f/05-01-02.
38. Object Management Group. *A UML profile for Modeling and Analysis of Real Time Embedded Systems, Beta 1*, August 2007. Adopted Spec., ptc/07-08-04.
39. Object Management Group. *UML Profile for Modeling Quality of Service and Fault Tolerant Characteristics and Mechanisms*, April 2008. V1.1, f/08-04-05.
40. G.J. Pai and J.B. Dugan. Automatic Synthesis of Dynamic Fault Trees from UML system models. In *Proc. of 13th Int. Symposium on Software Reliability Engineering*, pages 243–256, Annapolis, MD, USA, November 2002. IEEE CS.
41. A. Pataricza. From the General Resource Model to a General Fault Modelling Paradigm? Workshop on Critical Systems, held within UML'2000, 2000.
42. A. Pataricza et al. UML-based design and formal analysis of a safety-critical railway control software module. In G. Tarnai and E. Schnieder, editors, *Proc. of FORMS'03*, pages 125–132, Budapest (Hungary), May 2003.
43. R.A. Sahner, K.S. Trivedi, and A. Puliafito. *Performance and Reliability Analysis of Computer Systems: An Example-Based Approach Using the SHARPE Software Package*. Kluwer Academic Publishers, 1996.
44. B. Selic. A systematic approach to domain-specific language design using UML. In *10th IEEE Int.l Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'07)*, pages 2–9, 2007.
45. G. Zoughbi, L. Briand, and Y. Labiche. A UML Profile for Developing Airworthiness-Compliant (RTCA DO-178B), Safety-Critical Software. In *Proceedings of Models 2007*, volume 4735 of *LNCS*, pages 574–588. Springer-Verlag, 2007.