An UML profile for dependability analysis and modeling of software systems

Simona Bernardi, José Merseguer and Dorina Petriu

Technical Report, number RR-08-05

Registered in Departamento de Informática e Ingeniería de Sistemas,

University of Zaragoza, Spain

May 2008

Conformance signature

Head of the Department

Dr. Victor Vinyals

S.Bernardi is with Dipartimento di Informatica,Università di Torino, Italy.

J.Merseguer is with Departamento de Informática e Ingeniería de Sistemas, Universidad de Zaragoza, Spain.

D.Petriu is with Department of Systems and Computer Engineering,Carleton University, Canada.

COPYRIGHT

Copyright © 2008, by the authors.

The authors retain the copyright © of this publication.

TERMS OF USE

The material contained in this document can be used by third parties if they properly reference it and/or the articles, from the authors, derived from this work.

Abstract

In this document we define the Dependability Analysis Modelling profile, namely DAM profile. The process of deriving a DAM profile has been going through several steps. First of all, an in depth analysis of the literature has been carried out, in order to collect in a checklist the information requirements for the profile. Then a two-step approach for the profile definition has been followed. In the first step, a Dependability Analysis (DA) domain model is defined, in terms of a structured set of UML Class Diagrams, where the basic concepts supporting dependability analysis are represented. The domain model is assessed with respect to the works in the literature considered before passing to the second step. In the second step, the DAM profile is defined considering the domain model. The DAM profile is then assessed with respect to the checklist of information requirements.

I. APPROACH OVERVIEW

The process of deriving an UML profile for dependability analysis of software systems has been characterized by several tasks that can be summarized as follows:

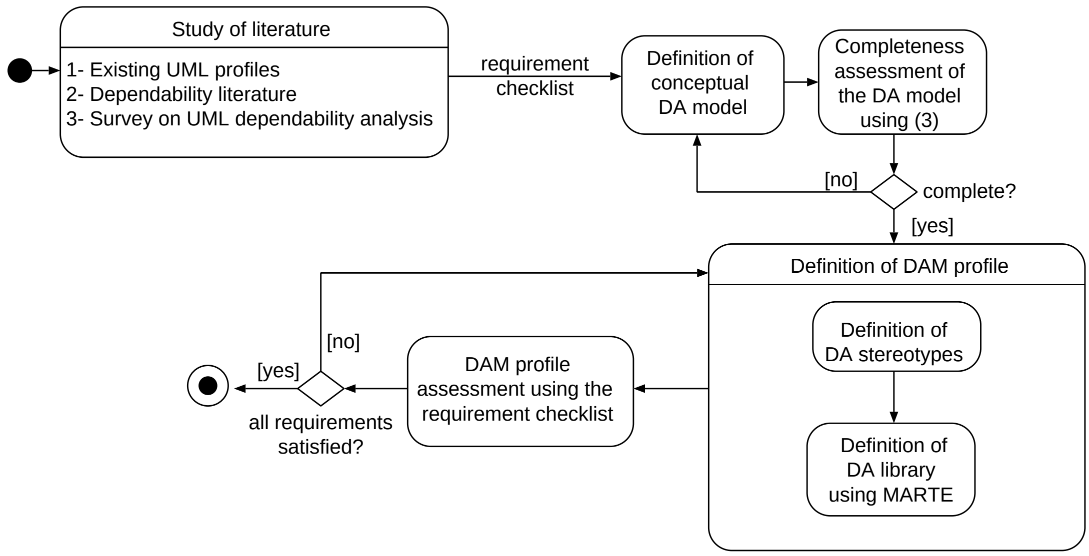

Fig. 1. Definition of the DAM profile

- *Study of literature* 1) The existing standard UML profiles for the analysis of non functional properties of software systems have been analyzed, in particular the SPT profile [27], the QoS&FT profile [28] and the MARTE profile [29]. None of them provides a comprehensive support for the dependability analysis, especially from the quantitative point of view (for SPT and QoS&FT, see the comparative work [7]). This lack of standard dependability analysis support has been the main motivation of this proposal. 2) We have also investigated the literature on the dependability main concepts and taxonomy (e.g.,Laprie et al. [3], Leveson [23]) as well as on standard methods used for the quantitative assessment of dependability (e.g., IEC standard [10]). 3) We have made a survey of the works in the literature proposing dependability modelling and analysis of UML system specifications (about 19 works). The output of this preliminary step is a a checklist of requirements that a UML profile for dependability analysis should satisfy.

- *Definition of conceptual Dependability Analysis (DA) model* We have defined a conceptual DA model to represent the main dependability concepts from the literature (2,3). The DA model consists of several UML class diagrams, organized in packages. The construction of the DA model goes to several refinement steps to consider all the works of the survey (3). The final DA model is described in detail in section III.
- *Completeness assessment of the DA model* The assessment of the DA domain model consists in verifying that all the concepts considered in the work survey (3) have been included. If a concept has not been considered, we either repeat the refinement step or we provide a motivation of its exclusion from the DA model. The assessment has been detailed in sub-section III-A.
- *Definition of the Dependability Analysis Modelling (DAM) profile* Using the DA model we define 1) the DA extensions, that is stereotypes and tags, and the 2) DAM library. The objective is to introduce a small set of stereotypes, that can be easily used by the software analyst, so there is not a one-to-one mapping between the conceptual classes of the DA model and the DAM stereotypes. The DAM library has been defined by importing the MARTE library and it consists of complex DA types, mapped from some DA model classes, and of basic DA types. The DAM profile is presented in section IV.
- *DAM profile assessment using the requirement checklist* The assessment of the DAM profile consists in verifying whether the requirements of the checklist are satisfied. If a requirement is not met by the DAM profile, we go back to the previous step in order to refine the latter. In subsection IV-C the DAM profile assessment is detailed.

II. Study of literature

Several approaches have been proposed, in the last few years, aimed at extending UML to support dependability modeling and analysis of software systems. In particular, UML profiles for safety critical systems are proposed in the works [34], [35], which have different goals with respect to ours.

In [35] a UML profile is defined for the elicitation of safety requirements of aerospace software systems, in order to improve the communication among the system stake holders as well as to generate automatically certification-related information from UML models. In [34] a UML profile is proposed, instead, for the modeling of safety-critical embedded real-time control systems and stereotypes are used to incorporate PEARL and Function Blocks constructs.

The recent survey [20] defines an interesting framework for the comparison of reliability/availability analysis methods in the literature for software architectures. Some of the mentioned works, address the issue of deriving dependability models from UML-based specification and will be considered in this section.

In the following, we focus on the works that aim at providing a support for the quantitative dependability analysis of UML based design. The study is carried out from a critical perspective, with the purpose of building on them our proposal of a common profile for the quantitative prediction of software system dependability.

We will then present the related work with the support of the following checklist of *information requirements* for a dependability profile, that we have drawn up considering the literature on dependability modeling and analysis:

(**IR1**) Identification of the dependability analysis context, in particular the types of non functional requirements to be assessed, i.e., reliability, availability, safety.

(**IR2**) Specification of dependability requirements in terms of upper/lower bounds, like the maximum (minimum) system or component reliability required, the minimum availability or safety level required.

(**IR3**) Specification of dependability measures to be estimated during the analysis. The set of supported measures should include, at least, the reliability/unreliability probability distribution functions, the system failure probability, MTTF, the time to repair distribution function, MTTR, the instantaneous and the steady state availability, the safe mission time, the risk factor associated to a failure/hazard.

(**IR4**) Specification of the dependability input parameters that are needed by the standard techniques for the quantitative evaluation of the system dependability. The dependability input parameters characterize, from a quantitative point of view:

- (**IP1**) The processes leading to service failures and accidents. In particular, the threats of dependability that may affect both hardware and software resources (e.g., the probability of fault occurrence, the error latency, the probability of failure, the hazard severity).
- (**IP2**) The repair/reconfiguration processes, in case of repairable systems/components, that remove basic or derived failures from the system (e.g., repair and restoration rates).

(**IR5**) Specification of different fault behaviors depending on their timing persistence (i.e., permanent, transient and intermittent) and of fault occurrence assumptions (e.g., single fault assumption).

(**IR6**) Specification of the error propagation between system components that interact with each others.

(**IR7**) Specification of the system failure modes with respect to different point of views: the domain, i.e, content, (early, late) timing failures, halting or erratic failures (**Dom**); the detectability, i.e., signaled or unsignaled failures (**Det**); the consistency, i.e., consistent or unconsistent failures (**Con**); the consequences, i.e., minor, marginal, critical and catastrophic failures (**Cons**); and, when multiple failure are considered, the failure dependency, i.e., independent or dependent failures (**Dep**).

(**IR8**) Specification of the hazards leading to accidents, in terms of their basic components (such as the severity, the likelihood of hazard occurring, the duration, the accident likelihood and the risk).

(**IR9**) Specification of (uncorrect) behavior of system components affected by threats as well as the reconfiguration activities that restore the system component states. In particular, identification of erroneous/ failure/ hazardous states, threat events, recovery triggers and actions.

(**IR10**) Specification of redundant hardware and software components. In particular, the number of copies existing in a redundant structure, the minimum number of components required in a redundant structure for a dependable system, and the type of spare redundancy.

Considering the analysis context (**IR1**), the works [1], [2], [4]–[6], [31] aim at supporting "generic" dependability analysis of UML software system specification, without emphasis on specific dependability attributes. Most of the works we examined, focus on reliability analysis of UML system specifications [9], [11], [13], [14], [17], [18], [21], [26], [30], only few of them [9], [13], [26], [30] provide a support also for availability analysis. Finally, few efforts have been devoted to safety analysis of UML-based models [16], [19], [22], [32].

Table I summarizes the contributions of the mentioned works to the *information requirements* checklist.

Pataricza [31] extends the General Resource Modeling package of the SPT profile, where the basic concepts of quality of service (QoS) characteristic and value are introduced, with the notion of faults and errors to support

	IR1	IR2	IR3	IR4		IR5	IR6	IR7					IR8	IR9	IR10
				IP1	IP2			Dom	Det	Con	Cons	Dep			
Pataricza [31]						✓	✓	✓						✓	
Addouche et al. [1], [2]				✓	✓							✓		✓	
Bernardi et al. [4], [5]		✓	✓	✓		✓	✓	✓			✓				
Bernardi-Merseguer [6]		✓	✓	✓		✓		✓							
Bondavalli et al. [9], [26]	R,A		✓	✓	✓	✓	✓					✓		✓	✓
DalCin [13]	R,A	✓													
Pai-Dugan [30]	R,A			✓	✓		✓					✓		✓	✓
D'Ambrogio et al. [14]	R			✓											
Cortellessa-Pompei [11]	R			✓											
Grassi et al. [17], [18]	R			✓											
Jürjens et al. [21], [22]	R,S	✓		✓				✓					✓		✓
Pataricza et al. [32]	S				✓									✓	
Goseva et al. [16]	S		✓								✓		✓		
Hassan et al. [19]	S		✓								✓		✓		

Legend **IR1**: R=reliability, A=availability, S=safety.

TABLE I

CONTRIBUTIONS TO THE DEPENDABILITY INFORMATION REQUIREMENTS OF THE MENTIONED WORKS

the analysis of the effect of local faults to the system dependability. The work emphasizes the importance of including two phenomena in the system model: permanent and transient faults in the resources (**IR5**) and error propagation across the system to estimate which fault may lead to a failure (**IR6**). Moreover, it suggests the usage of QoS values to characterize the domain of system failures (**IR7-Dom**). Explicit fault injection behavioral models are also proposed to represent faults as special *virtual clients* that request service to components and that have higher priority than the other actual clients. Fault injectors can be used also to model constraints on fault occurrence, e.g., single fault assumption (**IR5**). The effect of their request causes a change of state of the server (that is the hardware component affected by the fault occurrence) which moves from normal states (state in which the system is well-functioning) to faulty ones, and to normal states again in case of transient faults (**IR9**).

Addouche et al. [1], [2] define a profile for dependability analysis of real-time systems that is compliant with General Resource Modeling package of the SPT profile. The UML extensions are used to annotate UML models with QoS characteristics and to derive probabilistic time automata for the verification of dependability properties via temporal logic formulas. Dependability input parameters of system resources (i.e., reliability, maintainability) are specified as tags (**IR4**). A pair of stereotypes is also defined to include probabilistic aspects of functioning and malfunctioning. The static model of the system is enriched with new stereotyped classes that are associated with each class representing a resource. The *Indicator* classes are characterized by attributes related to the state machines of the resource classes associated with, and their values represent the degraded or failure state of the resource classes. The *Cause* classes are characterized by attributes representing logical expressions of failure occurrence in the resource classes associated with (**IR9**). This mechanism can be used by the analyst to specify components state-based conditional failures (**IR7-Dep**). The negative aspect of the approach is that new classes need to be defined and introduced in the system model, beside the classes representing the actual system components, for dependability analysis purposes.

Bernardi et al. [4], [5] propose a set of UML Class Diagrams (CD) structured in packages, to collect dependability and real-time requirements and properties of automated embedded systems with the use of COTS FT mechanisms. The approach provides support for a semi-automatic derivation of dependability analysis models, such as Stochastic Petri Nets and temporal logic models. Three stereotypes are defined for class attributes in order to discriminate input parameters (**IR2**), including the component and functions failure criticality level (**IR7-Cons**), metrics (**IR3**) and upper/lower bound requirements (**IR4-IP1**). The classifications of dependability threats [3] are represented as CDs. In particular, fault classes include attributes that characterize the fault timing persistency (**IR5**), and failure are classified according to their impact on the automation system in halting, degrading and repairing failures (**IR7-Dom**). The most interesting class diagrams are the "FEF chains", that define the causal relationships among faults, errors and failures, the relationships between the dependability threats and the affected system components, and the error propagation relationship (**IR6**).

In [6] we devise a method to assess Quality of Service (QoS) of fault tolerant (FT) distributed systems via derivation of performability models from SPT annotated UML behavioral and deployment diagrams. The objective of the analysis is to evaluate the QoS of the FT strategy implemented in the system under late-timing failure assumption (**IR7-Dom**). The QoS is defined as a function of two non functional properties: one is strictly related to the FT effectiveness (i.e., the time to detect a failure) and the other is related to the cost of the FT (i.e., communication overhead). State machines are proposed for the quantitative characterization of faults as well as for the behavioral specification of different type of fault w.r.t. their timing persistency (**IR5**). UML extensions have been explicitly introduced, since the SPT profile does not support the specification of dependability parameters, such as fault frequencies and latencies. Nevertheless, the SPT compliance provides an easy solution for the discrimination of the type of usage of each attribute, i.e., requirement (**IR2**), metric (**IR3**) or input parameter (**IR4-IP1**).

The most comprehensive approach for reliability and availability analysis of UML specifications has been proposed by Bondavalli et al. [9], [26]. The authors use UML standard extension mechanisms (i.e., stereotypes and tags) for annotating dependability properties of software systems on UML design models. Through a model transformation process, Timed Petri Net models are then derived via an intermediate model, that captures the relevant dependability information from the annotated UML models. The proposed approach is compliant with the taxonomy and basic concepts defined in [3]. Although no support is given for the specification of dependability requirements to be assessed, several dependability parameters are defined and they can be associated to both hardware and software components. The set of input parameters (**IR4**) includes the fault timing occurrence, the percentage of permanent faults (**IR5**), the error latency for components with an internal state, and repair delay. The set of metrics (**IR3**) includes the reliability probability distribution function, MTTF, the steady state and the immediate availability.

The approach supports the specification of error propagation between components (**IR6**) by assigning a probability to the model elements representing either relationships (e.g., associations) or interactions between such components (e.g., communication paths, messages). Concerning the type of failures with respect to their dependency, both independent and dependent failures can be specified (**IR7-Dep**). In particular, it is possible to assign common failure mode occurrence tags to redundant components belonging to complex FT structures (**IR10**). Extensions for states and events of state machines representing the behavior of *redundancy manager* components are introduced, in order to discriminate normal and failure states and events (**IR9**). Such extensions are used to analyze the failure conditions of the FT structures. The main drawback of the UML extensions proposed by Bondavalli et al., is the introduction of unnecessary redundant information in the UML system model, since the specification of some parameters requires the joint use of more than one stereotype. For example, a node, that models a hardware component in UML, must be stereotyped as *hardware* and *stateful* to specify the error latency.

DalCin [13] proposes a UML profile for specifying dependability mechanisms, that is hardware/software components to be implemented or integrated in the real-time system to ensure fault tolerance. The proposed profile is aimed at supporting the quantitative evaluation of the effectiveness of the FT strategy adopted and it provides facilities for capturing stochastic reliability and availability requirements of such mechanisms (**IR2**). However, the profile lacks of a support to the modeling of the interactions among dependability mechanisms and the system components.

Pai-Dugan [30] present a method to derive dynamic fault tree from UML system models. A set of stereotypes and tags are introduced to enrich UML system models with information needed for the reliability and availability analysis. In particular, tags are used to define input parameters, such as the failure rate of system components, the coverage factors, restoration rates, the error propagation (**IR4, IR6**). The method supports the modeling and analysis of dependent failures, such as sequence failure dependencies (**IR7-Dep**), redundancies and reconfiguration activities. Several stereotypes are defined to represent different kinds of dependencies between system components and to model the type of spare components, e.g., hot, cold and warm spares (**IR10**). State machines, without specific UML extensions, are used to model reconfiguration activities (**IR9**).

The works [11], [14], [17], [18] address specifically the reliability analysis of UML-based design.

D'Ambrogio et al. [14] define a transformation of UML models (Sequence and Deployment diagrams) into fault tree models to predict the reliability of component-based software. Although no UML extension standard mechanisms are used, several UML model elements whose failure (basic events in fault tree models) can lead to the system failure (top-event in fault tree models) are identified. In particular, the basic events considered are the failure of nodes and communication paths and the failure of call actions, operations and return actions (**IR4-IP1**).

Cortellessa and Pompei [11] propose a UML annotation for the reliability analysis of component-based systems, within the frameworks of the SPT and QoS&FT profiles. Their work is built on the approach in [12], where Bayesian models are derived from UML annotated models to compute the system failure probability. Although no specific extensions are used for the annotation of the reliability requirements and metrics, a set of stereotypes and related tags are proposed for the specification of reliability input parameters (**IR4-IP1**). Such stereotypes are specialization of stereotypes defined in the General Resource Modeling package of the SPT profile. The most interesting input parameters considered are the *atomic* failure probabilities of software components (*REcomponent*) or (logical/physical) links (*RE connector*), that is the probability that a component, or connector, fails in a single invocation of it. There are no explicit annotations for the failure probability of hardware components.

The annotations defined in [11] are used by Grassi et al. [17], [18], that propose a model-driven transformation framework for the performance and reliability analysis of component-based systems. Similarly to [9], [26], Grassi et al. build 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. Despite of [11], Grassi et al. associate failure input parameters to both hardware and software components and consider both atomic failures and failure probability distribution functions (**IR4-IP1**).

Jürjens et al. define a safety [22] and reliability [21] check list, based on UML extension standard mechanisms, to support the analyst in the identification of failure-prone components in the software design. The works [21], [22] propose a similar approach, although they address distinct dependability attributes, i.e., safety and reliability, respectively. Most of the UML extensions are used to specify requirements on communication, such as the stereotype *guarantees* whose tag *goal* is of complex type and can express either a maximum duration allowed for data transmission, or a probability that eventually a data is delivered or a maximum probability of message loss. Several stereotypes can be applied to specify guarantees at subsystem level. An interesting aspect considered by Jürjens et al., is the possibility of specifying both requirements, via the *guarantees* stereotype (**IR2**), and failure/hazards assumptions, via the *risk, crash/performance, value* stereotypes (**IR4-IP1, IR8**), according to the failure domain, that is timing failure or content failure (**IR7-Dom**). Some extensions are also proposed for the specification of FT structures, such as the types of voting algorithms implemented within redundancy strategies (**IR10**).

The approaches [16], [19], [32] support the safety analysis of UML-based system models.

Pataricza et al. [32] use UML stereotypes to identify erroneous states and error correcting transitions in state machine diagrams (**IR4-IP2**). The approach proposes to integrate the normal and the faulty behavior of a system component in a single state machine (**IR9**). The enriched behavioral models can be used then in the analysis to evaluate the effect of local faults to system service, in particular, to identify the error propagation paths that lead to catastrophic failures.

Goseva et al. [16] devise a methodology for the risk assessment of UML models at architectural level. A Markov model is constructed to estimate the scenario risk factors from risk factors associated to software components and connectors. Although no explicit UML extensions are provided, several safety related parameters are introduced as: metrics obtained directly from UML models, e.g., dynamic complexity of a component, dynamic coupling of a connector (**IR3**), properties estimated via safety analysis techniques like FMEA, e.g., severity indices associated to components and connectors (**IR3, IR7-Cons, IR8**), and composite metrics defined as function of basic ones, e.g., risk factors associated to components, connectors, failure scenarios, use cases and the overall systems (**IR3, IR8**).

Hassan et al. [19] introduce a methodology for the severity analysis of software systems modeled with UML. The work integrates different hazard analysis techniques (FFA, FMEA and FTA) to identify system level and component/connector level hazards and to evaluate the cost of failure of system execution scenarios, software components and connectors (**IR7-Cons**). Like in [16], no UML extensions are provided but several safety parameters are introduced to calculate the desired safety metrics (**IR3,IR8**). The results of the hazard analysis are reported in UML models with the use of notes.

Tables X and XI, in the appendix A, detail the contribution of the mentioned works to the *information*

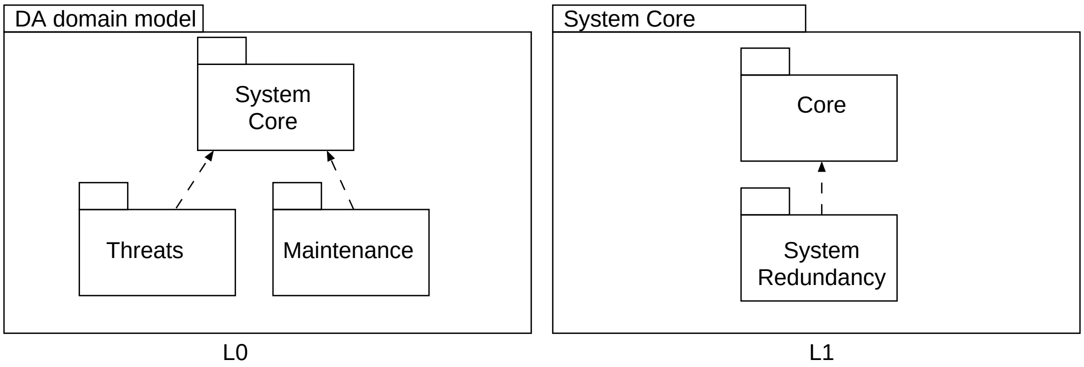

Fig. 2. Top-level package (L0), System Core package (L1)

requirements checklist drawn up at the beginning of this section. The Bondavalli et al.'s approach is the one that satisfies most of the information requirements of the checklist. In general, we can observe that more efforts have been devoted to support the reliability analysis of UML-based models. Concerning the works that focus on reliability and safety, we can deduce that the concepts of failures and of hazards are often used as synonymous. We can also notice that none of the works provide extensions to discriminate failure modes w.r.t. their detectability and consistency. Finally, only three proposals address the specification of dependent failures: Addouche et al. consider component state-based conditional failures. The contribution of Bondavalli et al. is limited to the special class of common cause of failures affecting redundant components, while in [30] non trivial dependent failures can be modelled, such as sequence dependent failures.

III. Dependability Analysis (DA) domain model

The aim of this domain model is to give support for dependability analysis of UML-based specifications and to provide the basis for adding a Dependability Profile to MARTE. The top-level package includes (Figure 2):

- System Core: represents the system to be analyzed, it is a component-based view of the system, according to [3] and [24]. Additional concepts are introduced for representing redundancy structures that are considered as part of the system structure that address (some) fault tolerant solutions. We do not aim at providing modelling support for fault tolerant architectures (this issue has been addressed in the UML QoS&FT profile [28]), rather we include redundancy concepts to provide a support for the dependability analysis in case of fault tolerance systems.
- Threats: introduces the concepts that represent the threat process that may affect the system. Such concepts are related to the system core (both the core concepts and the redundancy structure). Observe that modeling threats is necessary to carry out reliability and safety analysis. The adopted terminology, is slighly different in reliability domain and in safety domain. We have then added the abstract concept of *impairment* that can be refined for the specific analysis to be carried out.
- Maintenance: introduces those concepts that are necessary for availability analysis, basically the repair process from anomalous states. In [3] the term "maintenance" is introduced to indicate not only repairs but also modifications of the system that take place during its usage. So we can include also concepts

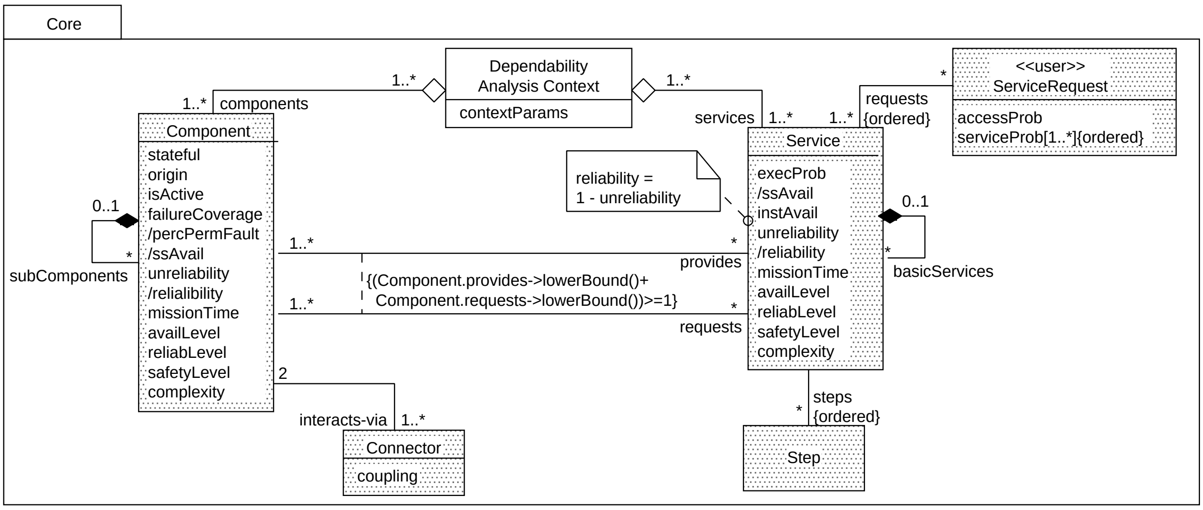

Fig. 3. Core model

related to system reconfiguration (topic dealt by Pay-Dugan in [30]). The concepts are related to the system core. Observe that modeling threat and maintenance is necessary to carry out availability analysis.

The Core model of the System Core Package (Figure 3) is a component-based description of the system to be analyzed, according to Laprie et al. [3] and [24]. This model will be useful at the next step of the definition of the DAM profile, for the identification of the UML model elements to be extended with dependability characteristics. There are several classes that can be considered as specialization of the General Quantitative Analysis Modeling package of MARTE: so this package could help MARTE people to integrate the dependability profile in MARTE.

TABLE II: Core model description.

Component	We consider both HW and SW components that may be affected by threats. A components provides and requests basic services and interact with the other components through connectors. A component must either provide or request at least one basic service (see OCL constraint between the two associations connecting Component and Service). A component may consist of a set of other components (which depends on the modeling abstraction level used).
Attributes	
stateful	(true) Faulty stateful components can be characterized by an error latency, so they can be restored before failure. (false) Faulty stateless components are considered as failed [9], [26].
origin	Discriminates between hardware and software components [9], [26].
isActive	(true) The component can perform its behavior autonomously and trigger behavior of other components [11].

Continued on next page

TABLE II – continued from previous page

failureCoverage	Percentage of failure coverage [30].
/percPermFault	Percentage of permanent faults [9], [26]. It can be derived from the association between Component and Fault (Threats package), and from the *persistency* attribute of the Fault class.
/ssAvail	Steady state availability (percentage) [4], [5]. The steady state availability can be defined as: MTTF/MTBF or MTTF/(MTTF+MTTR) or MTTF/(MTTF+recoveryDuration) [33]. It can be derived then from associations connecting Component with Failure, Repair (or Recovery) classes, and from the homonym attributes defined in the latter.
unreliability	Unreliability, that is the probability that the time to failure random variable is less or equal than time t (time dependent) [33].
/reliability	Reliability, that is the probability that the time to failure random variable is greater than time t (time dependent) or, in other words, that the component is functioning correctly during the time interval $(0, t]$ [33]. It is defined as 1-unreliability. It is a *survival function* [15].
missionTime	Time interval in which the component unreliability is lower than a preassigned threshold. It is the inverse function of unreliability.
availLevel	Availability level associated to the nines of availability. E.g., very high corresponds to 99,9% of ssAvail, etc.
reliabLevel	Reliability level.
safetyLevel	Safety level.
complexity	Complexity metric [4], [5], [16], [21], [22]. There are many complexity metrics in the literature (e.g., Halstead's Software Metric, McCabe's Cyclomatic Complexity). This attribute provides a quantitative characterization of the component complexity which is related to the component failure proneness.
Connector	We consider logical connectors that represent either potential or actual communications between components. Such connectors carri the error propagation between components.
Attribute	
coupling	Coupling metric [16]. It is related with error propagation proneness.
Dependability Analysis Context	This concept corresponds to one used in the GQAM of MARTE to declare the model parameters.
Attribute	
contextParams	Set of global variables for the given context.

Continued on next page

TABLE II – continued from previous page

Service	Service provided by the system to the users. A service consists of a set of basic services provided/required by the system components. A service is fulfilled through a sequence of steps.
Attributes	
execProb	Service execution probability [11], [16], [19].
/ssAvail	Steady state availability (percentage) [4], [5], [9], [26]. The steady state availability can be defined as: MTTF/MTBF or MTTF/(MTTF+MTTR) or MTTF/(MTTF+recoveryDuration) [33]. It can be derived then from associations connecting Service with Failure, Repair (or Recovery) classes, and from the homonym attributes defined in the latter.
instAvail	probability that the provided service at time t is correct (time dependent) [33]. Used in [9], [26].
unreliability	Unreliability, that is the probability that the time to failure random variable is less or equal than time t (time dependent) [33].
/reliability	Reliability [9], [26], that is the probability that the time to failure random variable is greater than time t (time dependent) or, in other words, that the service provided to the user is correct during the time interval $(0, t]$ [33]. It is defined as 1-unreliability. It is a *survival function* [15].
missionTime	Time interval in which the service unreliability is lower than a preassigned threshold. It is the inverse function of unreliability.
availLevel	Availability level associated to the nines of availability. E.g., high corresponds to 99% of ssAvail, etc.
reliabLevel	Reliability level.
safetyLevel	Safety level.
complexity	Complexity metric [4], [5], [16], [21], [22]. There are many complexity metrics in the literature (e.g., Halstead's Software Metric, McCabe's Cyclomatic Complexity). This attribute provides a quantitative characterization of the service complexity which is related to the service failure proneness.
ServiceRequest	The user that requests one or more services to the system.
Attributes	
accessProb	Probability that the user accesses to the system [11].
serviceProb	Probability that the user, once accessed to the system, requires a certain service [11]. It is a vector of real values ordered according to the list of services *requests* requested by the user.
Step	Step of a system component that is necessary to carry out a service.

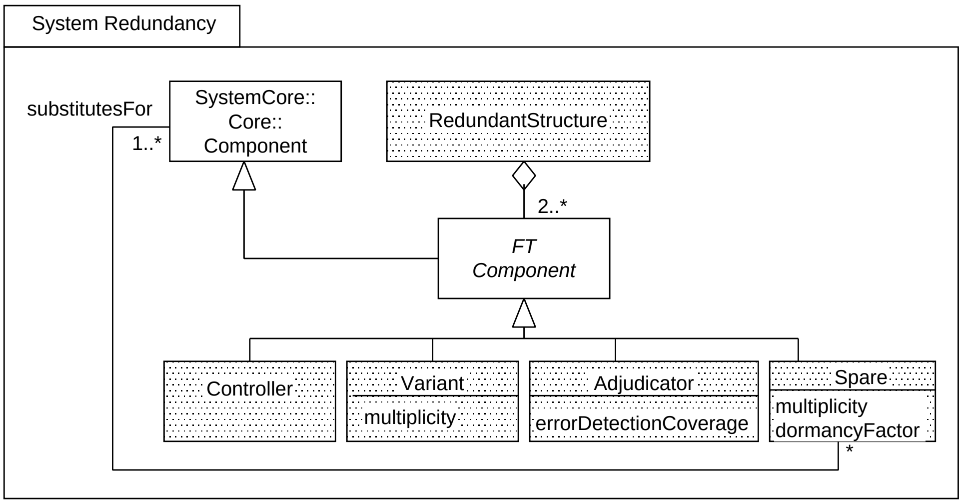

Fig. 4. System redundancy model

The System Redundancy model of the System Core Package (Figure 4) represents the redundancy structures that may characterize a system. Software and hardware redundancy are the typical means used to add fault tolerance capabilities to software systems (by eliminating single points of failure). In dependability analysis of fault tolerant systems is important to identify and evaluate the system under multiple dependent failure assumption (i.e., common mode failures).

TABLE III: System redundancy model description.

Adjudicator, Controller, Variant *(Adjudicator) Attribute* errorDetecCoverage *(Variant) Attribute* multiplicity	The three terms come from the software FT concept of "recovery block" [25]. These concepts are used in [9], [26]. It is the error detection coverage associated to the adjudicators (e.g., a tester that checks the results produced by the variants). Used in [9], [26]. Number of variant copies (required or assumed)
FT Component	Abstract concept that indicates a component belonging to a redundant structure.
Spare *Attribute*	This term is mainly referred to HW components, however there is not a common agreement (i.e., can be also used for SW components). A spare may substitute for one or more components. This concept is used in [30].
	Continued on next page

TABLE III – continued from previous page

dormancyFactor	Ratio between failure rate in standby mode and failure rate in operational mode. Its value is also used to discriminate the type of spare (i.e., hot, cold, warm). Used in [30].
multiplicity	Number of spare copies (required or assumed)
Redundant Structure	It is a container of FT components (at least two). This concept has been introduced to specify impairments that may affect (simultaneously) a set of components belonging to the redundant structure.

The Threats model of the Threats package (Figure 5) represents the threats that may affect the system that is faults, errors and failures [3], and hazards [23]. The model represents also the cause-effect relationships between the threats: the fault is the original cause of errors and affects system components. A fault generator concept is added to represent a mechanism, used for example in Petri net dependability models, to inject faults in the system as well as to specify the max number of concurrent faults to be tolerated by the system or to be assumed in the system for analysis purposes. Errors are related to component anomalous states and can be propagated among component states. In particular, when an error affects an external state of a component (that is, affects the service interface of that component), then the propagation may occur between system components, via their connectors. Finally, errors may cause impairments, that is failures/hazards at different system level: 1) at service step level, when the service provided by the component becomes not correct, then leading to failure/hazard steps; 2) at component/connector level, when the component/connector is not able to provide any basic service; 3) at system level, when the failure is perceived by the system users. The abstract concept of *impairment* has been introduced: it may refer to either failure or hazard, depending of the dependability analysis domain (i.e., reliability/availability or safety). Observe that also redundancy structures can be affected by impairments: in this case, the impairment affects all the FT component belonging to the redundant structure.

TABLE IV: Threats model description.

Error	Error descriptor.
Attributes	
latency	The time elapsed between the error occurrence and the error detection [4], [5].
probability	Probability of an error occurrence [4], [5].
Error Propagation	Error propagation relation between interacting components (represents the concept of external error propagation). It is characterized by a direction [4], [5], [9], [26].
Attribute	

Continued on next page

TABLE IV – continued from previous page

probability	Error propagation probability [4], [5], [9], [26].
Error Propagation Relation *Attribute* propagationExpr	The relation is defined over the set of error propagation relations to express non trivial relationships among the latter, including sequence dependencies ("order" constraint attached to the aggregation relation). It is a logical expression that models non trivial error propagation relationships. It is introduced to support the approach [30] based on the derivation of dynamic fault trees.
Error Step	An erroneous state/action.
Failure *Attributes* occurrenceRate MTTF MTBF occurrenceDist domain detectability consistency consequence condition	Failure descriptor. Failure occurrence rate, i.e., number of failures per unit time. Used as either input parameter [14], [17], [18], [30] or requirement [4], [5]. Mean Time To Failure. It can be either a requirement, or metric or input parameter (see comments for "occurrenceDist") [4], [5], [9], [26]. Mean Time Between Failures. It can be either a requirement, or metric or input parameter (see comments for "occurrenceDist") [4], [5]. Failure occurrence distribution (time dependent). Depending on the affected system level, it can be either a requirement, or metric (e.g., top-level service unreliability [9], [26]) or an input parameter (e.g., component failure assumption [14], [17], [18], [30]). Failure domain [3], i.e., content, early timing, late timing, halt or erratic. Used in [4]–[6], [21], [22], [31]. Failure detectability [3], i.e., signaled or unsignaled. Failure consistency [3], i.e., consistent or inconsistent. Failure consequence [3], i.e., minor, marginal, major, or catastrophic. Used in [4], [5], [16], [19]. Logical condition that leads to the failure. Used to express relationships among component failure states [1], [2], [8].
Failure Step	It has various meanings, that is: state/activity affected by failure [17], [18], state reached after failure occurrence [6], [9], [26], failure event/transition/call that leads to a failure state [9], [14], [26], [30].
Fault *Attributes* occurrenceRate	Fault descriptor. Fault occurrence rate, e.g. number of faults per year [6], [9], [26].

Continued on next page

TABLE IV – continued from previous page

latency	Fault latency is the time elapsing between a fault occurrence and the instant in which it is perceived by the component(s) [6].
occurrenceProb	Probability of a fault occurrence (time independent) [11].
occurrenceDist	Probability of a fault occurrence within time t (time dependent). E.g., it can be specified as negative exponential distribution with input parameter the occurrenceRate attribute.
persistency	Indicates the type of fault w.r.t. persistency [3], that is either transient or permanent [6], [31].
duration	Fault duration (from its occurrence). This attribute can be used to discriminate the fault persistency [4], [5].
Fault Generator	Fault injector. This concept can be modeled, for example, by a UML state machine that represents the behavior of the injected fault(s) [6], [31].
Attribute	
numberOfFaults	Minimum number of faults (with the characterization given by the fault association end) to be tolerated by the system or maximum number of faults that affect simultaneously the system (for analysis purposes) [31].
Hazard	Hazard descriptor [23].
Attribute	
origin	Depending on the factors that provoked it, it can be classified as endogenous (due to factors inherent in the system) or exogenous (due to external phenomena).
severity	Worst possible accident that could result from the hazard given the environment in its most unfavorable state.
likelihood	Likelihood of hazard occurring (qualitative).
/level	Derived attribute: it is a combination of severity and likelihood.
latency	Duration from its occurrence to an accident.
accidentLikelihood	Likelihood of hazard leading to an accident.
guideword	Guideword that describes the hazard [19], e.g., applied in FFA.
accident	Accident on the system environment that may be provoked by the hazard.
Hazard Step	Similar meaning of failure step but referred to hazard.
Impairment	Abstract concept that may correspond concretely to either failure or hazard.
Attribute	
occurrenceProb	Occurrence probability of the impairment (time independent). When applied to failures, it has been used as either requirement [4], [5], or as metric [19].

Continued on next page

TABLE IV – continued from previous page

/risk	Risk factor [16]. It is a derived attribute, when applied to failure is a combination of (failure) occurrence probability and of failure consequence. When applied to hazards, is a combination of (hazard) latency and accident-Likelihood.
cost	Cost of the impairment (accepted measure of consequences) [19].

The Maintenance model of the Maintenance package (Figure 6) concerns repairable systems and represents the maintenance actions undertaken to restore the system affected by threats. According to [3], we distinguish repair actions, that involve the participation of external agents (e.g., repairman, test equipment, etc) and recovery actions, usually carried out in fault tolerant systems, that aim at transforming the system anomalous states into correct states. This package includes concepts that are necessary to support the evaluation of system availability.

TABLE V: Maintenance model description.

External Agent	repairman, test equipment, remote reloading software, etc. that undertakes repair actions on system component affected by threats. This class represents an external actor (out of the system border).
Maintenance Action	Abstract concept that includes both repair and recovery actions.
Attribute	
rate	Rate of the maintenance action.
distribution	Probability distribution associated to the maintenance action, that is time to repair/recover (time dependent) [6].
Reallocation Step	Reconfiguration step in which an ordered set of sw components are reallocated onto an ordered collection of hw spare components [30] (to model the mapping, the set and the collection should have the same size).
Reconfiguration Step	Abstract concept that represents a step in which a reconfiguration technique is carried out. The latter consists of either switching in spare components or reallocating sw components among non failed hw components [3].
Recovery	Recovery descriptor. A recovery activity/action is usually carried out by the system itself as a part of an implemented fault tolerance strategy [3].
Attribute	
duration	recovery duration.
coverageFactor	probability of recovery given that a fault is occurred in the system.

Continued on next page

TABLE V – continued from previous page

Repair	Repair descriptor. A repair activity/action is carried out on system components by external agents.
Attribute	
MTTR	Mean Time To Repair [4]–[6], [9], [26].
Replacement Step	Reconfiguration step in which an ordered set of failed components are replaced with an ordered set of spare components [30] (to model the mapping, the two sets should have the same size).

A. DA model assessment

The assessment aims at verifying whether the concepts proposed in the literature (limited to the proposals we considered) are represented in the DA domain model.

- ✓ Pataricza [31]: *QoSvalues* added to critical resources and steps to support qualitative analysis. Our DAM profile aims at supporting quantitative evaluation of dependability (that is dependability metric estimation), so the above concept is considered not relevant for DAM purposes.
- ✓ Addouche et al. [1], [2]: the *reliability* and *maintainability* QoS characteristics have been defined as attributes of *Service* (*reliability*), *Recovery* (*recoveryDist*) and *Repair* (*repairDist*) classes. QoS characteristics related to activity durations/deadlines have not been considered since they are specifically related to timing constraints (the MARTE schedulability sub-profile can be used for this purpose). The concept related to *Indicator* and *Cause* classes, used to represent conditional component failures, are captured by the *condition* attribute of the *Failure* class.
- ✓ Bernardi et al. [4], [5]: all relevant concepts included.
- ✓ Bernardi-Merseguer [6]: a fault step is missing. Fault step refers to the behavior of the fault generator (outside the system border). Maybe is not necesary to include it (to avoid confusions).
- ✓ Bondavalli et al. [9], [26]: *normal (response)* step, that is an event representing a normal response of an object toward the client. Not included explicitly, can be considered as a state (step) in the domain model (see Core model).
- ✓ Dal Cin [13]: the reliability and availability requirements can be captured by the correspondent attributes of *Service* and *Component* classes. We do not include instead concepts to support the specification of dependability mechanisms and fault tolerant architectures (this is addressed by the QoS&FT profile).
- ✓ D'Ambrogio et al. [14]: all the concepts included.
- ✓ Pay-Dugan [30]: 1) *hot,warm,cold* to discriminate spare components. We use the *dormancyFactor* attribute to discriminate them, as suggested, also, by Pay-Dugan. 2) Error propagation relation from hw to software components. 3) error propagation relation between components related with a "use-service-of" relation. The latter two are not explicity represented, they can be seen as a refinement of the external error propagation relation (see Threat Chain model).

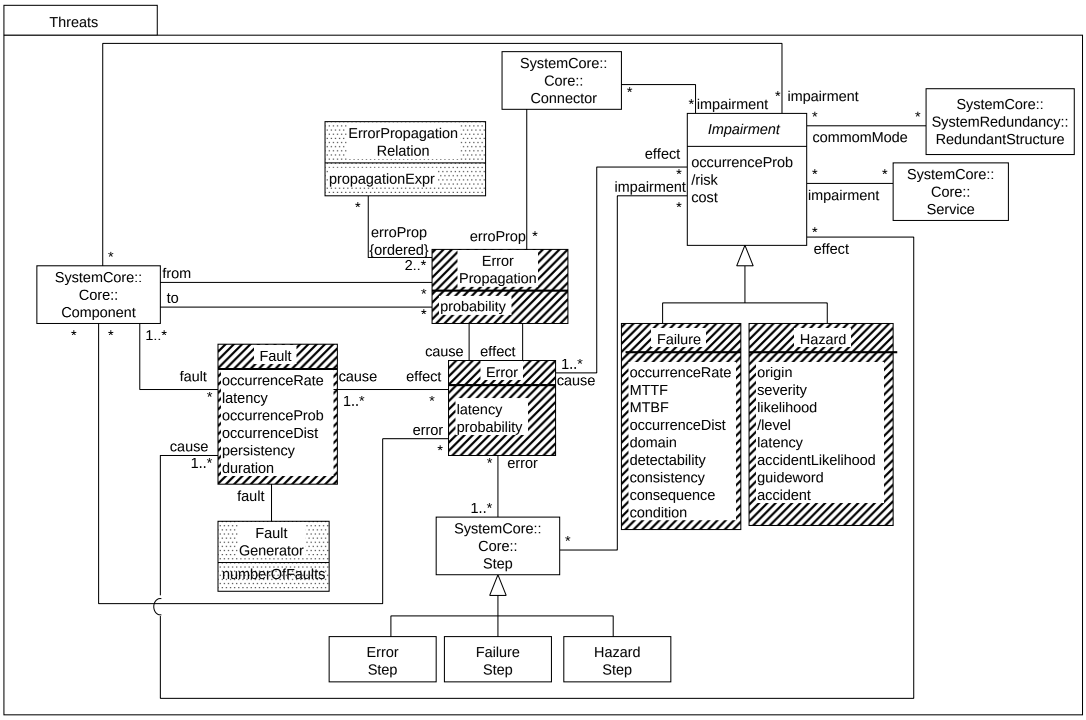

Fig. 5. Threats model

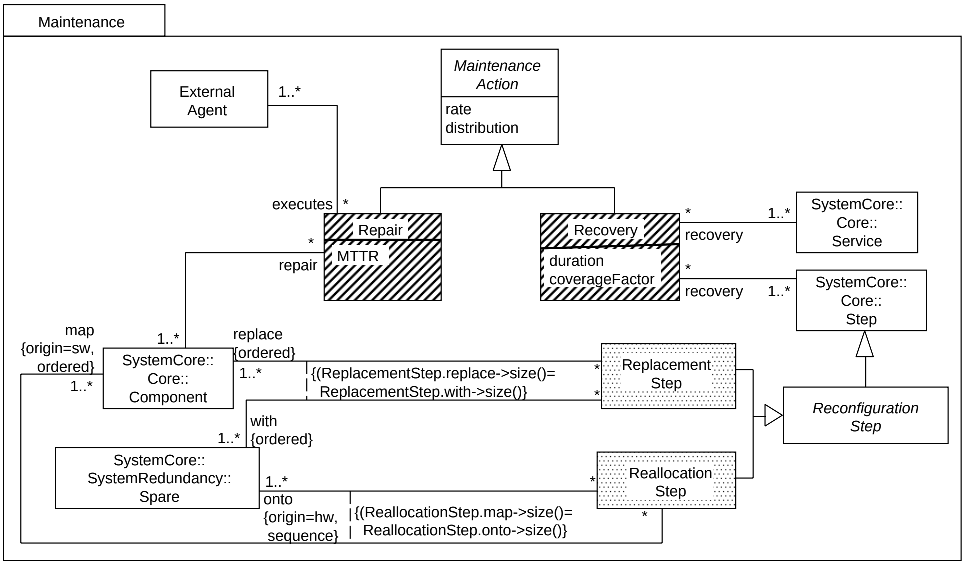

Fig. 6. Maintenance model

- ✓ Cortellessa-Pompei [11]: 1) *REbp*, that is the number of invocation associated to components, and *REnummsg*, that is number of invocation of a connector. They are strictly related to the method of derivation of the target dependability model, and, actually, they can be calculated using the information in the UML diagrams (without profile extensions). 2) *REindexHost* associated to Execution Host to represent the list of host names physically connected to the current one. As before. The *REconnector* concept has a correspondence with the concept of *FailureStep* in the domain model.
- ✓ Grassi et al. [17], [18]: all the concepts included.
- ✓ Jurjens [21], [22]: the type of failures considered are covered by the concepts associated to the *Failure* class. The redundancy concepts introduced by Jurjens are misleading w.r.t. the terminology included by Laprie et al. Indeed, they indicate different fault masking policies (e.g., replication with majority voting) rather than redundancy characteristics. In the domain model spatial redundancy is represented.
- ✓ Pataricza et al. [32]: faulty behavior and error propagation concepts are addressed by the *FailureStep* and *ErrorPropagation* classes.
- ✓ Goseva et al. [16]: all the concepts included. In particular, the failure severity and risk concepts are captured by the *consequence* and *risk* attributes of *Failure* class. Complexity and coupling concepts have been associated to *Component*/*Service* and *Connector* classes, respectively.
- ✓ Hassan et al. [19]: failure modes are captured by several *Failure* attributes. The concept of failure/hazard cost has been addressed by the *cost* attribute defined in the *Failure* and *Hazard* classes. Hazard guidewords are captured by the *guideword* attribute of *Hazard* class. Finally, the scenario execution probability is represented by the *execProb* attribute of *Service* class.

IV. DEPENDABILITY ANALYSIS MODELING (DAM) PROFILE

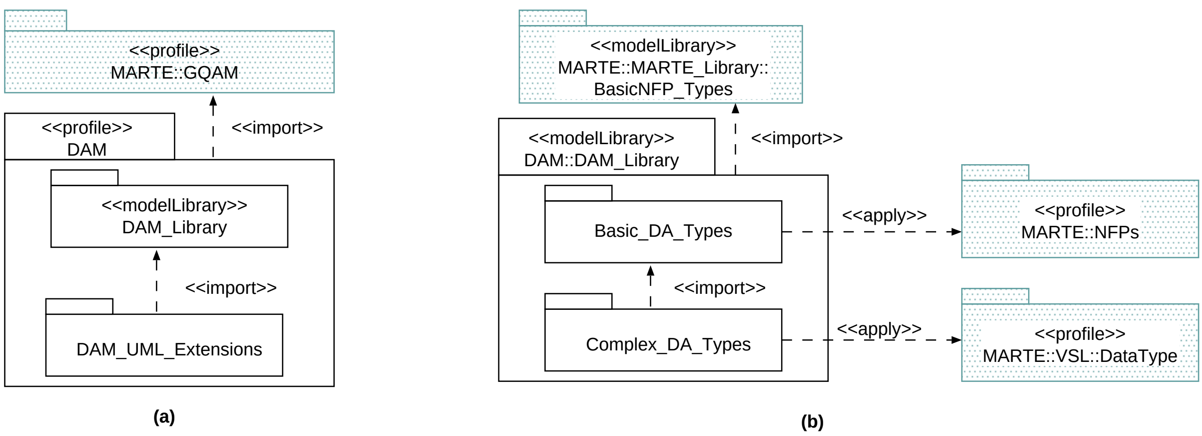

Fig. 7. DAM profile overview

The DAM Profile, in Figure 7(a), includes a set of UML extensions, that is stereotypes, their attributes and constraints, together with a model library that provides the necessary dependability data-types for the definition of stereotype attributes. The DAM Profile specializes UML extensions of the General Quantitative Analysis

Modeling profile of MARTE. The UML extensions are defined in the following, considering the DA domain model of the previous section and are described using a tabular format.

A. DAM UML extensions

The domain classes that are mapped into stereotypes are depicted as dotted classes in Figures 3,4,5 and 6.

The stereotype names are prefixed by *Da*, namely Dependability analysis, and for each stereotype (first column of Table VI):

- An explicit reference is given to the domain class represented by the stereotype. Then, the semantics associated to the stereotype is the one of the mapped domain class.
- A stereotype may extend UML(v.2) meta-classes or specialize a MARTE stereotype.
- A stereotype can be generalized by another stereotype, that is it inherits all the properties of the super-stereotype. In particular, the generalization relationship (direct and indirect) between stereotypes maps a generalization relationship between the corresponding mapped domain classes of the domain model.
- A stereotype attribute can map either an attribute of the (mapped) domain class, in this case we maintain the same attribute name (and the same semantics), or an association end of a domain association between the (mapped) domain class and another domain class. In the latter case, the name (and the semantics) of the stereotype attribute is the name (and the semantics) of the association end. The stereotype attributes are characterized by attribute types. For the stereotype attributes, that map domain class attributes, we use primitive types, like *Boolean*, and the *basic NFP types* of MARTE library when possible (they are prefixed as *NFP_*). New basic dependability types are also used and they will be described in IV-B2. For the stereotypes attributes that map association ends, we define new complex dependability types (that will be described in sub-section IV-B1).
- Constraints can be assigned to stereotypes and represent constraints for the use of the profile by the end-users (e.g., software analysts that use the profile to annotate their UML models).

TABLE VI: Stereotypes description.

DaComponent	maps the SystemCore::Core::Component domain class	
	Extensions	
	Generalization	MARTE::GRM::Resource
	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[*]

Continued on next page

TABLE VI – continued from previous page

	reliability	NFP_CommonType[*]
	missionTime	NFP_CommonType[*]
	availLevel	DaLevel[*] - Application specific
	reliabLevel	DaLevel[*] - Application specific
	safetyLevel	DaLevel[*] - Application specific
	complexity	NFP_Real[*]
	fault	DaFault[*] - Faults affecting the component
	error	DaError[*] - Error affecting the component
	failure	DaFailure[*] - Failures affecting the component
	hazard	DaHazard[*] - Hazards affecting the component
	repair	DaRepair[*] - Repairs undergone by the component
DaConnector	maps the SystemCore::Core::Connector domain class	
	Extensions	Association, CommunicationPath, Deployment, Connector, InvocationAction, Dependency (e.g., Usage), Message, Extend, Include
	Generalization	none
	Attributes	
	coupling	NFP_Real[*]
	errorProp	DaErrorPropagation[*] - Error propagations carried by the connector
	failure	DaHazard[*] - Failures affecting the connector
	hazard	DaHazard[*] - Hazards affecting the connector
DaService	maps the SystemCore::Core::Service domain class	
	Generalization	MARTE::GQAM::GaScenario
	Attributes	
	execProb	NFP_Real[*]
	ssAvail	NFP_Percentage[*]
	instAvail	NFP_CommonType[*]
	unreliability	NFP_CommonType[*]
	reliability	NFP_CommonType[*]
	missionTime	NFP_CommonType[*]
	availLevel	DaLevel[*] - Application specific
	reliabLevel	DaLevel[*] - Application specific
	safetyLevel	DaLevel[*] - Application specific
	complexity	NFP_Real[*]
	failure	DaFailure[*] - Failures affecting the service

Continued on next page

TABLE VI – continued from previous page

	hazard	DaHazard[*] - Hazards affecting the service
	recovery	DaRecovery[*] - Recovery actions undertaken on the service
DaServiceRequest	maps the SystemCore::Core::ServiceRequest domain class	
	Extensions	Classifier (e.g. Actor), Lifeline, Interaction, Instance-Specification
	Generalization	none
	Attributes	
	accessProb	NFP_Real[*]
	serviceProb	NFP_Real[*]{ordered}
	requests	DaService[*]{ordered}
	Constraints	the order of *serviceProb* corresponds to the order of *requests*.
DaStep	maps the SystemCore::Core::Step domain class	
	Generalization	MARTE::GQAM::GaStep
	Attributes	
	kind	StepKind - Enumeration indicating the type of step
	error	DaError[*] - Errors affected by the step
	failure	DaFailure[*] - Failures affected by the step
	hazard	DaHazard[*] - Hazards affected by the step
	recovery	DaRecovery[*] - Recovery actions undertaken in the step
DaAdjudicator	maps the SystemCore::SystemRedundancy::Adjudicator domain class	
	Extensions	none
	Generalization	DaComponent
	Attributes	
	errorDetecCoverage	NFP_Percentage[*]
	Constraints	origin = sw
DaRedundant Structure	maps the SystemCore::SystemRedundancy::RedundantStructure domain class	
	Extensions	Package
	Generalization	none
	Attributes	
	commonModeFailure	DaFailure[*]
	commonModeHazard	DaHazard[*]

Continued on next page

TABLE VI – continued from previous page

	Constraints	It is a property related to two or more FT components (controllers, variants, adjudicators,spares). The stereotyped model-element is a package containing the redundant components affected by the common mode failure/hazard.
DaController	maps the SystemCore::SystemRedundancy::Controller domain class	
	Extensions	none
	Generalization	DaComponent
	Attributes	none
DaSpare	maps the SystemCore::SystemRedundancy::Spare domain class	
	Extensions	none
	Generalization	DaComponent
	Attributes	
	multiplicity	NFP_Integer[*]
	dormancyFactor	NFP_Real[*]
	substitutesFor	String[1..*] - Component names (to be) substituted by the spare
	Constraints	The components to be substituted must be DaComponents.
DaVariant	maps the SystemCore::SystemRedundancy::Variant domain class	
	Extensions	none
	Generalization	DaComponent
	Attribute	
	multiplicity	NFP_Integer[*]
DaErrorProp Relation	maps the Threats::ErrorPropagationRelation domain class	
	Extensions	Constraint
	Generalization	none
	Attributes	
	propagationExpr	PropExpression - logical expression with *errorProp* terms
	errorProp	DaErrorPropagation[2..*]{ordered} - Error propagation terms
	Constraints	the error propagation terms are ordered to support the specification of sequence dependencies.
DaFaultGenerator	maps the Threats::FaultGenerator domain class	

Continued on next page

TABLE VI – continued from previous page

	Generalization	MARTE::GQAM::GaWorkloadGenerator
	Attributes	
	numberOfFaults	NFP_Integer[*] = 1 - Number of faults to be tolerated or number of system components concurrently affected. Default is one. Redefine the GaWorkloadGenerator concept *pop*.
	fault	DaFault - Characterization of the generated faults
	Constraints	multiple faults (that is when *numberOfFaults* is greater than one) have the same characterization given by the complex value associated to *fault* attribute.
DaReplacementStep	maps the Maintenance::ReplacementStep domain class	
	Extensions	none
	Generalization	DaStep
	Attributes	
	replace	String[1..*]{ordered} - Failed component to be replaced
	with	String[1..*]{ordered} - Component that replaces the failed one
	Constraints	1) the failed components to be replaced is are DaComponent 2) the components that replace the failed ones are a DaSpare, 3) the order of *replace* corresponds to the order of *with*.
DaReallocationStep	maps the Maintenance::ReallocationStep domain class	
	Extensions	none
	Generalization	DaStep
	Attributes	
	map	String[1..*]{ordered} - The component to be reallocated
	onto	String[1..*]{ordered} - The spare component that hosts the reallocated component
	Constraints	1) The reallocated components are DaComponent, with origin=sw, 2) the components that host the reallocated ones are DaSpare, with origin=hw, 3) the order of *map* corresponds to the order of *onto*.

The DAM profile provides support to the specification of non trivial threat assumptions. In particular, the (sequence) dependencies of error propagation between components, state-based failure conditions and common mode failures of a set of redundant components.

a) Error propagation dependencies: The *DaErrorPropagation* stereotype that can be used to specify, via constraints, non trivial relationships on a set of error propagations between system components. This is achieved by assigning proper values to the attributes *propagationExpr* (*PropExpression* type) and *errorProp* (*DaErrorPropagation* type). The value of the former is a logical expression on a set of error propagation terms (variables) while the value of the latter represents the (ordered) set of error propagation terms, at least two (the variable declaration and initialization). The syntax used for the specification of error propagation dependencies is given in Table VII.

`propagationExpr-value ::=`	`term logical-op term | '(' propagationExpr-value ')' logical-op term`
	`| 'not' '(' term ')' | 'not' '(' propagationExpr-value ')'`
	`| 'ordered' '(' setOfTerms ')'`
`logical-op ::=`	`'and' | 'or' | 'xor' | 'implies'`
`setOfterms ::=`	`term ',' term | setOfterms ',' term`
`term ::=`	`'$' variable-name`
`errorProp-value ::=`	`'(' errorProp-body ')'`
`errorProp-body ::=`	`term '=' errorProp-term [';' errorProp-body ]`
`errorProp-term ::=`	`'(' 'probability' '=' prob ',' 'from' '=' component-source ','`
	`'to' '=' component-target ')'`
`prob ::=`	`NFP_Real`
`component-source ::=`	`string`
`component-target ::=`	`string`

TABLE VII

BNF SYNTAX FOR THE SPECIFICATION OF ERROR PROPAGATION DEPENDENCIES

b) State-based failure conditions: State-based failure conditions can be specified for either components or services, by assigning a value to the *condition* attribute (*FailureExpression* type) of the complex NFP *failure* (*DaFailure* type). The value is actually a logical expression on a set of state-failure terms. The syntax used to specify a state-based failure conditions is given in Table VIII.

`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 VIII

BNF SYNTAX FOR THE SPECIFICATION OF STATE-BASED FAILURE CONDITIONS

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* of *failed*. Then, we can stereotype both the components as *DaComponent* and annotate the following property on component *A*:

```
failure = (condition =
                (component = B, state=degraded) or (component=B, state=failed))
```

c) Common mode failures/hazards: The stereotype *DaRedundantStructure* is used to characterize the impairments affecting simultaneously the set of FT components belonging to a redundant structure (that is components stereotyped as either *variant*, or *controller*, or *adjudicator* or *spare*). For example, the common mode failure probability. The annotation at model specification level is carried out by including the set of FT components into a package stereotyped as *DaRedundantStructure* and then specifying the value of the attribute *commonModeFailure* as a package property.

B. DAM model library

The DAM library contains complex and basic dependability types as depicted in Figure 7(b). We use MARTE profile, in particular:

- Basic NFPs types from MARTE library are imported in order to reuse them (both in the definition of complex and basic dependability types)
- The MARTE sub-profile *NFPs* is applied to the definition of new basic dependability types.
- The MARTE sub-profile *VSL* is applied to the definition of complex dependability types.

1) Complex dependability types: Complex dependability types are MARTE *tupleTypes* characterized by basic NFPs, from MARTE library, and/or basic dependability types. They map the domain classes depicted with diagonal stripes in Figures 5 and 6. As for stereotypes, a complex dependability type is prefixed by *Da* and an attribute of a complex dependability type (e.g., *DaErrorPropagation*) can map either an attribute of the (mapped) domain class (e.g., *probability*) or an association end of a domain association between the mapped domain class and another domain class (e.g., *from*). In both cases we use the same names (and semantics) of the mapped domain elements.

TABLE IX: Complex dependability types description.

DaError	maps the Threats::Error domain class	
	Attribute	
	latency	NFP_Duration[*]
	probability	NFP_Real[*]
DaError Propagation	maps the Threats::ExternalErrorPropagation domain class	
	Attribute	
	probability	NFP_Real[*]
	from	String[0..1] - name of the source of error propagation
	to	String[0..1] - name of the destination of error propagation

Continued on next page

TABLE IX – continued from previous page

	Constraints	the names of the source and destination are names of *DaComponent* elements connected by a *DaConnector*.
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[*]
	condition	FailureExpression[0..1] - logical expression with *failstate* terms, i.e., *failstate* = (*component* = String, *state* = String), where the component string is the name of a *DaComponent* and the state string is a state of the *DaComponent*.
DaFault	maps the Threats::Fault domain class	
	Attribute	
	occurrenceRate	DaFrequency[*]
	latency	NFP_Duration[*]
	occurrenceProb	NFP_Real[*]
	occurrenceDist	NFP_CommonType[*]
	persistency	Persistency[0..1]
	duration	NFP_Duration[*]
DaHazard	maps the Threats::Hazard domain class	
	Attribute	
	origin	FactorOrigin[0..1]
	severity	DaCriticalLevel[*]
	occurrenceProb	NFP_Real[*]
	likelihood	DaLikelihood[*]
	level	NFP_Real[*]
	latency	NFP_Duration[*]
	accidentLikelihood	DaLikelihood[*]

Continued on next page

TABLE IX – continued from previous page

	risk	NFP_Real[*]
	cost	DaCurrency[*]
	guideword	Guideword[*]
	accident	String[*]
DaRecovery	maps the Maintenance::Recovery domain class	
	Attribute	
	rate	DaFrequency[*]
	duration	NFP_Duration[*]
	distribution	NFP_CommonType[*]
	coverageFactor	NFP_Real[*]
DaRepair	maps the Maintenance::Repair domain class	
	Attribute	
	rate	DaFrequency[*]
	MTTR	NFP_Duration[*]
	distribution	NFP_CommonType[*]

2) Basic dependability types: Basic dependability types, represented in Figure 8, can be either simple enumeration types (*Origin, Detectability, Persistency, CriticalLevel, Likelihood, Level, Consistency, Domain, FactorOrigin, Guideword, StepKind, DaCurrencyUnitKind* and *DaFrequencyUnitKind*) or data-types.

In particular, the latter include new NFP types obtained by specializing the *NFP_CommonType* and *NFP_Real* concepts of MARTE library (*DaFrequency, DaCurrency, DaLevel, DaCriticalityLevel*). The imported NFP types are shown in grey in the Figure. Such new types inherit from super-types several properties, that are:

- *expr*: expressions in MARTE Value Specification Language (VSL),
- *source*: origin of the specification, such as *estimated* (e.g., a metric to be estimated) and *required* (e.g., a requirement to be satisfied),
- *statQ*: type of statistical measure (e.g., maximum, minimum, mean),
- *dir*: type of the quality order relation in the allowed value domain of the NFP, for comparative analysis purposes.

C. DAM Profile assessment

The assessment aims at verifying whether the concepts listed in the check list of *information requirements* for a dependability profile (reported below) are represented in the DAM profile.

✓ *(IR1) Identification of the dependability analysis context, in particular the types of non functional requirements to be assessed, i.e., reliability (R), availability (A), safety (S).*

The dependability analysis context has been identified via the set of introduced stereotypes. There is not an explicit representation of R/S/A analysis contexts (maybe is not necessary, since during evaluation both reliability

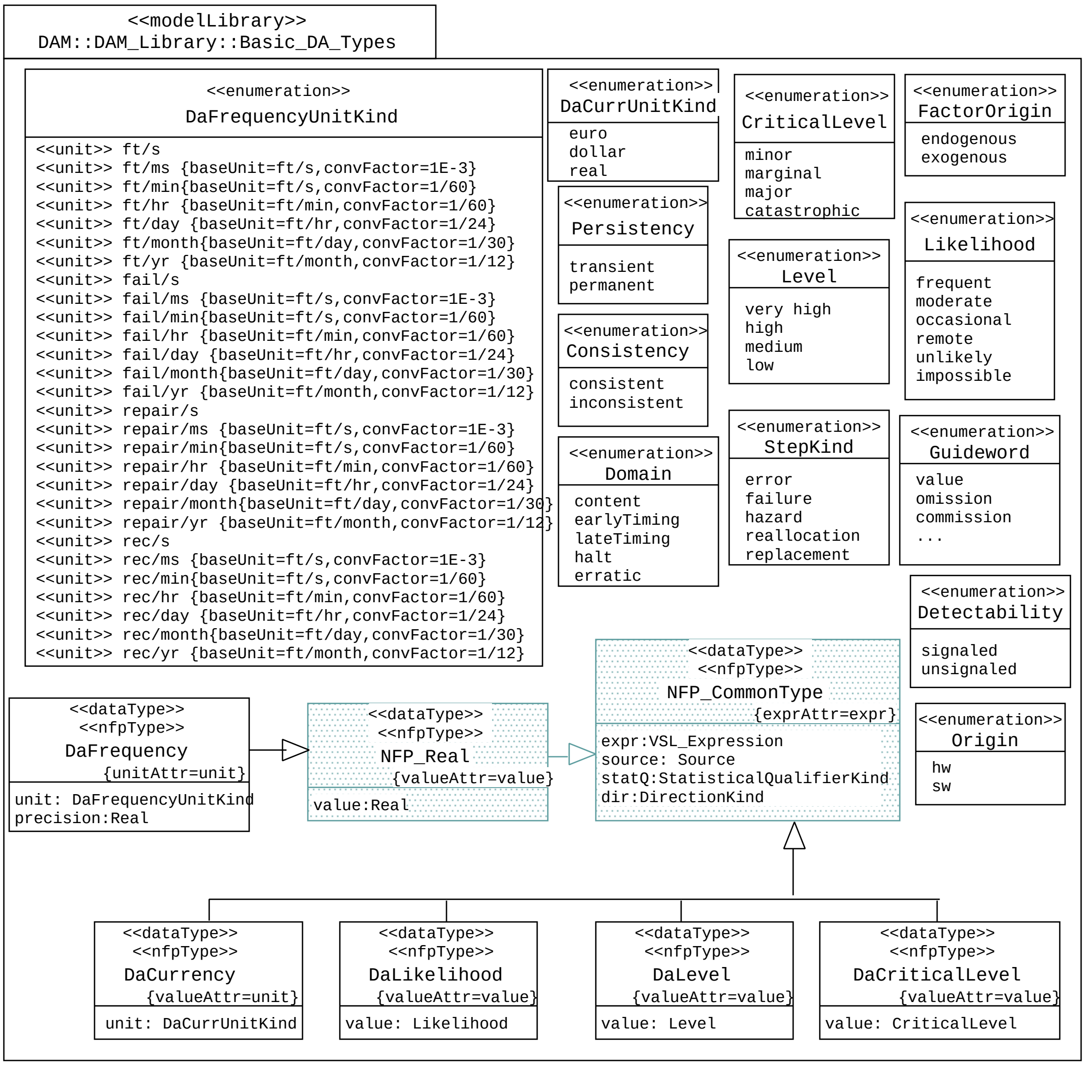

Fig. 8. DAM basic types

and availability metrics could be calculated using the same context).

- ✓ *(IR2) Specification of dependability requirements in terms of upper/lower bounds, like the maximum (minimum) system or component unreliability (reliability) required, the minimum availability or safety level required.*

The characteristics of interest are defined as attributes of NFP or dependability basic types (see e.g., attributes associated to *DaComponent* and *DaService*), so they can represent requirements - by setting the *source* property to the *required* value, during the model annotation - and they can be expressed in terms of upper/lower bounds - either via *expr* property or via *statQ* properties. For example, considering a *DaService* we can define the following requirements[1]:

```
1) reliability = (expr= R(50,hrs) >= 0.9, source=req)
```

[1]We use the extended VSL notation, in the examples

```
2) unreliability = (expr= U($t) < 0.1, source=req)
3) ssAvail = (value=99%, statQ=min, source=req)
3) safetyLevel = (value=high, statQ=min, source=req)
```

that is: 1) The service reliability, during 50 hours of provisioning, should be at least 0.9. 2) The service unreliability, during t hours of provisioning, should be less than 0.1. 3) The minimum service steady state availability should be 99%. 4) The minimum service safety level should be *high*.

✓ *(IR3) Specification of dependability measures to be estimated during the analysis. The set of supported measures should include, at least, the reliability/unreliability PdFs, the system failure probability, MTTF, the maintainability PdF, MTTR, the instantaneous and the steady state availability, the safe mission time, the safety risk factor.*

The dependability metrics are defined as attributes of NFP or basic dependability types, so it is sufficient to set the *source* property to *estimated* value, during the model annotation to represent a metric to be evaluated during the analysis. For example, considering a *DaService* we can define the following metrics representing the required set:

```
1) reliability = (expr= R(50,hrs), source=est)
2) unreliability = (expr= U($t), source=est)
3) failure = (occurrenceProb= (value = $FP, source=est))
4) failure = (MTTF = (value = $mttf, source=est))
5) repair = (repairDist = (expr= M($t), source=est))
6) repair = (MTTR = (value = $mttr, source=est))
7) instAvail = (expr = A($t), source=est)
8) ssAvail = (value=$avail,  source=est)
9) missionTime = (expr=MT(0.1),  source=est)
10) failure = (risk= (value=$risk, source = est))
```

✓ *(IR4) Specification of the dependability input parameters that are needed by the standard techniques for the quantitative evaluation of the system dependability. The dependability input parameters characterize, from a quantitative point of view:*

- *(IP1) The processes leading to service failures and accidents. In particular, the threats of dependability that may affect both hardware and software resources (e.g., the probability of fault occurrence, the error latency, the probability of failure, the hazard severity).*
- *(IP2) The repair/reconfiguration processes, in case of repairable systems/components, that remove basic or derived failures from the system (e.g., repair and restoration rates).*

The dependability threats have been represented as complex dependability types: *DaFailure,DaError,DaFault, DaHazard* (IP1). Maintenance concepts are represented by complex dependability types: *DaRepair* and *DaRecovery* (IP2). Each type consists of a set of attributes (of NFP or basic dependability type) that can be used to give a quantitative characterization of faults, errors, failures, hazards and of repair/recovery. For example, considering a *DaComponent* we can define the following input parameters:

```
1) fault = (occurrenceProb= (value = 0.4))
2) repair = (repairRate = (value = (5,repair/day)))
```

that is 1) a fault has an occurrence probability of 0.4, and 3) the repair rate is of 5 repairs per day. Considering a *DaService* we can define the following input parameter:

```
recovery = (recoveryRate = (value = (5,rec/min) ))
```

Error latency is specified instead using the *DaStep* stereotype:

```
error = (latency= (value = (10,ms) ))
```

Failure and hazard input parameters can be specified for both *DaComponent* and *DaService*:

```
1) failure =
   (occurrenceDist = (expr= weibull($t,shape=0.9,scale=(5,hrs))))
2) hazard = (severity = (value= minor))
```

that is , 1) the failure probability (time dependent) is given by the Weibull PdF, and 2) the hazard severity level is minor.

✓ *(IR5) Specification of different fault behaviors depending on their timing persistence (i.e., permanent, transient and intermittent) and of fault occurrence assumptions (e.g., single fault assumption).*

DaFault complex dependability type includes the *persistency* attribute, of enumeration type, that is used to discriminate permanent and transient faults. Intermittent fault concepts, has not been mentioned since in [3] intermittent concepts is not considered. Fault occurrence assumption can be specified using *DaFaultGenerator* stereotype, via *numberOfFault* attribute.

✓ *(IR6) Specification of the error propagation between system components that interact with each others.*

Addressed by the *DaConnector* stereotype, via the *errProp* attribute. Non trivial error propagation relationships, such as sequence dependencies, can be specified via the *DaErrorPropRelation* stereotyped note symbols, for example:

```
propagationExpr =  ordered($term1, $term2)

errorProp = ($term_1 = (probability=0.3,from=A,to=B);
             $term_2 = (probability=0.2,from=B,to=C))
```

where the two error propagation terms, specifying the error propagation probability from component *A* to component *B* and from component *B* to component *C*, respectively, are related with the *implies* logical connector.

✓ *(IR7) Specification of the system failure modes with respect to different point of views: the domain, i.e, content and timing failures (***Dom***); the detectability, i.e., signaled or unsignaled failures (***Det***); the consistency, i.e., consistent or unconsistent failures (***Con***); the consequences, i.e., minor, marginal, critical and catastrophic failures (***Cons***); and, when multiple failure are considered, the failure dependency, i.e., independent or dependent failures (***Dep***).*

DaFailure complex dependability type includes the *domain, detectability, consistency, consequence* attributes that can be used for specifying failure modes w.r.t. the firsts four points of view. Concerning failure dependency, the *DaRedundantStructure* stereotype can be used to specify the common mode failures/hazards of a set of FT components by packaging them and stereotyping the package. The conditional (component) failures are specified via the *condition* constraint associated to the *Failure* class. For example, assuming a failure dependency of component *A* with respect to the state of components *B* and *C*, then we can specify the failure probability of *A* given the states of *B* and *C*, by stereotyping the components as *DaComponent* and for *A* specifying:

```
failure =
 (occurrenceProb = (value = 0.3),
  condition= ( (component=B, state=degraded) or (component=C,state=failed))
 )
```

Independent failures are specified via the *DaFailure* complex dependability type.

✓ *(IR8) Specification of the hazards leading to accidents, in terms of their basic components (such as the severity, the likelihood of hazard occurring, the duration, the accident likelihood and the risk).*

Addressed by the *DaHazard* complex dependability type, that can be associated to both system components and services. In particular, it consists of a set of attributes that are used to characterize the hazard (e.g., severity, likelihood of hazard occurring and of hazard leading to an accident, duration and risk).

✓ *(IR9) Specification of (uncorrect) behavior of system components affected by threats as well as the reconfiguration activities that restore the system component states. In particular, identification of erroneous/ failure states, threat events, recovery triggers and actions.*

DaStep stereotype is used to specify erroneous/failed/hazardous states/events/transitions as well as actions/activities affected by faults. *DaReplacement* and *DaReallocation* stereotypes are used to specify reconfiguration states/events/activities, etc.

✓ *(IR10) Specification of redundant hardware and software components. In particular, the number of variant/spare copies existing in a redundant structure, the minimum number of components required in a redundant structure for a dependable system, and the type of spare redundancy.*

DaAdjudicator, DaController, DaVariant and *DaSpare* stereotypes are used to specify redundant components. The first two properties are addressed by the *multiplicity* attribute of *DaVariant* and *DaSpare* stereotypes. The type of spare is specified via the *dormancyFactor* attribute of *DaSpare* stereotype.

APPENDIX A

CONTRIBUTIONS TO THE INFORMATION REQUIREMENTS CHECKLIST

TABLE X: Summary of contributions to IR1-IR6

Approach	IR1	IR2	IR3	IR4-IP1	IR4-IP2	IR5	IR6
Pataricza [31]	Dep.					No UML extensions used. Emphasizes the importance of discriminating permanent and transient faults. It suggests the introduction of fault injector SM models that can be used to specify constraints on fault occurrences.	No UML extensions used. Emphasizes the importance of representing error propagation across the system.
Addouche [1], [2]	Dep.			Resource reliability is specified as $\ll qos \gg$ tagged value. State-conditional component failure probability specified using $\ll Indicator \gg$ and $\ll Cause \gg$ classes.	Resource maintainability is specified as $\ll qos \gg$ tagged value.		

Continued on next page

TABLE X – continued from previous page

Approach	IR1	IR2	IR3	IR4-IP1	IR4-IP2	IR5	IR6
Bernardi [4], [5]	Dep.	Requirements (e.g., MTTF, MTTR, steady state availability) are specified as upper or lower bounds using $\ll Input/Output \gg$ attributes.	Metrics to be estimated (e.g., fault dormancy, error latency) are specified using $\ll Output \gg$ attributes.	Input parameters (e.g., fault duration and occurrence, component criticality and complexity level) are specified using $\ll Input \gg$ attributes.		Fault classification of [3] is represented with a class diagram. Class attributes are used to specify timing characteristics of faults according to their persistency.	Error propagation between system components is represented in the FEF chain Class Diagram.
Bernardi-Merseguer [6]	Dep.	Requirements specification (e.g., max time to detect a failure) is compliant to the SPT profile.	Metrics specification (e.g., time to detect a failure) is compliant to the SPT profile.	Fault occurrence and latency are specified for $\ll FTstep \gg$ transitions and do-activities.		SM are used to model different types of faults w.r.t. their timing persistency.	
Bondavalli [9], [26]	Rel., avail.		Metrics (e.g., reliability, MTTF, steady state and immediate availability) are specified in use case diagrams using *measure of interest* tag.	Input parameters (e.g., fault occurrence, percentage of permanent faults, error latency, error propagation probability) are specified by using different combinations of stereotypes.	Repair delays are specified by using different combination of stereotypes.	Permanent faults can be quantified via the percentage of permanent faults tagged value of $\ll hardware \gg$ stereotype.	Error propagation probability is specified via the $\ll propagation \gg$ stereotype.

Continued on next page

TABLE X – continued from previous page

Approach	IR1	IR2	IR3	IR4-IP1	IR4-IP2	IR5	IR6
DalCin [13]	Rel., avail.	$\ll requirements \gg$ note symbols, specifying, e.g., max reliability and steady state availability are written using an ad-hoc language.					
D'Ambrogio [14]	Rel.			No UML extensions used. Input parameters of the fault-tree model derived from UML design are MTTF of nodes, communication paths, call and return actions and operations.			
Pai-Dugan [30]	Rel., avail.			Input parameters, e.g., failure rate, coverage factor, error propagation, are specified by using $\ll hardware \gg$ and $\ll software \gg$ classes.	Restoration rates are specified by using $\ll hardware \gg$ and $\ll software \gg$ classes.		Error propagation probability due to hw faults is specified via $\ll propagates\ error\ to \gg$ associations.

Continued on next page

TABLE X – continued from previous page

Approach	IR1	IR2	IR3	IR4-IP1	IR4-IP2	IR5	IR6
Cortellessa-Pompei [11]	Rel.			Input parameters (e.g., atomic failure probabilities of software components and logical links) are specified using stereotypes that specialize concepts introduced in the General Resource Modeling package of the SPT profile.			
Grassi [17], [18]	Rel.			Extend the usage of [11]'s annotations to hardware components and consider both atomic failures and failure PdFs.			

Continued on next page

TABLE X – continued from previous page

Approach	IR1	IR2	IR3	IR4-IP1	IR4-IP2	IR5	IR6
Jürjens [21], [22]	Rel., safety	Reliable/safe communication reqs. (e.g., max duration for data transmission, max probability of message loss, probability of eventual data delivery). They are specified using the $\ll guarantee \gg$ stereotype.		Communication failure assumptions (on nodes and links) are specified using the $\ll risk \gg$ stereotype.			
Pataricza [32]	Safety				SM transitions that model error correction are stereotyped $\ll ErrorCorrecting \gg$.		
Goseva [16]	Safety		No UML extensions used. Metrics are either derived from the UML model (e.g., component complexity, connector coupling) or estimated via safety analysis techniques (e.g., severity). Composite metrics (risk factors) are defined as functions of the basic ones.				

Continued on next page

TABLE X – continued from previous page

Approach	IR1	IR2	IR3	IR4-IP1	IR4-IP2	IR5	IR6
Hassan [19]	Safety		No UML extensions used. Metrics/ properties are estimated via safety analysis techniques (e.g., cost of hazard, failure probabilities, hazard guidewords).				

TABLE XI: Summary of contributions to IR7-IR10

Approach	IR7-Dom	IR7-Det	IP7-Con	IR7-Cons	IR7-Dep	IR8	IR9	IR10
Pataricza [31]	Usage of QoS values to qualify the failure domain, e.g., early/late timing failure.						Components move from *normal* to *faulty* states as effect of a fault injector request.	
Addouche [1], [2]					Use of SM states and guards to specify component state-based conditional failures		$\ll Indicator \gg$ and $\ll Cause \gg$ classes are related to the dynamic aspects of resources. The former are used to specify degraded/failure states, while the latter are used to specify logical conditions of failure occurrence.	

Continued on next page

TABLE XI – continued from previous page

Approach	IR7-Dom	IR7-Det	IP7-Con	IR7-Cons	IR7-Dep	IR8	IR9	IR10
Bernardi [4], [5]	A class diagram of failure modes is presented. Failures are classified, according to their impact on the automation system, in halting (passive and silent), degrading and repairing failures.			A *criticality level* attribute is associated to system components and functions to specify their failure criticality.				
Bernardi-Merseguer [6]	No UML extensions used. The QoS assessment is carried out under late timing failure assumption.							

Continued on next page

TABLE XI – continued from previous page

Approach	IR7-Dom	IR7-Det	IP7-Con	IR7-Cons	IR7-Dep	IR8	IR9	IR10
Bondavalli [9], [26]					Dependent failures are considered in case of system redundancy. A *common mode failure* tag is used to specify the common mode failure occurrences of components belonging to a redundant FT structure.		Stereotypes are used in order to represent failure and normal states/events in SM representing the behavior of redundancy manager components.	Stereotypes are used to identify elements of a redundant structure (manager, variant, adjudicator, tester, comparator). Tags can be associated to a group of elements to specify common mode failures and error detection coverage.
DalCin [13]								
D'Ambrogio [14]								
Pai-Dugan [30]					Several stereotypes are defined to model different kinds of dependencies between system components. The method supports the analysis of sequence failure dependencies.		Reconfiguration activities are modelled with SM. No specific UML extensions are used for this purpose.	Stereotypes are used to discriminate the type of spare components.
Cortellessa-Pompei [11]								

Continued on next page

TABLE XI – continued from previous page

Approach	IR7-Dom	IR7-Det	IP7-Con	IR7-Cons	IR7-Dep	IR8	IR9	IR10
Grassi [17], [18]								
Jürjens [21], [22]	Different stereotypes are introduced to specify the type of failure, i.e., $\ll crash/performance \gg$ for timing failures, $\ll value \gg$ for content failures.					$\ll risk \gg$ stereotype is introduced to specify failure assumptions.		$\ll redundancy \gg$ stereotype is used to specify the type of voting model designed within a FT structure.
Pataricza [32]							Normal and faulty behavior of components are integrated in a single SM. UML stereotypes are used to identify erroneous states and error correcting transitions.	

Continued on next page

TABLE XI – continued from previous page

Approach	IR7-Dom	IR7-Det	IP7-Con	IR7-Cons	IR7-Dep	IR8	IR9	IR10
Goseva [16]				Severity indices are associated to system components and connectors to estimate the risk factors.		Hazard parameters (e.g, severity, occurrence likelihood) associated to components and connectors are estimated to quantify the scenario risk factor.		
Hassan [19]				Cost of failure concept is introduced to estimate the failure consequences on the system components, connectors and scenarios. Although no UML extension is used, the costs of failure are annotated in UML sequence diagrams using note symbols.		Hazard parameters (e.g., cost, occurrence likelihood) associated to components and connectors are estimated to quantify the cost of scenario failures.		

REFERENCES

[1] N. Addouche and J. Antoine, C.and Montmain. UML models for dependability analysis of real-time systems. In *In Proc. International Conference on Systems, Man and Cybernetics*, volume 6, pages 5209 – 5214. IEEE CS., Oct. 2004.

[2] Nawal Addouche, Christian Antoine, and Jacky 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] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 01(1):11–33, 2004.

[4] 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.

[5] S. Bernardi, S. Donatelli, and G. Dondossola. A class diagram framework for collecting dependability requirements in automation systems. In *In Proc. of the 1st International Symposium on Leveraging Applications of Formal Methods (ISOLA'04)*, Paphos (Cyprus), October 2004.

[6] S. Bernardi and J. Merseguer. QoS Assessment via Stochastic Analysis. *IEEE Internet Computing*, pages 32–42, May-June 2006.

[7] S. Bernardi and D. Petriu. Comparing two UML Profiles for non-functional requirement annotations: the SPT and QoS Profiles. SVERTS - Satellite Events at the UML Conference, Lisbon (Portugal) - 2004.

[8] 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.

[9] A. Bondavalli, M. Dal Cin, D. Latella, I. Majzik, A. Pataricza, and G. Savoia. Dependability analysis in the early phases of UML-based system design. *Int. Journal of Computer Systems Science & Engineering*, 16(5):265–275, 2001.

[10] International Electrotechnical Commission. IEC-60300-3-1 standard: Dependability management.

[11] 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.

[12] V. Cortellessa, H. Singh, and B. Cukic. Early reliability assessment of UML based software models. In *In Proceedings of Third International Workshop on Software and Performance*, pages 302–309, Rome, Italy, July 2002.

[13] M. Dal Cin. Extending UML towards a Useful OO-Language for Modeling Dependability Features. In *In Proc. of 9th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2003 Fall)*, pages 325–330, Anacapri (Capri Island), Italy, October 2003. IEEE Computer Society.

[14] 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.

[15] M. Evans, N. Hastings, and B. Peacock. *Statistical distributions*. Wiley, New York, 2000.

[16] Katerina Goseva-Popstojanova, Ahmed Hassan, Ajith Guedem, Walid Abdelmoez, Diaa Eldin M. Nassar, Hany Ammar, and Ali Mili. Architectural-level risk analysis using uml. *IEEE Transactions on Software Engineering*, 29(10):946–960, 2003.

[17] 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.

[18] Vincenzo Grassi, Raffaela Mirandola, and Antonino 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.

[19] A. Hassan, K. Goseva-Popstojanova, and H. Ammar. UML Based Severity Analysis Methodology. Alexandria, VA, January 2005.

[20] A. Immonen and E. Niemelä. Survey of reliability and availability prediction methods from the viewpoint of software architecture. *Software System Model*, (7):49–65, 2008.

[21] 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.

[22] Jan Jürjens. Developing safety-critical systems with UML. In *UML 2003, San Francisco*, volume 2863 of *LNCS*, pages 360–372. Springer-Verlag, October 2003.

[23] Nancy G. Leveson. *Safeware*. Addison-Wesley, 1995.

[24] Michael R. Lyu, editor. *Handbook of Software Reliability Engineering*. IEEE Computer Society Press, 1996.

[25] M.R. Lyu. *Software Fault Tolerance*. John Wiley & Sons, Ltd., 1995.

[26] I. Majzik, A. Pataricza, and A. Bondavalli. Stochastic Dependability Analysis of System Architecture Based on UML Models. In R. De Lemos, C. Gacek, and A. Romanovsky, editors, *Architecting Dependable Systems, LNCS 2677*, Lecture Notes in Computer Science, pages 219–244. Springer-Verlag, Berlin, Heidelberg, New York, 2003.

[27] Object Management Group. *UML Profile for Schedulabibity, Performance and Time Specification*, January 2005. Version 1.1, formal/05-01-02.

[28] Object Management Group. *UML Profile for Modeling Quality of Service and Fault Tolerant Characteristics and Mechanisms*, May 2006. Version 1.0, formal/06-05-02.

[29] OMG. *A UML profile for Modeling and Analysis of Real Time Embedded Systems (MARTE)*, 2007. Working document, ptc/07-08-04.

[30] G. J. Pai and J.B. Dugan. Automatic synthesis of dynamic fault trees from uml system models. In *In Proc. of 13th International Symposium on Software Reliability Engineering (ISSRE-02)*, pages 243–256, Annapolis, MD, USA, November 2002. IEEE Computer Society.

[31] A. Pataricza. From the General Resource Model to a General Fault Modelling Paradigm ? Workshop on Critical Systems, held within UML'2000, 2000.

[32] A. Pataricza, I. Majzik, G. Huszerl, and V'arnay G. UML-based design and formal analysis of a safety-critical railway control software module. In G. Tarnai and E. Schnieder, editors, *In Proc. of Symposium Formal Methods for Railway Operation and Control Systems (FORMS03)*, pages 125–132, Budapest (Hungary), May 2003.

[33] 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.

[34] L. Shourong and A.H. Wolfgang. A UML Profile to model safety-critical embedded real-time control systems. In *Studies in Computational Intelligence (SCI)*, volume 42, pages 197–218. Springer-Verlag Berlin Heidelberg, 2007.

[35] G. Zoughbi, L. Briand, and Y. Labiche. A UML Profile for Developing Airworthiness-Compliant (RTCA DO-178B), Safety-Critical Software. In G. Engels, editor, *Proceedings of Models 2007*, volume 4735 of *LNCS*, pages 574–588. Springer-Verlag Berlin Heidelberg, 2007.