Dependability modeling and analysis of software systems specified with UML

SIMONA BERNARDI, Centro Universitario de la Defensa
JOSÉ MERSEGUER, Universidad de Zaragoza
DORINA C. PETRIU, Carleton University

The goal is to survey dependability modeling and analysis of software and systems specified with UML, with focus on reliability, availability, maintainability and safety (RAMS). From the literature published in the last decade, 33 approaches presented in 43 papers were identified. They are evaluated according to three sets of criteria regarding UML modeling issues, addressed dependability characteristics and quality assessment of the surveyed approaches. The survey shows that more works are devoted to reliability and safety, fewer to availability and maintainability and none to integrity. Many methods support early life-cycle phases (from requirements to design). More research is needed for tool development to automate the derivation of analysis models and to give feedback to designers.

Categories and Subject Descriptors: A.1 [**General Literature**]: Introductory and Survey; C.4 [**Computer System Organization**]: Performance of Systems—*Modelling techniques*; D.2.1 [**Software Engineering**]: Requirements/Specifications—*Methodologies*; D.2.4 [**Software Engineering**]: Software/Program Verification—*Formal methods*; D.2.8 [**Software Engineering**]: Metrics—*Complexity measures*; D.2.10 [**Software Engineering**]: Design; D.2.11 [**Software Engineering**]: Software Architectures; D.3.2 [**Programming Languages**]: Language Classification—*UML*

General Terms: Design, Languages, Reliability, Standardization, Verification

Additional Key Words and Phrases: Availability, maintainability, safety, model transformation, dependability analysis

ACM Reference Format:
S. Bernardi, S., Merseguer, J., and Petriu, D.C. 2011. Dependability modeling and analysis of software systems specified with UML. ACM Comput. Surv. 0, 0, Article 00 (2011), 47 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

1. INTRODUCTION

Dependability is a non-functional property (NFP) of a system, defined by [Avizienis et al. 2004] as the ability to avoid failures that are more frequent and severe than acceptable. Dependability encompasses a set of attributes: reliability, availability, maintainability, integrity and safety. The assessment of these attributes may imply quantitative and/or qualitative evaluation of the system, which has been a research topic since the early times of computing; the use of models for this purpose is extensively

José Merseguer has been supported by the European Communitys Seventh Framework Programme under project DISC (Grant Agreement n. INFSO-ICT-224498), by CICYT DPI2010-20413 and by Fundación Aragón I+D.

Author's addresses: S. Bernardi, Centro Universitario de la Defensa, Academia General Militar, Zaragoza (Spain); J. Merseguer, Departamento de Informática e Ingeniería de Sistemas, Universidad de Zaragoza (Spain); D.C. Petriu, Department of Systems and Computer Engineering, Carleton University (Canada).
Author's e-mail: simonab@unizar.es, jmerse@unizar.es, petriu@sce.carleton.ca.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2011 ACM 0360-0300/2011/-ART00 $10.00

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

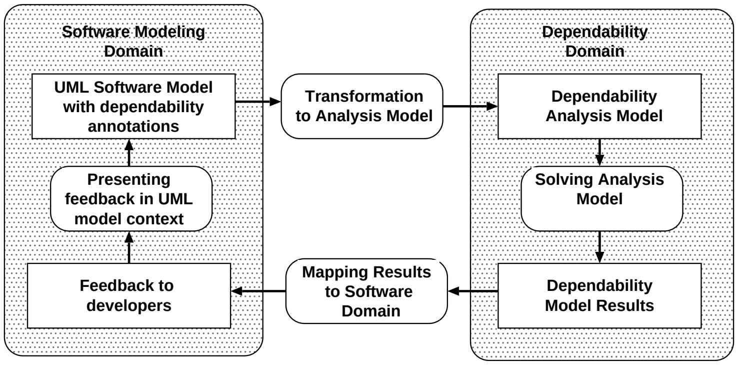

Software Modeling Domain
UML Software Model with dependability annotations
Presenting feedback in UML model context
Feedback to developers
Transformation to Analysis Model
Mapping Results to Software Domain
Dependability Domain
Dependability Analysis Model
Solving Analysis Model
Dependability Model Results

Fig. 1. Integrating dependability modeling and analysis in a UML-based MDD process.

recognized [Lyu 1996]. A *model* is an abstraction of the system for the purpose of understanding it before building it [Rumbaugh et al. 1991]. A *software system model* describes a specific system view; in a broad sense we can distinguish behavioral and structural software views, which together constitute the model of the system. A *dependability model* considers the abstractions needed to represent the failures of the system and their consequences. This implies that in some manner the dependability model needs to be related to the behavioral model of the system or at least to its abnormal behaviors. Models are developed using different kinds of languages and/or notations, some of them with an underlying mathematical formalism supporting some kind of analysis (e.g., fault trees, Markov chains, Petri nets or Bayesian networks). These are called *formal models*, analyzable models or models for analysis. The task of developing models is known as *modeling*, while the task of analyzing quantitative or qualitative properties is known as *analysis*.

According to the literature, dependability attributes are assessed through formal dependability models or using technical methods (e.g., FMEA, HAZOP) [IEC-60300-3-1 2003]. However, the focus of this survey is not on dependability analysis models per se, but on approaches for modeling and analyzing dependability in the context of UML software models used for software development. We are particularly interested in works contributing toward the integration of dependability modeling and analysis within the UML-based model-driven software development process [Schmidt 2006]. Figure 1 shows an ideal process, where a UML software model used by the software developers is extended with dependability annotations and then (automatically) transformed into a formal dependability model , which is solved with existing solvers and methods. The dependability results from the formal model are mapped to feedback to the developers, expressed in terms of the software model. The process bridges two domains: the software modeling domain and the dependability analysis domain and is dealing with two categories of models and modeling languages: a) for software development and b) for dependability analysis. This survey is focused on how and what dependability annotations need to be added to UML software models (i.e., in the software modeling domain) in order to support the derivation of dependability analysis models that can be used for dependability analysis.

The most widely used modeling language for software development is the Unified Modeling Language [UML 2005], standardized by the Object Management Group (OMG). There is a large body of work extending UML with concepts required for carrying out quantitative and qualitative analyses of different non-functional properties, such as performance, schedulability, dependability, security. For instance, a UML-based methodology for unifying dependability modeling and analysis by associating the concepts of UML models, dependability, and mathematical analysis was promoted in [Pataricza and Györ 2004]. Analyzing non-functional properties (NFP) based on design models allows developers to start verifying early in the life cycle whether the system under construction will meet its non-functional requirements. It is less expensive to analyze different design alternatives and choose the best one at an early phase, when the investment in building the product has not been completely made yet. However, the verification may continue throughout the lifecycle.

Adding NFP specifications to UML is possible due to a number of UML characteristics. First, a UML model can abstract the structure and behavior of the system and also the hardware platform where it will execute. The system can be described with a great detail and a variety of UML diagrams. Some are representing the system structure (e.g., class diagrams), others the behavior (e.g., use case, state machines, activity diagrams or sequence diagrams) and others the platform (e.g., deployment diagrams).

Secondly, UML contains standard extension mechanisms allowing users to define a *profile* for customizing UML models for particular domains and platforms. A profile allows refining standard semantics in strictly additive manner. Appendix 6 explains the basis of UML and its profile mechanisms to help understanding some terminology in the paper. So, a UML model together with the specification of NFPs by standard extension mechanisms represents a complete system model for carrying out the analysis of different non-functional properties.

Thirdly, it is possible to combine the use of multiple profiles for the same model, which would support consistent specification of different NFPs and their relationships, as well as the analysis of trade-off between different NFPs.

A milestone in extending UML is the *UML Profile for Modeling and Analysis of Real-Time and Embedded systems* [UML-MARTE 2009] standardized by OMG, that addresses the issue of attaching information necessary for quantitative NFP analysis to UML model elements (more specifically, for performance and schedulability analysis). Previously to MARTE, OMG adopted other two profiles for analysis purposes, the Schedulability, Performance and Time Specification [UML-SPT 2005] and the profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms [UML-QoS&FT 2008]. All these profiles satisfactorily address the problem of quantitative analysis aimed at performance evaluation. However, so far no standard profile has been proposed or adopted for dependability modeling and evaluation (neither quantitative nor qualitative).

For the reasons indicated above, the focus of this survey is on dependability modeling and analysis of software and systems specified with UML. The goal of the survey is to cover a large body of work published in the last decade that models and analyses reliability, availability, maintainability and safety (RAMS) NFPs with UML. We apprise the reader that, although we have looked for UML extensions to model and analyze *integrity*, we have not found any. According to [Avizienis et al. 2004], another important NFP, *security*, is sharing two attributes, availability and integrity, with dependability and is adding one more, confidentiality. We decided not to include confidentiality in this survey because the kind of models and techniques used for the analysis of confidentiality are different from those for dependability. Hence, we focus the survey strictly on dependability.

Considering the concern of dependability modeling, a majority of the works from literature use profiling mechanisms, as explained in Appendix 6. We identified a common approach, which consists of incorporating dependability information in UML designs in the form of UML extensions, producing UML-annotated models. Therefore, these UML-annotated models can be seen as (non-formal) dependability models where the structural and behavioral views are given by the UML design and the dependability view by the UML extensions and the abnormal behaviors described in the behavioral view. Such models constitute the starting point of the process presented in Figure 1. Among the surveyed works dealing with the dependability analysis aspect, two approaches can be recognized. A first approach transforms the UML-annotated models, i.e. the (non-formal) dependability model, into a formal dependability model used for analysis (e.g., Fault Trees [Vesely et al. 1981] for safety and reliability analysis, stochastic Petri nets [Ajmone-Marsan et al. 1995] for reliability analysis), as in Figure 1. Such formal models, as well as methods and tools for their analysis have been studied and developed for many years, even before the introduction of UML.

The second approach uses the very same annotated UML model, without model transformations, to apply well-known dependability techniques (e.g., HAZard and OPerability study [UK Ministry of Defence 2000]).

We believe that this survey will help researches by informing them which topics in this field have already been addressed and which ones are still open and need additional effort. In particular, our study of the literature has revealed that the modeling concern has been more thoroughly treated so far than the analysis concern, which requires more research. Fundamentally, the translations of UML-annotated models into *dependability* analysis models need to be improved, as well as the eventual evaluation of analysis models. Work should be also invested in the feedback from NFP analysis to the design model, for instance being able to pinpoint design flaws from analysis results. Tool support is another lack we found. Another problem is that there are many proposals for extending UML models with dependability annotations, each one covering only a subset of dependability aspects and using the concepts and terminology inconsistently with each other. This is a consequence of a fact mentioned before, that so far there is no standard UML profile for dependability NFPs that would unify all the necessary annotations. This is another aspect that needs improvement.

This survey comes to fill a gap not addressed in literature by other works. [Gokhale 2007] provides the state of the art in architecture-based software reliability analysis. Differences with our paper are that we deal not only with architectural aspects but also with the software requirements and detailed design, and besides reliability we address the other RAMS attributes. However, [Gokhale 2007] is not restricted to architectural designs based on UML, they also consider other languages and architectural proposals. [Immonen and Niemelä 2008] survey specifically concentrates on reliability and/or availability prediction methods based on software architectural models. Finally, the work in [Balsamo et al. 2004] is similar to ours in objectives, but targeted to performance evaluation instead of dependability.

The paper is organized as follows. In Section 2 we review the most important dependability concepts and introduce a discussion in the UML context, so to start drawing the boundaries of the state of the art in UML dependability modeling and analysis. Section 3 introduces the evaluation criteria that, along with dependability concepts, are used to classify and discuss the surveyed works. Sections 4 and 5 present the actual discussion and comparisons of the works. Conclusions are drawn in Section 6.

2. DEPENDABILITY MODELING AND ANALYSIS BACKGROUND

This section establishes the basic framework of our work, overviewing fundamental dependability concepts. Since the survey covers both UML modeling of dependability as well as its analysis or evaluation, we start by recalling dependability concepts, such as fault, error and failure, and continue with issues related to dependability analysis, such as traditional approaches to dependability assessment, different kinds of analysis, and basic concepts such as dependability measure. While recalling the basic dependability definitions for these concepts, we also discuss the implications of expressing them in UML (see the subsections entitled “Discussion in the UML context”).

The goal is to help readers understand the challenges addressed by the surveyed works in order to seamlessly represent such a large set of concepts in UML models.

Definitions of dependability can be obtained from multiple sources; here we mainly follow [Avizienis et al. 2004; Lyu 1996]. In [Avizienis et al. 2004] the dependability terminology is surveyed from a systemic point of view, while [Lyu 1996] specifically addresses the software domain, focusing on reliability. Another source is [Leveson 1995] for definitions of safety related concepts. Our intention is not to offer a comprehensive guide on the large number of existing dependability definitions (which was the goal of the sources just mentioned), but to clarify the conceptual framework for the survey.

The dependability concepts and issues addressed in this section are summarized in a checklist shown in Table I. Each concept is given a label that will permit to easily pinpoint it in later discussion (in Sections 4 and 5). The last column of the Table indicates whether an issue is addressed in the survey or not. The goal is to place the scope of the survey in the dependability arena. The remainder of the section is organized according to main dependability issues: system view, attributes, threats and means of the dependability.

2.1. System view and basic definitions

Before discussing basic definitions for dependability, we need to specify the context in which they apply. Both [Avizienis et al. 2004; Lyu 1996] consider a component-based view of a *system* for which dependability is investigated. A *component* is seen as an entity that interacts with other entities (hardware and/or software) and with the physical world. A component is then by itself a system made up of other components which interact through *connectors*. It is considered that the system or a component provides an expected *service* to the environment or the user, which is a sequence of outputs agreeing with a given specification [Lyu 1996].

An accepted definition of dependability is the ability of a system to avoid failures that are more frequent and severe than acceptable [Avizienis et al. 2004]; it encompasses the following attributes (labeled **DA** in Table I): reliability, availability, safety, maintainability and integrity. According to [Avizienis et al. 2004], *reliability* (**DA.R**) ensures the continuity of correct service for a given system and is defined in [ANSI/IEEE 1991] as *the probability of failure-free software operation for a specified period of time in a specified environment*. For repairable systems [Arnold 1973], i.e., those that can be recovered after failure, the *availability* (**DA.A**) and *maintainability* (**DA.M**) attributes are of special significance. The former is defined as the systems readiness for correct service, and the latter represents the systems ability to undergo modifications and repairs. *Integrity* is related to both dependability and security. In a broad sense, integrity is the absence of improper system alterations. [ANSI/IEEE 1991] defines it as the degree to which a system or component prevents unauthorized access to, or modification of, computer programs or data. In [Biba 1977] integrity ensures that the system cannot be corrupted to perform in a manner contrary to the original determination. In safety-critical systems, the *safety* attribute (**DA.S**) emphasizes the absence of catastrophic consequences as a result of the system and/or software usage [Leveson 1995].

2.1.1. Discussion in the UML context. The fundamental concepts of system, component, connector and service can be represented using notation offered by the UML diagrams, as these concepts belong to the UML vocabulary. Hence, no additional effort has been carried out by the surveyed works to introduce them. What UML does not initially have is the ability to specify dependability attributes and their properties (e.g., how to define a measure of reliability in a UML diagram). A frequent solution uses the UML profiling mechanism, which allows users to extend UML with domain-specific concepts defined as stereotypes, tagged values and constraints [UML 2005], while using standard UML editors. Appendix 6 describes how the UML profiling mechanisms work. We finally remark that, as mentioned in the introduction, all dependability attributes except integrity have been addressed in some of the UML works herein surveyed.

2.2. Dependability model-based evaluation

During the first stages of the life-cycle, when implementations are not yet available, dependability attributes can be evaluated by solving formal *dependability models*. Indeed, *dependability measures* (**DM** in Table I) represent a quantified estimation or calculation of a dependability attribute [Hosford 1960]. However, not all dependability attributes can be quantified. For instance, in the safety analysis context, *safety properties* (**DM.S**) are not strictly measures, but rather qualitative indicators used to express the level of injury caused by system hazards or seriousness associated to system's unsafe states [Leveson 1995].

Once an estimation of a dependability measure is obtained by solving the formal dependability model, it has to be checked against the *dependability requirements* (**DR**) [Littlewood and Strigini 1993]. A dependability requirement can be thought of as an upper or lower bound (**DR.BOUND**) of a specific dependability measure. However, in the case of safety, a requirement (**DR.S**) represents the satisfaction of a given safety property (**DM.S**).

It is important to recall that the computation of reliability, availability and maintainability measures basically means a *quantitative* evaluation of the formal dependability model, while safety properties imply the *qualitative* evaluation of the model [Billinton and Allan 1992]. This does not necessarily mean that the nature of safety models has to differ from the others; for example, fault-trees or Petri nets can be used to perform both forms of evaluation, while Bayesian networks aims just to quantitative evaluation. It is also true that safety properties can be checked without an underlying formal dependability model [Leveson 1995], but with specific techniques such as HAZard and OPerability study (HAZOP) or Functional Failure Analysis (FFA).

2.2.1. Discussion in the UML context. As explained in the Introduction, a UML model is able to describe both the structural and behavioral views of the system, hence the works here surveyed have extended UML to specify *measures* and *requirements*. In fact, the majority of the surveyed works propose some mechanism to include measures and/or requirements in the UML models. As previously discussed, most of the works surveyed in this paper introduce an (automatic) transformation of the UML model with dependability annotations into a formal dependability model (e.g., Bayesian networks, Petri nets, fault trees) which can be used for computing the desired measures. Such formal dependability models will need to interpret the measures as expressed in the original UML model.

We identified some works interested in a kind of measures similar to *software complexity measures* (**DM.C**), which differ from the measures discussed above since they are defined in model terms. They are indirect dependability measures that in our scope refer either to "failure-proneness" in software components as in [Goseva-Popstojanova et al. 2003] or to the maintainability of the UML design as in [Genero et al. 2007]. The latter, for example, uses measures such as diagram structural complexity (e.g., number of associations) or size measures (e.g., number of classes) in relationship with the maintainability of UML class diagrams. Another example is the cyclomatic complexity of UML state machines defined in [Goseva-Popstojanova et al. 2003].

2.2.2. Examples of dependability measures and properties. In the following, we describe some important dependability measures most of them used in the surveyed works. The measures for reliability **DM.R**, availability **DM.A** and maintainability **DM.M** are usually defined with respect to time or the number of program runs [Lyu 1996; Trivedi

2001; Hosford 1960; Johnson 1989]. The execution time or calendar time are appropriate to define the reliability as:

$$R(t) = Prob\{\tau > t\} \tag{1}$$

that is, reliability at time t is the probability that the time to failure (τ) is greater than t or, the probability that the system is functioning correctly during the time interval $(0, t]$. Considering that $F(t) = 1 - R(t)$ (i.e., unreliability) is a probability distribution function, we can calculate the expectation of the random variable τ as:

$$\int_0^{\infty} t\ dF(t) = \int_0^{\infty} R(t)dt \tag{2}$$

This is called *MTTF* [Johnson 1989] and represents the expected time until the next failure will be observed. Another measure is the *failure rate* (called also rate of occurrence of failures), which represents the probability that a component fails between (t, dt), assuming that it has survived until the instant t, and is defined as a function of $R(t)$:

$$h(t) = -\frac{1}{R(t)}\frac{dR(t)}{dt} \tag{3}$$

The *cumulative failure* function denotes the average cumulative failures associated with each point in time, $E[N(t)]$.

Maintainability is measured by the probability that the time to repair (θ) falls into the interval $(0, t]$ [Johnson 1989]:

$$M(t) = Prob\{\theta \leq t\} \tag{4}$$

Similarly, we can calculate the expectation of the random variable θ as:

$$\int_0^{\infty} t\ dM(t), \tag{5}$$

that is called *MTTR* (Mean Time To Repair), and the *repair rate* as:

$$\frac{dM(t)}{dt}\frac{1}{1 - M(t)} \tag{6}$$

A key reliability measure for systems that can be repaired or restored is the *MTBF* (Mean Time Between Failure) [Johnson 1989], that is the expected time between two successive failures of a system. Some of the addressed works also consider the system/service *reliability on-demand* that is the probability of success of the service when requested. When the average time to complete a service is known, then it might be possible to convert between MTBF and reliability on-demand.

Availability is defined as the probability that the system is functioning correctly at a given instant [de Souza e Silva and Gail 1989]:

$$A(t) = Prob\{\text{state} = UP, \text{time} = t\}.$$

In particular, the *steady state* availability can be expressed as function of *MTTF* and *MTTR* (or *MTBF*):

$$Availability_{\infty} = \frac{MTTF}{MTTF + MTTR} = \frac{MTTF}{MTBF}.$$

Safety properties (**DM.S**) are traditionally expressed in qualitative terms, such as safety levels or risk factors associated to failures or hazards [Leveson 1995]. Nevertheless, often they are defined in function of quantitative criteria. An interesting example is the safety integrity level (SIL) [IEC-61508 1998] that specifies the required protection against software of system failure and corresponds to an interval of the “average probability of failure to perform a safety function on demand” [IEC-61508 1998] (e.g., SIL1[10E-2, 10E-1), SIL2 [10E-3, 10E-2)). Other examples of safety properties are the probability of reaching (or of being in) a safe/unsafe state and the tolerable accident rate.

Integrity is a property common to dependability and security. Similar to SIL, the concept of integrity level was defined in [Biba 1977], in security context, to grade damage caused by information sabotage. However, for integrity it is more common to use procedures to verify and ensure the integrity of the system than to use formally defined measures to compute grades of integrity. [Clark and Wilson 1987] presented a formal model for integrity verification, while [Sailer et al. 2004] developed an architecture for integrity measurement that relies on code measurement based on standards defined by [TCG 2011].

2.3. Dependability threats

Faults, errors and failures are usually referred as threats to dependability (**DT**) [Avizienis et al. 2004]. They are seen as a causal chain (F-E-F) that threatens the dependability of a system in the sense that the chain completion leads the system to a state that reports incorrect service or outage. More specifically, in F-E-F a fault is the cause of an error; in turn, the error is part of a state of the system that may lead to a failure (or service failure). In this causal view, an error is seen as an intermediate stage between failure and fault.

In [Avizienis et al. 2004] a very rich and precise taxonomy of faults is given. They account, among others, for hardware/software faults, development/operational faults, malicious/non malicious, fault *persistence* (**DT.FP**) and fault *occurrence* (**DT.FO**). The latter refers to “single” and “multiple” fault assumption. The quantitative characterization of a “single/multiple” fault assumption distinguishes between the rate and probability of fault occurrences (**DT.FOQ**)[1].The faulty behavior of the components, connectors and services (**DT.FB**) means to identify the states of these elements in which a fault is active [Hawkings et al. 2003].

“Erroneous behavior” (**DT.EB**) of a component, connector or service is the counterpart of “faulty behavior” (**DT.FB**), i.e., the characterization of error states for components, connectors and services due to a fault occurrence [Hawkings et al. 2003]. **DT.EB** and **DT.FB** are common in the model-based dependability. The quantitative characterization of an error (**DT.EQ**) is the probability of its occurrence assuming that the fault has positively occurred (**DT.FOQ**). On the other hand and due to the component-based assumption, sometimes a component raises an error that does not reach the system boundaries, which means that it does not cause a failure. This obviously happens when the service delivered by the component is not in the system interface. However, such component may offer its service to another internal component; this may lead to *error propagation* between components [Cortellessa and Grassi 2007].

Failures or service failures are events that occur when the user perceives that the system ceases to deliver the expected service [Avizienis et al. 2004]. A failure can be classified according to different *modes* (**DT.FMD** to **DT.FMDep**) [Powell 1992]. A “failure mode” is the way in which the system manifest the deviation from correct to incorrect service. A failure mode w.r.t. the domain (**DT.FMD**) is classified as a “content” and/or “timing” failure. Detectability (**DT.FMDet**) distinguishes “signalled” failures, those in which the system sends a warning signal to the user, and “unsignalled” ones. The consistency criteria (**DT.FMC**) differentiates “consistent” from “inconsistent” or

[1] Note that DT.FOQ and DT.EPQ are also dependability measures.

byzantine failures [Lamport et al. 1982]. The consequence of a failure grades it into *severity levels* (**DT.FMSL**) (we referred to this concept in Section 2.2 as a safety measure, concretely *safety level*). Different criteria can be used to define severity levels, for example according to the standard [MIL-STD-882d 1999] they are rated as catastrophic, critical, marginal or minor. Finally, failures can be "dependent" or "independent" (**DT.FMDep**). Failures are dependent when the affected components share the cause of the failure, i.e., the error is common to all of them. An example of multiple dependent failures is a "common failure mode", that typically occurs within a redundant structure.

"Failure behavior" (**DT.FailB**) of a component (connector or service) refers to the specification of failure events/conditions that lead to the degraded/ failure states of the component (connector or service) as well as the degraded/failure states themselves [Bondavalli et al. 2001].

For safety-critical systems, the concepts of fault, error and failure are supplemented with that of *hazard*. [Leveson 1995] defines *hazard* as a state or set of conditions in a system that, together with other conditions in the environment of the system, will inevitably lead to an accident. An *accident* is an undesired and unplanned (but not necessarily unexpected) event that results in (at least) a specified level of loss. For every possible hazard in the system it is important to know, at least, its *origin* (**DT.HO**) and main characteristics, i.e., *severity* (**DT.HS**) and *likelihood* (**DT.HL**) [Leveson 1995]. The hazard severity is defined as the worst accident that could result from the hazard, and as in the case of failures, hazard severity can be graded by *severity levels* such as minor, marginal, critical and catastrophic. The hazard likelihood can be defined quantitatively or qualitatively (e.g., frequent, probable, occasional, remote, improbable, impossible). Severity and likelihood are combined to obtain the *hazard level*. Some safety-critical techniques, such as HAZOP [UK Ministry of Defence 2000], use guide-words (**DT.HGW**) and parameters (specifics of the process in study) to identify hazards in the system. For example, the set of guide-words in HAZOP is: No, More, Less, As Well As, Reverse and Other Than.

2.3.1. Discussion in the UML context. We found that only two kind of fault characteristics are addressed in the surveyed works: fault *persistence* (**DT.FP**) and fault *occurrence* (**DT.FO**). As for *error propagation* we found that only its description (**DT.EP**) was given in surveyed works, e.g., [Pai and Dugan 2002], and a quantitative characterization (**DT.EPQ**), e.g., [Yacoub et al. 2004], which basically specifies the probability of propagation occurrence.

2.4. Dependability means

The problem of achieving a dependable software/system is usually addressed by applying four technical methods, also known as dependability means [Avizienis et al. 2004]: *fault prevention*, *fault removal*, *fault tolerance* and *fault forecasting*. *Fault prevention* encourages the use of techniques that prevent the system from faults [Chillarege et al. 1992]. Formal methods are useful for performing automatic software verification that, together with software testing, are common techniques for *fault removal* [Boehm 1984; Weyuker 1982]. *Fault tolerance* techniques [Avižienis 1967; Lyu 1995] aim at avoiding failures despite the presence of faults after the software or system is deployed. The last method, the *fault/failure forecasting* is carried out through the qualitative and quantitative evaluation of the system behavior with respect to faults/failures occurrences [Meyer 1980].

Fault tolerance techniques offers "system recovery" (**FT.R**) and "error detection" to achieve failure avoidance. Recovery tries to transform an erroneous or faulty system state into a correct one by using "error handling" and "fault handling" techniques. Error handling uses rollback, rollforward and compensation, while fault handling relies on diagnosis, isolation, reconfiguration and reinitialization.

Fault tolerance techniques can be implemented in different ways (e.g., redundancy, n-version, reflection or self-checking component) [Avizienis 1985; Huang and Kindala 1996]. “Redundancy” (**R**) implies to describe:

- type[2] (**R.T**): information, software and hardware;
- level (**R.L**): number of components in a redundant structure;
- failures (**R.F**): maximum number of tolerated failures;
- roles played by the component within the FT structure (**R.R**): replica, controller, adjudicator, voter, hot/cold/warm spare.

Maintenance[3] follows “fault handling” in the life-cycle and it refers to *repairs* (**M.R**) as well as *modifications* (**M.M**) of the system during the use phase [ISO/IEC 14764 2006]. The distinction between fault tolerance and maintenance is that the latter is carried out by an external actor. Moreover, repair and fault tolerance are much related concepts; actually, repair is seen sometimes as a fault tolerance activity.

2.4.1. Discussion in the UML context. Among dependability means *fault tolerance* is the mean the surveyed works have paid more attention. “System recovery” is the only fault tolerance technique addressed in the works here surveyed.

Fault prevention and fault removal have not been addressed, it seems reasonable since *fault prevention* is mostly a concern of development methodologies rather than of modeling notations, while *fault removal* is carried out during the development and system usage stages, whereas UML is usually exploited in early life-cycle stages. Different is the case of *fault/failure forecasting*, that can be carried out during development (evaluation testing [Boehm 1984]) but also in early stages through modeling. Dependability models mentioned in Section 2.2 for measures estimation can be also useful for forecasting faults or failures. A common approach in some of the surveyed works is that analysts try to identify where the fault/failure could be located in the dependability model (e.g., a Petri net), and then trace back to the UML design where the fault/failure emerges in the system.

Table I: Checklist for dependability concepts

(1) Dependability issue addressed in this work

L1	L2	Dependability concept or issue	Restriction	(1)
DA	Dependability attributes			
	R	Reliability		✓
	A	Availability	repairable system	✓
	M	Maintainability	repairable system	✓
		Integrity	dependability & security	
	S	Safety	safety-critical system	✓
DM	Dependability measures			
	R	Reliability measures.		✓
	A	Availability measures.	repairable system	✓
	M	Maintainability measures.	repairable system	✓
		Integrity measures.		
	S	Safety measures and safety properties.	safety-critical system	✓
	C	Software complexity measures.	design level	✓
DR	Dependability requirements			
	BOUND	Upper/lower bound requirements on dependability measures.		✓

Continued on next page

[2] Values associated to *type, level, failures* and *roles* are representative examples of the ones found in the works analysed in this paper.

[3] It is important to note the difference between maintainability (a dependability attribute) and maintenance (a technical method to achieve dependability).

Table I – ***continued from previous page***

(1) Dependability issue addressed in this work

L1	L2	Dependability concept or issue	Restriction	(1)
	S	Safety properties to be checked w.r.t. the system behavior.	safety-critical system	✓
DT	Dependability threats (Fault)			
	FP	Fault persistence.		✓
	FO	Fault occurrence.		✓
		Other Fault classifications: hw/sw, development/operational faults.		
	FB	Faulty behavior of components, connectors, services.		✓
	FOQ	Fault occurrence quantitative characterization.		✓
DT	Dependability threats (Error)			
	EB	Erroneous behavior (error states).		✓
	EQ	Error quantitative characterization.		✓
	EP	Error propagation.		✓
	EPQ	Error propagation quantitative characterization.		✓
DT	Dependability threats (Failure)			
	FMD	Failure mode w.r.t. the domain.		✓
	FMDet	Failure mode w.r.t. the detectability.		✓
	FMC	Failure mode w.r.t. the consistency.		✓
	FMSL	Failure mode w.r.t. the consequence (severity levels).		✓
	FMDep	Failure mode w.r.t. the dependency.		✓
	FailB	Component, connector or service failure behaviour.		✓
DT	Dependability threats (Hazard)			
	HO	Hazard origin.	safety-critical system	✓
	HS	Hazard severity (severity levels).	safety-critical system	✓
	HL	Hazard likelihood.	safety-critical system	✓
	HGW	Hazard guide-words.	safety-critical system	✓
FT	Fault tolerance			
		Error detection.		
	FT.R	Recovery.		✓
M	Maintenance		repairable system	
	M.M	Modifications.		✓
	M.R	Repair.		✓
R	Redundancy (A fault tolerance implementation)			
	R.T	Type.		
	R.L	Level.		
	R.F	Failures (max. number of tolerated failures).		
	R.R	Roles.		

3. EVALUATION CRITERIA

This section introduces and discusses the set of evaluation criteria that constitute the basis for analysis in the survey. Having a set of evaluation criteria will allow us to present, discuss and classify the surveyed works, tasks that are carried out in Sections 4 and 5. For an easy presentation, we merged these criteria into three groups summarized in Table II. The first group helps to understand how the surveyed approaches deal with important software engineering and UML modeling aspects. The second group concentrates on dependability concerns and the third addresses the quality of the approaches. Some criteria in the first and third groups have been taken from the surveys of [Balsamo et al. 2004] and [Immonen and Niemelä 2008], the rest have been identified during literature review, while reading and comparing the surveyed papers.

3.1. Software engineering and UML

(*C1*) **Life-cycle phase**. Almost all of the surveyed approaches aim to obtain dependability results early in the software life-cycle. Advantages of getting dependability results prior to implementation were discussed in the Introduction. Most of the sur-

Table II. Summary of evaluation criteria

Software engineering & UML		
Code	*Criteria*	*Values*
C1	Phase	requirements; architecture; design; implementation; deployment
C2	Diagrams	class; object; UC; SM; act.; seq.; IOD; col.; comp.; deploy.
C3	Process	general; use case; CBSE; SPL; MDD
C4	Software domain	general; RTES; SOA
C5	Application domain	general; aerospace; automotive; railway control; automated production
C6	Specification	profile; OCL; non-extended UML models; extensions
C7	Tool support	yes; no

Dependability characteristics		
Code	*Criteria*	*Values*
C8	Attribute	DA
C9	Analysis type	qualitative; quantitative (stochastic; non stochastic)
C10	Analysis model	FMEA; HAZOP; Fault trees; stochastic Petri nets, ...
C11	Input Parameters	DM, DT, FT, M, R
C12	Output Parameters	DM
C13	Requirements	DR

Quality		
Code	*Criteria*	*Values*
C14	Validation	case studies; empirical analysis; no validation
C15	Compliance	compliant; not compliant
C16	Results	N/A; basic; UML-feedback; sensitivity analysis
C17	Limitation	text describing the limitation

veyed approaches address the following phases: *requirements*, *design* and *software architectural design*. Although the latter can be considered a design sub-phase, we decided to analyze it separately due to the high number of approaches addressing it. We have also found works targetting the *implementation* and *deployment* phase.

(*C2*) **Diagrams**. As explained in Appendix 6, UML distinguishes between structural and behavioral diagrams. The kind of UML diagrams used by an approach strongly relates to the software life-cycle phase the approach addresses. In early phases, the structural diagrams used are mainly class and object, and the behavioral diagrams are use case, sequence, state machines, activity, collaborations and interaction overview diagram. In later phases, component and deployment diagrams are used.

(*C3*) **Software development process**. The surveyed approaches are applied in conjunction with a variety of software development processes. We identified the following cases:

- *general*: category including all "traditional" software development processes (iterative, incremental, waterfall or a combination of them).
- *use case*: use-case based process [Jacobson 1995].
- *CBSE*: component-based software development process [Szyperski 1998].
- *SPL*: specifically addressing Software Product Lines [Clements and Northrop 2001]
- *MDD*: Model Driven Development processes [Stahl and Völter 2006]. Although MDD can be applied in conjunction with all of the above categories, we treat it separately for simplicity. Note that the categories in the next criterion, the software domain, can also be addressed by using MDD techniques.

(*C4*) **Software domain**. The approaches can also be classified by the specific software domain they address. We have identified the following:

- *RTES*: targeted to real-time and embedded systems [Liu 2000].
- *SOA*: targeted to service-based systems [Bell 2008].

— *general*: approaches not focused on a specific software domain.

(*C5*) **Application domain**. Most of the approaches are focused on the development of general software systems, so they fall in what we call the *general* category. However, a few of them target a specific application domain, in particular: *aerospace, automotive, railway control* and *automated production*.

(*C6*) **Dependability specification**. A common technique for introducing dependability specifications (input parameters, output parameters and requirements – criteria *C11*, *C12* and *C13*) in the UML diagrams is by using *UML profiles* (see Appendix A for an explanation). However, some of the analyzed works use other approaches such as *OCL* restrictions (Object Constraint Language [OCL 2010] of UML), non-extended UML models or UML extension mechanisms (notes, stereotypes and tagged values) not organized around a specific profile.

(*C7*) **Tool support**. UML diagrams are supported by a large variety of CASE tools (Computer Aided Software Engineering), both commercial and non-commercial. The integration of an approach with a tool allows to incorporate the dependability specification as extensions to the UML model. The more advanced tools also incorporate as a feature the analysis of the underlying dependability model, while others make use of a third-party tool that actually supports the analysis of the model.

We consider important at this point to highlight that the criteria given in this section define differences among UML-based approaches and the rest of model-based approaches. Differences that remark the need of a separate study of these two concerns as we justify in the following.

A *dependability specification (C6)* accomplished with UML will use the resources UML offers, such OCL or extension mechanisms, however this does not apply to others model-based approaches. This aspect also involves *tool support (C7)* since comparisons among UML and non-UML tools will be meaningless, for the same reasons. Moreover, tools based on UML are nowadays dominant in the software engineering market and they have left few room for others. Another important aspect concerns the *development approaches (C3)*, since they can be strongly influenced by UML. For example, CBSE-like approaches develop architectural models according to UML component and deployment diagrams. More significant is the case of Use case based approaches, since today all of them follow the UML notation. Last but not least is the importance of some methodological aspects of the UML diagrams related to the *software life-cycle (C1)*. When a phase, say for example behavioral design, is accomplished with UML, it mandatorily has to be carried out using the UML diagrams for the purpose (e.g., state machines or sequence diagrams). However, for others model-based approaches the field is opened and they are not restricted by diagrams neither by the phase they can be applied, so, some of them use formal specification languages (for which hundreds of them exist, mere examples are Z [Z 2002], Troll [Jungclaus et al. 1996] or process algebra-like languages [Fokking 2000]), others use formal graphical models (e.g., stochastic Petri nets [Ajmone-Marsan et al. 1995], or queuing network models [Lazowska et al. 1984]), and the list may continue.

For all the reasons above, we argue that UML has evolved to the verge of being a language for which methodologies, specifications and tools around it are so complex and huge in number that conform a body of knowledge that deserves to be studied in isolation.

3.2. Dependability characteristics

This group of criteria is based on the dependability concepts in Table I discussed in Section 2. Besides, we have added two other important concerns, the *analysis type* and the *analysis model*.

(*C8*) **Attribute**. This is a cross-reference to **DA** from the checklist in Table I, so it refers to reliability **DA.R**, availability **DA.A**, maintainability **DA.M** and safety **DA.S**.

(*C9*) **Analysis type**. The nature of the dependability analysis may be either *quantitative* or *qualitative* [IEC-60300-3-1], and this strongly constraints the type of specification as we later discuss. For quantitative analysis we have found approaches that follow either *stochastic* analysis or *not stochastic*. For the sake of simplicity, “stochastic” and “probabilistic” are considered synonymous. In general, qualitative analysis aims to prove dependability properties, while quantitative analysis aims to estimate dependability measures.

(*C10*) **Analysis model**. There is a huge variety of dependability models and/or techniques used for dependability analysis. A few of them, like stochastic Petri nets [Ajmone-Marsan et al. 1995], are useful for both quantitative and qualitative analysis. However, others are exclusively targeted to qualitative analysis, such as HAZOP [UK Ministry of Defence], and others to quantitative analysis such Bayesian models or Markov models.

The analysis (either quantitative or qualitative) of a dependability model requires a proper specification of the input/output parameters and requirements, as explained below:

(*C11*) **Input parameters**. The input dependability parameters required by the approach to effectively carry out the proposed analysis. They support the specification of dependability characteristics (cross-reference to **DM**, **DT**, **FT**, **M**, **R** from the checklist in Table I).

(*C12*) **Output parameters**. The kind of dependability measures or properties the approach evaluates (cross-reference to **DM** from the checklist in Table I).

(C13) **Requirements**. The kind of requirements the approach supports (cross-reference to **DR** from the checklist in Table I).

Observe that, both the criteria (*C11*) and (*C12*) reference the dependability measure item of the checklist in Table I (**DM**). Indeed, an approach may require dependability measures as input parameters; for example, the Mean Time To Failure (MTTF) of system components, is needed to evaluate the overall system failure rate.

3.3. Quality

The criteria in this group will help to assess the overall quality and maturity of the surveyed approaches.

(*C14*) **Validation.** Some of the approaches have not been validated at the time of the publication, while for others a validation effort has been carried out. Most often the validation is carried out by the means of case studies that demonstrate, using realistic examples, how the dependability concepts are integrated with UML; sometimes, it is also shown how to obtain a dependability analysis model from the UML model. In a few works, empirical analysis is used instead, which mainly consists in extrapolating information, using testable working hypotheses, from the application of the proposal by third parties.
Therefore, in this criteria we will distinguish approaches belonging to one of the following categories: *case studies*, *empirical analysis* and *no validation*.

(*C15*) **Compliance with standards**. Several dependability standards exist, some with general purpose [ISO/IEC9126-1.2 2001; IEC-60300-3-1 2003; IEC-61508 1998], others targeted to specific application domains [MIL-STD-882d 1999; MIL-STD-1629A 1984; ARP-4754 1994; 1995; RTCA 1992; EN-50126 1999; 2001; 2001]. Compliance of an approach with a standard represents certainly an asset; this is especially true in the safety domain, where certifications following standards are common practice. In this regard we will classify an approach as *compliant* when it adheres to some standard or *not compliant* otherwise.

(*C16*) **Presentation of results**. When the dependability analysis has been carried out, results should be automatically interpreted in the problem domain, and subsequently presented to the software engineer. However, this is a tricky concern that, in our opinion, has not been satisfactorily solved by any of the surveyed approaches yet. Despite this drawback, some of the approaches present the results in some basic form (for example, in textual, tabular or plot forms). The best approaches feedback results to the same UML model (e.g. using trace visualization on sequence diagrams) and/or provide support for sensitivity analysis. We classify approaches in four categories: *N/A* (do not deal with this aspect), *basic* (offer some kind of basic support), *UML feedback* and *sensitivity analysis*.

(*C17*) **Limitations**. The analyzed approaches present drawbacks related to some of the previous criteria. For example, a few of them do not offer UML annotations for the definition of basic input parameters such as failures of components. The large number and diversity of these limitations hindered us from classifying them. However, we studied these limitations and pointed out the more relevant.

4. CONTRIBUTIONS

We surveyed 33 approaches (a total of 43 papers) addressing dependability modeling and analysis of UML-based software systems and collected the information regarding the criteria in previous section. They are presented according to the dependability attribute they address (criterion *C8*), i.e., reliability, availability, maintainability or safety. Approaches focussed on more than one dependability attribute will be presented in the last subsection.

4.1. Reliability

A first set of surveyed works contribute specifically to the reliability analysis of UML-based software systems. Following criterion *C1*, software architecture and design are the only phases of the software life-cycle dealt with by these works, so §4.1.1 and §4.1.2 address them, respectively. Since most of the works in this group follow a component-based approach, criterion *C3*, §4.1.3 discusses this aspect.

4.1.1. Software architecture. [D'Ambrogio et al. 2002] define a transformation of sequence and deployment diagrams (*C2*) into fault tree models (*C10*) to predict the system failure rate (*C12* - **DM.R**). Although no UML extension standard mechanisms are used (*C6*), 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, such as failure of nodes and communication paths, call/return actions and operations. Mean Time To Failure (MTTF) is assigned to such elements as input parameter (*C11* - **DM.R**).

Both [Yacoub et al. 2004] and [Rodrigues et al. 2005] aim at calculating the system reliability on-demand (*C12* - **DM.R**) as a function of the component/ connector reliability (*C11* - **DM.R**) and the scenario execution probabilities. [Yacoub et al. 2004] consider also the probability of error propagation between components (**DT.EPQ**). To compute the metric, [Yacoub et al. 2004] construct a probabilistic model, called Com-

Table III. Reliability and software architecture.

Works	Criteria
[D'Ambrogio et al. 2002]	C2, C6, C10, C11, C12
[Yacoub et al. 2004]	C11, C12
[Rodrigues et al. 2005]	C2, C10, C11, C12

Table IV. Reliability and software design

Works	Criteria
[Singh et al. 2001]	C2, C10, C12
[Cortellessa et al. 2002]	C2, C10, C11, C12
[Pai and Dugan 2002]	C6, C10, C11, C12
[Grassi et al. 2005; 2007]	C3, C6, C10, C11, C12
[David et al. 2009]	C10, C11, C17

ponent Dependency Graph (CDG), from sequence diagrams and develop an algorithm, based on the CDG. Instead, [Rodrigues et al. 2005] use Labeled Transition Systems to synthesize sequence diagrams and interpret them as Markov models (*C2*,*C10*).

4.1.2. Software design. [Singh et al. 2001; Cortellessa et al. 2002] use the Bayesian framework (*C10*) to derive the probability distribution of the system reliability (*C12* - **DM.R**) from UML use case and sequence diagrams (*C2*). [Cortellessa et al. 2002] improve the previous approach of [Singh et al. 2001] by considering also deployment diagrams and the connector failure (Beta) distribution beside the component failure (Beta) distribution and the use case execution probabilities (*C11* - **DM.R**).

[Pai and Dugan 2002] use dynamic fault tree as target formalism (*C10*) to evaluate the system unreliability (*C12* - **DM.R**) of fault-tolerant software systems. Unlike [D'Ambrogio et al. 2002], [Pai and Dugan 2002] introduce a set of stereotypes and tags to enrich UML system models with information needed for the reliability analysis (*C6*). In particular, tags are used to define input parameters, such as the failure rate of system components and the error propagation probability (*C11* - **DT.EPQ, DM.R**). The method supports the modeling and analysis of sequence error propagations (**DT.EP**) that lead to dependent failures (**DT.FMDep**), redundancies and reconfiguration activities (**FT.R**). 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 (**R.T**).

[Grassi et al. 2005; 2007] propose a model-driven transformation framework (*C3*) for the performance and reliability analysis of component-based systems. 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 (*C10*) can be derived for the computation of the service reliability (*C12* - **DM.R**). Grassi et al. uses the UML extensions of [Cortellessa and Pompei 2004] (*C6*) and complements them, by associating failure input parameters to both hardware and software components and by considering both atomic failures and failure probability distribution functions (*C11* - **DM.R**).

Finally, [David et al. 2009] focus on the identification of behavioral failure modes, e.g., with respect to their detectability (*C11* - **DT.FMDet**), in system design using Failure Mode and Effects Analysis (FMEA) technique (*C10*). Beside UML, SysML [SysML 2010] is also considered as modeling notation for the specification of the system functional behavior. The main drawback of the approach is that a *Dysfunctional Behavior data-base*, that organizes the knowledge about possible elementary failure modes of components, needs to be constructed and maintained to support an automated reuse of the method (*C17*).

Table V. Reliability and component-based systems.

Works	Criteria
[D'Ambrogio et al. 2002]	C3, C9
[Yacoub et al. 2004]	C3, C9
[Rodrigues et al. 2005]	C3, C9, C16
[Singh et al. 2001]	C3, C9, C17
[Cortellessa et al. 2002]	C3, C9, C17
[Grassi et al. 2005; 2007]	C3, C9
[Cortellessa and Pompei 2004]	C3, C6, C11

4.1.3. Component-based systems. Component-based modeling approach for the functional specification of the software (*C3*) is common to all of the aforementioned works, but [Pai and Dugan 2002] and [David et al. 2009]. These component-based proposals address reliability analysis from the quantitative and stochastic point of view (*C9*), using different dependability techniques. [Yacoub et al. 2004; Rodrigues et al. 2005; Singh et al. 2001; Cortellessa et al. 2002] assume failure independence of components and connectors; moreover, [Yacoub et al. 2004] considers independent scenarios and sequential execution of components and [Singh et al. 2001; Cortellessa et al. 2002] rely on the regularity assumption of component/ connector failure probability distributions (i.e, component busy periods are characterized by the same failure probability distribution) - (*C17*). They all provide support to analyse the sensitivity of the system reliability to the critical components, but only in [Rodrigues et al. 2005] the results of the analysis are fed-back to the UML model using annotations (*C16*).

Finally, [Cortellessa and Pompei 2004] provide support to the reliability analysis of component-based systems, in both phases (i.e., architecture and design), by proposing UML extensions within the frameworks of the SPT and QoS&FT standard profiles (*C6*). In particular, the set of 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 or (logical/physical) links (*C11* - **DM.R**), that is the probability that a component, or connector, fails in a single invocation of it.

4.2. Availability

To the best of our knowledge, [Bernardi and Merseguer 2006] is the ony work that tackles, exclusively, software availability. They devise a method to evaluate the quality of service (QoS) of fault tolerant (FT) distributed system design specification (*C1*), under late-timing failure assumption (*C11* - **DT.FMD**). The QoS metric is defined as a function of two non-functional requirements (*C13* - **DR.BOUND**): one is related to the system availability, i.e., the time to detect an error and isolate it (**FT.ED**), and the other is related to the cost of the FT strategy, i.e., communication overhead. [Bernardi and Merseguer 2006] propose a transformation of UML sequence, state-chart and deployment diagrams (*C2*), annotated with the SPT profile [UML-SPT], into a performability Generalized Stochastic Petri Nets (GSPN) model (*C10*). The latter is then analysed via simulation to evaluate, under different system configurations, the considered QoS metric (*C16*). State-charts are also proposed for the quantitative characterization of faults (**DT.FOQ**) as well as for the behavioral specification of different types of fault with respect to their timing persistency (**DT.FP**). In this case UML extensions have been explicitly introduced (*C6*), since the SPT profile does not support the specification of dependability parameters (e.g., fault occurrence probabilities).

Table VI. Availability.

Works	Criteria
[Bernardi and Merseguer 2006]	C1, C2, C6, C10 C11, C13, C16

Table VII. Maintainability.

Works	Criteria
[Genero et al. 2003; Genero et al. 2007]	C1, C2, C11, C12, C14, C15, C17

Table VIII. Safety and requirements elicitation.

Works	Criteria
[Allenby and Kelly 2001]	C2, C3, C5, C7, C10, C11, C13, C15
[Johannessen et al. 2001]	C2, C3, C5, C7, C10, C11, C15

4.3. Maintainability

[Genero et al. 2003; Genero et al. 2007] is the only approach, among the surveyed ones, that addresses specifically the maintainability of UML specifications during the design stage of the software lifecycle (*C1*). They rely on the software quality standard [ISO/IEC9126-1.2] (*C15*) and propose a set of metrics as good predictors of two maintainability sub-characteristics, that is understandability and modifiability (*C11* - **M.M**). The set of metrics includes both typical size metrics (e.g., number of classes, attributes and methods) and structuraly complexity ones (e.g., number of aggregations, dependencies and generalizations) - (*C12* - **DM.C**) - which can be applied on UML class diagrams (*C2*). An empirical analysis is carried out to evaluate the correlation between the proposed metrics and the considered maintainability characteristics (*C14*). Nevertheless, no guidelines are provided to the software designers on how to use such metrics to evaluate the maintainability of the UML class diagrams (*C17*).

4.4. Safety

Another group of works focus on safety-critical systems. They are presented according to the activities they support along the software life-cycle (*C1*). Moreover, we have considered of interest to jointly analyze those works focussed on risk assessment, irrespective of the phase of the life-cycle they apply, they are presented in §4.4.5.

4.4.1. Requirements elicitation. The safety requirement elicitation approaches are application domain-specific (*C5*) and use Use Case diagrams (*C2*,*C3*) to identify system level functionalities of aerospace software ([Allenby and Kelly 2001]) or in automotive domain ([Johannessen et al. 2001]). They are both compliant to safety-standards ([ARP-4754],[ARP-4761],[IEC-61508] - *C15*) and provide systematic methods to identify failure modes with the help of guidewords (*C11* - **DT.HGW**). In particular, [Allenby and Kelly 2001] apply a subset of HAZard OPerability guidewords (*C10*) to pre- and post-condition, guard condition, and scenario sections of use case descriptions to identify failure modes considering the domain and their consequence (*C11* - **DT.FMD, DT.FMSL**) and to derive safety requirements related to use cases (*C13* - **DR.S**). Instead, [Johannessen et al. 2001] adopt Functional Failure Analysis guidewords (*C10*) and failures are classified according to their consequence (*C11* - **DT.FMSL**). Unlike [Allenby and Kelly 2001], [Johannessen et al. 2001] provide also support to analyse the consequence of combined failures (*C11* - **DT.FMDep**). Both the approaches are characterized by a low degree of automation (to the best of our knowledge, no tools are available to support them - *C7*).

4.4.2. Software architecture. [Hansen et al. 2004] and [Iwu et al. 2007] use HAZard OPerability guidewords (*C10*,*C11* - **DT.HGW**) to identify hazards in software architec-

Table IX. Safety and software architecture.

Works	Criteria
[Hansen et al. 2004]	C2, C5, C10, C11, C17
[Iwu et al. 2007]	C2, C3, C5, C6, C7, C10, C11, C12, C13
[Liu et al. 2007]	C2, C3, C6, C10, C11, C12, C13, C16, C17

ture specification. Both the works address specific application domains (automotive in [Hansen et al. 2004] and embedded systems in [Iwu et al. 2007] - *C5*). While [Iwu et al. 2007] adopt an approach similar to [Allenby and Kelly 2001] (i.e., use case-based, *C3*), [Hansen et al. 2004] considers each model element in package, class, component, object, sequence and deployment diagrams (*C2*). Then, the main drawback of [Hansen et al. 2004] is the limited scalability of their proposal that, when applied of a real case study, may result in a time consuming activity (*C17*).

[Iwu et al. 2007] use also Fault Tree Analysis to combine faults that give rise to identified hazards (*C10*). Such faults are related to UML model elements (e.g., classes in class diagrams, messages in sequence diagrams - *C2*) and are used to establish derived safety requirements. Safety requirements and healthiness properties (*C13* - **DR.S**, *C12* - **DM.S**) are specified using Practical Formal Specification state machines (*C6*) and tool support is provided to check their consistency and completeness (*C7*).

[Liu et al. 2007] address safety analysis on the variations in software product-lines proposing a five-step approach (*C3*). In the first step, common and variability analysis is carried out to identify requirements for the entire product line and for specific product members. Hazard analysis is then performed by using Software Fault Tree Analysis (SFTA) customized to product-line domain (*C10*). The root node of the tree is typically a negation of a safety requirement, or it can be identified from pre-existing hazard lists (*C11* - **DT.HGW**), while the leaf nodes are labeled with a commonality or variation, previously identified. In the third step, such leaf nodes are mapped into architectural components, whose behavior is then modeled with a UML state-chart (*C2*). Safety requirement and failure scenarios are then derived from the fault tree (*C13* - **DR.S**) and, finally, behavioral safety-properties (*C12* - **DM.S**) are checked in state-based models (*C6*) through scenario-guided execution or animation (*C16*). The safety-properties that can be automatically checked include ordering logic and relative timing of failures of events while, due to the tool limitations, the verification of exact time values is not supported (*C17*).

4.4.3. Software design. [Hawkings et al. 2003], [Pataricza et al. 2003], [Ober et al. 2006] and [Zoughbi et al. 2007; 2006] consider the UML design of safety critical software. [Hawkings et al. 2003] address the preliminary system safety assessment of UML design. They construct a Fault Tree (*C10*) where hazardous basic events are related to classes and operations in UML Class Diagrams (*C2*). Then, the behavior of the classes - represented by UML StateCharts - is analysed in order to derive detailed safety requirements. Beside the normal behavior, the faulty behavior (*C11* - **DT.FB**) is modeled by adding extra transitions in the StateCharts with the help of hazard guidewords (e.g., omission, commission and value) (*C11* - **DT.FMD, DT.HGW**). A reachability analysis of the mutated StateChart is performed to check whether the introduced faulty behavior can lead to unsafe states. The derived safety requirements restrict the hazardous behaviors and are specified with OCL (*C6*) as contracts on classes/operations (*C13* - **DR.S**).

The modeling of the normal and the faulty behavior of a system component in a single state machine (*C2,C11* - **DT.FB**) is also proposed by [Pataricza et al. 2003], whose objective is to identify the error propagation paths leading to catastrophic failures in railway control software (*C5,C12* - **DM.S**). They define UML stereotypes (*C6*) for erro-

Table X. Safety and software design.

Works	Criteria
[Hawkings et al. 2003]	C2, C6, C10, C11, C13
[Pataricza et al. 2003]	C2, C5, C6, C11, C12
[Ober et al. 2006]	C2, C4, C6, C10, C11, C12, C13,
[Zoughbi et al. 2007; 2006]	C2, C5, C6, C11, C12, C14, C16, C17

neous states and correcting transitions in UML State Machines (*C11* - **DT.EB, DT.EP, FT.R**).

[Ober et al. 2006] present a technique for the verification of safety properties of real-time and embedded systems (*C4*) via model checking (*C12* - **DM.S**). A UML profile (OMEGA) is defined (*C6*) to specify timing constraints in the UML design (Class Diagrams and State Machines, *C2*) as well as dynamic and time dependent safety requirements (*C13* - **DR.S**). In particular, the latter are expressed by *observers* UML classes, whose behavior is described as a State Machine characterized by error or invalid states (*C11* - **DT.EB**). Both design and requirements are then transformed into communicating extended timed automata (*C10*) and the design is verified against the requirements using model checking techniques.

[Zoughbi et al. 2007; 2006] define a UML profile (*C6*) for the specification of safety concepts of aerospace software systems (*C5*) in the design phase to support the automated generation of certification-related information. The proposed UML extensions are compliant to the airworthiness standard [RTCA] (*C15*), they are used to record safety-related design decisions - e.g., failure consequence/ hazard severity (*C11* - **DT.FMSL, DT.HS**), roles within replicated structures (*C11* - **R.R**), safety/confidence levels (*C12* - **DM.S**) and complexity metrics (*C12* - **DM.C**) for collaboration, class, operation and relationship (*C2*) - and trace them back to the requirements (*C16*). Their approach is based on a rigorous definition of the profile (through a safety domain model) and an exhaustive completeness/consistency assessment with respect to the considered safety standard (*C14*). The main weakness is the use of dynamic concepts (through the profile) that extend typically static concepts, then leading to mixed static/dynamic views in the same UML diagram (Class Diagram) - (*C17*).

4.4.4. Software architecture, design and implementation. [Cancila et al. 2009] and [Lu and Halang 2007] focus on the specification of safety-related properties in different phases of the software life-cycle and for different purposes. Their approaches are characterized by a high automation degree, although only the work of [Cancila et al. 2009] is supported by a software tool (i.e., the used UML profiles are implemented as Papyrus [CEA-LIST 2008] plug-ins, *C7*).

[Cancila et al. 2009] consider software architecture and design specifications of railways transport systems (*C5*) and propose a UML profile (SOPHIA) for safety concerns (*C6*). SOPHIA relies on the OMG standard profile MARTE to express safety metrics (tolerable accident rate, i.e., TAR, and frequency of an accident *C12* - **DM.S**), requirements (i.e., maximum TAR *C13* - **DR.BOUND**), and characteristics (accident severity, accident severity/frequency table, accident frequency *C11* - **DT.HS, DT.HL**). The work also proposes an algorithm for the automatic generation of derived safety attribute values (i.e., TAR) in design model.

[Lu and Halang 2007] address the design of safety-critical distributed embedded real-time systems (*C4*) and the automated code derivation from UML design, considering PEARL as target programming language and Function Blocks [IEC-61131-1] (*C15*) to support the code reusability. A set of PEARL code structures are proposed as suitable for building applications which have to meet safety integrity level requirements [IEC-61508] (*C13* - **DR.BOUND**). The SIL-related PEARL constructs are rep-

Table XI. Safety and different phases in the life-cycle.

Works	Criteria
[Cancila et al. 2009]	C5, C6, C7, C11, C12, C13
[Lu and Halang 2007]	C4, C6, C13, C15, C17

Table XII. Safety and risk assessment.

Works	Criteria
[Goseva-Popstojanova et al. 2003]	C2, C10, C11, C12
[Hassan et al. 2005]	C6, C10, C11, C12, C15

resented as UML stereotypes and the Object Constraint Language (OCL) is used to specify constraints (i.e., pre- and post-conditions, invariants) on the expected execution of stereotyped components (*C6*). [Lu and Halang 2007] define UML extensions also for representing Function Block diagrams as UML component diagrams. A limitation of the approach concerns the timing-related issues which are vaguely dealt in the paper (*C17*).

4.4.5. Risk assessment. [Goseva-Popstojanova et al. 2003] and [Hassan et al. 2005] address the risk assessment step within the system safety analysis process. They both consider software architectures specified with UML. [Goseva-Popstojanova et al. 2003] estimate the scenario risk factor (*C12* - **DM.S**) from risk factors associated to software components and connectors by constructing and solving a Markovian model (*C10*). The component/connector risk factor is computed as the product of two safety metrics: the severity level (*C11* - **DT.FMSL, DT.HS**) and the complexity/coupling associated to the component/connector. The severity is obtained using FMEA technique (*C10*), while the component complexity and connector coupling are estimated considering the UML dynamic specifications - State Machines and Sequence Diagrams - (*C2,C12* - **DM.C**).

[Hassan et al. 2005] focus the problem of evaluating the failure severity based on UML specification. They integrate different severity techniques (FFA, FMEA and FTA - *C10*) to identify and relate system level hazards and component/connector failure modes (*C11* - **DT.HO, DT.HL, DT.HGW**). A cost of failure graph is then constructed to evaluate the cost of failure (*C12* - **DM.S**) of system execution scenarios, software components/connectors. The costs of failure are reported in the UML models with the use of notes (*C6*). Finally, the scenario and component/connector severity (*C11* - **DT.FMSL, DT.HS**) are obtained from the estimated costs of failure using a non-linear mapping. Both [Goseva-Popstojanova et al. 2003] and [Hassan et al. 2005] rely on the US military standard for safety critical systems [MIL-STD-1629A] (*C15*).

4.5. More than one dependability attribute

The remaining works address several dependability properties at a time. In the following we analyze them.

4.5.1. Reliability and availability. [Bondavalli et al. 2001; Majzik et al. 2003; Pataricza 2000], [DeMiguel et al. 2001] and [Leangsuksun et al. 2003] aim at analyzing the reliability and availability of UML-based software systems in different phases of the software life-cycle (from requirement to deployment, *C1*), using stochastic techniques (*C9*). They all adopt a model transformation approach to get an analyzable model from UML annotated models. Also [Dal Cin 2003] considers the reliability and availability properties, but his main contribution is providing a support for the specification of fault-tolerant and real-time software systems rather than the analysis.

[Bondavalli et al. 2001; Majzik et al. 2003; Pataricza 2000] is the most comprehensive approach, with respect to the checklist of Table I, for reliability and availability analysis of UML software architectures (*C1*). UML standard extension mechanisms (i.e., stereotypes and tags) are used for annotating dependability properties of software systems on UML specifications (*C6*). Through a rigorous graph transformation process, Timed Petri Net models (*C10*) are derived via an intermediate model, that captures the relevant dependability information from the annotated UML models. Component faulty behavior is triggered by independent fault injectors, modelled as State Machines (*C2*), which allow one to specify fault activation restrictions, such single fault assumption (*C11* - **DT.FO**). Several input parameters are defined (*C11*) for hardware and software components, such as fault occurrence rate (**DT.FOQ**), the percentage of permanent faults (**DT.FP**), the error latency for components with an internal state (**DT.EQ**) and repair delay (**DM.M**). The approach also supports the specification of error propagation between components (**DT.EPQ**) by assigning a probability to the model elements representing either relationships (e.g., associations) or interactions between such components (e.g., communication paths, messages). The set of dependability measures that can be evaluated (*C12* - **DM.R,DM.A**) includes the reliability probability distribution function, MTTF, the steady state and the immediate availability. Concerning the type of failures with respect to their dependency, both independent and dependent failures can be specified (*C11* - **DT.FMDep**). In particular, it is possible to assign common failure mode occurrence tags to redundant components belonging to complex FT structures (*C11* - **R.L, R.R**). Failures can be discriminated also with respect to the domain (*C11* - **DT.FMD**). 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 (*C11* - **DT.FailB**). Such extensions are used to analyse the failure conditions of the FT structures. The main drawback of the UML extensions proposed by [Bondavalli et al. 2001; Majzik et al. 2003; Pataricza 2000] 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 (*C17*). For example, a node, that models a hardware component in UML, must be stereotyped as *hardware* and *stateful* to specify the error latency.

[DeMiguel et al. 2001] consider the software architecture and detailed design UML specification of distributed real-time systems (*C1*,*C4*). Simulation models are generated automatically from UML models, annotated with dependability input parameters (*C11*) - i.e., object and network error occurrence, object time to failure and repair (**DT.EB, DT.EQ, DM.R, DM.M**). In particular, the tool [OpNet 1999] is used as simulation kernel (*C7*). The approach supports the evaluation of several dependability measures, such as object and network availability, object failure distribution (*C12* - **DM.A, DM.R**), and different type of statistics can be computed (i.e., mean, variance, distribution).

[Leangsuksun et al. 2003] derive from UML deployment diagrams (*C2*), normally used during the late software design and deployment stages (*C1*), fault tree and Markov chain models (*C10*) for the analysis of the reliability and availability (*C8*), respectively. The UML diagrams are annotated (*C6*) with node failure rate and repair rate parameters (*C11* - **DM.R, DM.M**). The method supports the computation of the reliability (i.e., survival function) and the steady state availability (*C12* - **DM.R, DM.A**), under hardware failure independence assumptions.

[Dal Cin 2003] proposes a UML profile (*C6*) for designing dependability mechanisms, that is hardware/ software components to be implemented or integrated in the real-time system (*C4*) to ensure fault tolerance (*C11* - **FT.ED, FT.R, R.R**). The proposed profile is aimed at supporting the quantitative evaluation of the effectiveness of the fault tolerant strategy adopted (*C9*), in terms of reliability and steady state availability (*C12* - **DM.R, DM.A**). It provides a language (i.e., SQIRL) for specifying stochastic

Table XIII. Reliability and availability.

Works	Criteria
[Bondavalli et al. 2001; Majzik et al. 2003; Pataricza 2000]	C1, C2, C6, C9, C10, C11, C12, C17
[DeMiguel et al. 2001]	C1, C4, C9, C11, C12
[Leangsuksun et al. 2003]	C1, C2, C6, C8, C9, C10, C11, C12
[Dal Cin 2003]	C4, C6, C9,, C11, C12, C13, C17

reliability and availability requirements of such mechanisms (*C13* - **DR.BOUND**). However, the profile lacks of a support to the modeling of the interactions among dependability mechanisms and the system components (*C17*).

4.5.2. Reliability, availability and maintainability. The following two approaches deal, as the previous ones, with reliability and availability, besides they address also maintainability (*C8*). Both [Addouche and Antoine 2004; Addouche et al. 2006] and [Bernardi et al. 2004a; 2004b] provide support in the requirement and design phases (*C1*) of the real-time software development (*C4*). In particular, the considered application domains are, respectively, automated production systems and distributed control automation systems (*C5*).

[Addouche and Antoine 2004; Addouche et al. 2006] define a profile *(C6)* that is compliant with General Resource Modeling package of the SPT profile [UML-SPT]. The UML extensions are used to annotate UML models with QoS characteristics (*C11* - **DM.R, DM.M**) and to derive probabilistic time automata *(C10)* for the verification of dependability properties via temporal logic formulas and model checking (*C12* - **DM.R, DM.A, DM.M**). 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 system resource. Such new classes are used to specify, via their attributes, the failure conditions and the possible degraded/ failure states of the resources (*C11* - **DT.FailB**). This mechanism can be used by the analyst to specify components state-based conditional failures (*C11* - **DT.FMDep**). The negative aspect of the approach is the poor separation of concerns, in fact new classes need to be defined and introduced in the system model, beside the classes representing the actual system components, for dependability analysis purposes *(C17)*.

[Bernardi et al. 2004a; 2004b] propose a set of UML class diagrams *(C2)*, structured in packages (i.e., a CD framework), as a reusable pattern to collect dependability and real-time requirements of distributed control automation systems and to support the design of an appropriate fault tolerance strategy. They also propose a systematic method for the derivation of dependability analysis models, such as TRIO [Ghezzi et al. 1990] temporal logic models *(C10)*. The class attributes define dependability or fault tolerance (**R.T**) characteristics; they can represent either input or output parameters (*C11*,*C12* - **DM.R, DM.A, DM.M**) or upper/lower bound requirements (*C13* - **DR.BOUND**), depending on the type of stereotype associated to the attribute *(C6)*. The fault-error-failure (FEF) chain [Avizienis et al. 2004] as well as the fault tolerance mechanisms are represented as class diagrams. In particular, fault classes include attributes that characterize the fault timing persistency and occurrence rates (*C11* - **DT.FP, DT.FOQ**) in system components. Error classes allows one to quantify error latencies, error probability and bit error rates (*C11* - **DT.EQ**) in automation functions. Finally, failures are classified according to their impact on the automation system in halting, degrading and repairing failures (*C11* - **DT.FMD**). From the analysis of TRIO temporal logic models it is possible to visualize traces of the system execution that concentrate on the predicates of interest *(C16)*. The requirement specification and analysis approach is compliant with the standard dependability management process [IEC-

Table XIV. Reliability, availability and maintainability.

Works	Criteria
[Addouche and Antoine 2004; Addouche et al. 2006]	C1, C4, C5, C6, C8, C10, C11, C12, C17
[Bernardi et al. 2004a; 2004b]	C1, C2, C4, C5, C8, C10, C11, C12, C13, C14, C15, C16, C17

Table XV. Reliability and safety.

Works	Criteria
[Mustafiz et al. 2008; Mustafiz and Kienzle 2009]	C1, C2, C3, C6, C8, C10, C11, C12, C13, C14
[Zarras et al. 2004]	C1, C4, C6, C8, C10, C11, C12
[Jürjens 2003; Jürjens and Wagner 2005]	C1, C6, C8, C10, C11, C12, C13

60300-3-1] *(C15*) and it has been applied on a primary substation of power distribution network *(C14*). Unfortunately, the customization of the CD framework for a given application is a time consuming activity; moreover, it requires modelers with expertise in TRIO language to express predicates/axioms as well as to conduct the analysis of the TRIO models *(C17*).

4.5.3. Reliability and safety. Reliability and safety (*C8*) topics are jointly treated by [Mustafiz et al. 2008; Mustafiz and Kienzle 2009], [Zarras et al. 2004] and [Jürjens 2003; Jürjens and Wagner 2005]. The three approaches provide support in different phases of the software life-cycle, that is respectively, requirement, software architecture and design (*C1*).

[Mustafiz et al. 2008; Mustafiz and Kienzle 2009] devise a requirement engineering process (DREP) for the elicitation, specification and analysis of reliability and safety requirements. They extend use cases (*C2*,*C3*) to discover exceptional situations that can interrupt the system normal behavior and to define derived requirements to handle such situations. Use cases are mapped to DA-Charts (*C6*), a type of state-charts where probabilities are associated to success/failure transitions (*C11* - **DT.FailB, DM.R**). A Markov chain is then constructed from a DA-Chart (*C10*) to compute the reliability on demand and the probability of reaching safe states from the initial system state (*C12* - **DM.R, DM.S**). The dependability analysis produces information to the designer on the maximal achievable reliability and safety (*C13* - **DR.BOUND**), considering only the failures of the system environment (e.g., hw sensor failures) and assuming the system under development be reliable. The applicability and effectiveness of the DREP approach has been evaluated empirically, in academic environment, using an electronic toll collection system as case study (*C14*).

[Zarras et al. 2004] address mainly reliability and safety analysis of composite web services (*C4*); availability is also dealt, but as secondary dependability property. They consider BPEL [BPEL 2007] as software architecture specification language and propose a UML representation of BPEL constructs through stereotypes. UML extensions are also defined to express the parameters necessary for dependability analysis (*C6*). In particular, they include: reliability, safety and availability measures (*C12* - **DM.R, DM.S, DM.A**) to be computed, the fault characterization of objects (*C11* - **DT.FP, DT.FO, DT.FOQ**) - e.g., fault rate, fault persistency, phase of occurrence, boundary and nature - the failure domain and consistency (*C11* - **DT.FMD, DT.FMC**), and redundancy schema within the devised FT techniques (*C11* - **R.L, R.F, R.R**) - e.g., the type of adjudicators in error-detection mechanisms, the redundancy level and FT level of a redundant schema. The UML annotated models are then transformed into Block Diagrams and Markov models which enable the dependability evaluation of the composite web services (*C10*).

Table XVI. RAMS.

Works	Criteria
[Bernardi et al. 2009]	C1, C6, C8, C11, C13

[Jürjens 2003; Jürjens and Wagner 2005] define safety and reliability checklists, using UML stereotypes and tags (*C6*), to support the analyst in the identification of failure-prone components (*C12* **- DM.S, DM.C**) in the software design. The UML extensions are used to specify requirements on communication - e.g., maximum probability of message loss, safety/reliability level (*C13* **- DR.BOUND, DR.S**) - and failure assumptions (*C11* - **DM.R**) of communication links/nodes as a function of the failure domain (*C11* - **DT.FMD**) and of the type of voters within redundancy structures (*C11* - **R.R**). A precise semantics is provided to check the design, via temporal logic formulas (*C10*), against the requirements and constraints specified with the proposed UML extensions.

4.5.4. Reliability, availability, maintainability and safety. [Bernardi et al. 2009] propose a UML profile (namely DAM, *C6*), as a specialization of the OMG standard [UML-MARTE 2009], to support the dependability analysis of UML-based software systems, in the early phases of the software life-cycle (i.e., requirement, software architecture and design, *C1*). In particular, the DAM profile focuses on the RAMS properties (*C8*) and its definition was based on a thorough analysis of different approaches, included in the present survey, for dependability specification and analysis within UML. The main objective of the work has been to unify the terminology and concepts for different dependability aspects *C11-13* - (**DR.BOUND, DM, DT, FT.R, M.R, R.L, R.L, R.R**) under a common consistent dependability domain model, reusing the best practices and choices reported in literature on model transformation to generate formal dependability analysis models.

5. DISCUSSION

In this section we analyze, from a critical perspective, the set of contributions presented in Section 4. We use the criteria presented in Section 3 as guidelines for our discussion. Some considerations about the fulfillment of the dependability concepts from the checklist (Table I) by the surveyed works will be also provided. In Table XVII we have summarized each approach and labeled them with an identifier that will be used throughout this section. The Table is arranged according to the order of presentation of the approaches in Section 4 and the last column indicates the concrete subsection where the approach is discussed.

5.1. Software engineering and UML criteria

5.1.1. Life-cycle phase. The support provided by the approaches within the software lifecycle spans from the requirement to the deployment phases (Figure 2, Table XVIII) although major contributions are given in the early phases, in particular during the requirement, software architecture and design specification. This is an expected result, as occurred for performance model-based approaches [Balsamo et al. 2004], since the major modeling effort is placed early in the life-cycle, where the detection of both functional and non functional (e.g., dependability, performance) problems is more effective from the software costs point of view.

We observed that a significant number of works aim at providing approaches to the dependability analysis of software systems, while few contributions address requirement elicitation or just dependability specification (i.e., how to express dependability characteristics in the software specification). Among the surveyed works, only Goseva and Hassan provide methods for the risk assessment of safety-critical systems.

Table XVII. List of surveyed approaches

Approach ID	Authors	Papers	Section
D'Ambrogio	D'Ambrogio A., Iazeolla G., Mirandola R.	[D'Ambrogio et al. 2002]	§4.1.1,§4.1.3
Yacoub	Yacoub S.M., Cukic B., Ammar H.H.	[Yacoub et al. 2004]	§4.1.1,§4.1.3
Rodrigues	Rodrigues G.N., Rosemblum D.S., Uchitel S.	[Rodrigues et al. 2005]	§4.1.1,§4.1.3
Singh	Singh H., Cortellessa V., Cukic B., Gunel E., Bharadwaj V.	[Singh et al. 2001; Cortellessa et al. 2002]	§4.1.2,§4.1.3
Pai	Pai G.J., Dugan J.	[Pai and Dugan 2002]	§4.1.2
Grassi	Grassi V., Mirandola R., Sabetta A.	[Grassi et al. 2005; 2007]	§4.1.2,§4.1.3
David	David P., Idiasak V., Kratz F.	[David et al. 2009]	§4.1.2
Cortellessa	Cortellessa V., Pompei A.	[Cortellessa and Pompei 2004]	§4.1.3
Bernardi-a	Bernardi S., Merseguer J.	[Bernardi and Merseguer 2006]	§4.2
Genero	Genero M., Piattini M., Manso E., Cantone G., Visaggio A., Canofra G.	[Genero et al. 2003; Genero et al. 2007]	§4.3
Allenbi	Allenbi K., Kelly T.	[Allenby and Kelly 2001]	§4.4.1
Johannessen	Johannessen P., Grante C., Alminger A. Eklund U., Torin J.	[Johannessen et al. 2001]	§4.4.1
Hansen	Hansen K., Wells L., Maier T.	[Hansen et al. 2004]	§4.4.2
Iwu	Iwu F., Galloway A., McDermid J., Toyn I.	[Iwu et al. 2007]	§4.4.2
Liu	Liu J., Dehlinger J., Lutz R.R.	[Liu et al. 2007]	§4.4.2
Hawkins	Hawkings R., Toyn I, Bate I.	[Hawkings et al. 2003]	§4.4.3
Pataricza	Pataricza A., Majzik I., Huszerl G., V‘arnay G.	[Pataricza et al. 2003]	§4.4.3
Ober	Ober I., Graf S., Ober I.	[Ober et al. 2006]	§4.4.3
Goseva	Goseva-Popstojanova K., Hassan A., Guedem A., Abdelmoez W., Nassar D.E.M., Ammar H., Mili A.	[Goseva-Popstojanova et al. 2003]	§4.4.5
Hassan	Hassan A., Goseva-Popstojanova K., Ammar H.	[Hassan et al. 2005]	§4.4.5
Cancila	Cancila D., Terrier F., Belmonte F., Dubois H., Espinoza, H., Grard S., Cuccuru A.	[Cancila et al. 2009]	§4.4.4
Zoughbi	Zoughbi G., Briand L., Labiche Y.	[Zoughbi et al. 2007; 2006]	§4.4.3
Lu	Lu S., Halang W.	[Lu and Halang 2007]	§4.4.4
Bondavalli	Bondavalli A., Dal Cin M., Latella D., Majzik I., Pataricza A., Savoia G.,	[Bondavalli et al. 2001; Majzik et al. 2003; Pataricza 2000]	§4.5.1
DeMiguel	DeMiguel M., Lambolais T., Piekarec S., Betgé-Brezetz S., Péquery J.	[DeMiguel et al. 2001]	§4.5.1
Leangsuksun	Leangsuksun C., Song H., Shen L.	[Leangsuksun et al. 2003]	§4.5.1
DalCin	Dal Cin M.	[Dal Cin 2003]	§4.5.1
Addouche	Addouche N., Antoine C., Montmain J.	[Addouche and Antoine 2004; Addouche et al. 2006]	§4.5.2
Bernardi-b	Bernardi S., Donatelli D. Dondossola G.	[Bernardi et al. 2004a; 2004b]	§4.5.2
Mustafiz	Mustafiz S., Sun X., Kienzle J., Vangheluwe H.	[Mustafiz et al. 2008; Mustafiz and Kienzle 2009]	§4.5.3
Zarras	Zarras A., Vassiliadis P., Issarny V.	[Zarras et al. 2004]	§4.5.3
Jüriens	Jüriens J., Wagner S.	[Jürjens 2003; Jürjens and Wagner 2005]	§4.5.3
Bernardi-c	Bernardi S., Merseguer J., Petriu D.C.	[Bernardi et al. 2009]	§4.5.4

Table XVIII. Contributions by life-cycle phase.

Approach ID	Requirements	Architecture	Design	Impl./Deployment
D'Ambrogio		✓		
Yacoub		✓		
Rodrigues		✓		
Singh			✓	
Pai			✓	
Grassi			✓	
David			✓	
Cortellessa		✓	✓	
Bernardi-a			✓	
Genero			✓	
Allenbi	✓			
Johannessen	✓			
Hansen		✓		
Iwu	✓	✓	✓	
Liu		✓		
Hawkins	✓		✓	
Pataricza			✓	
Ober			✓	
Goseva		✓		
Hassan		✓		
Cancila		✓	✓	
Zoughbi			✓	
Lu			✓	✓
Bondavalli		✓		
DeMiguel		✓	✓	
Leangsuksun			✓	✓
DalCin	✓		✓	
Addouche	✓		✓	
Bernardi-b	✓		✓	
Mustafiz	✓			
Zarras		✓		
Jüriens			✓	
Bernardi-c	✓	✓	✓	

The implementation and deployment stages are addressed by only one contribution each (Lu and Leangsuksun, respectively), while none of the considered works focus on the testing activities. We think that research efforts should be devoted to combine model-based approaches with experimental ones in the testing phase, e.g., by exploiting use-case to drive the testing activities through test cases and to trace back the latter to dependability requirements.

5.1.2. Diagrams. Concerning the UML specifications assumed as input to the method, class and deployment are the mostly used structural diagrams (Table XIX). Unlike in performance analysis of UML-based systems, where UML behavioral specifications are necessary to get a performance model, dependability models can also be derived from only structural specifications.

At the first sight, it seems unusual that deployment diagrams are used in works which address the early phases of the software life-cycle. This can be justified, considering that dependability issues can arise not only from software faults but also from hardware ones (e.g., node crashes, broken communication physical links). Therefore, dependability requirements for a software system need to address not only the software, but also the platform dependent architectures of the entire system (usually modeled by deployment diagrams).

Use case, sequence and state machines are the typically assumed behavioral diagrams. In particular, use case diagrams are used not only in requirement elicitation

Fig. 2. Contribution to life-cycle phases.

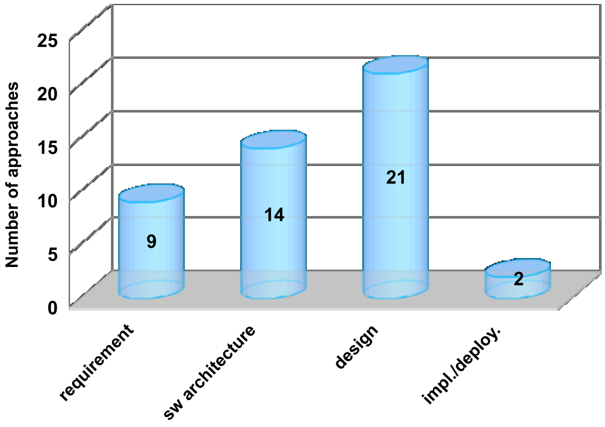
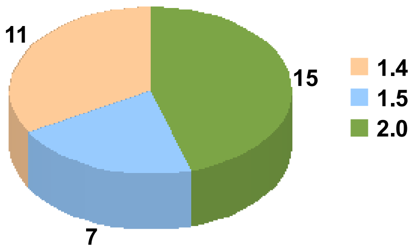


Table XIX. UML diagrams.

Approach ID	Class	Object	UC	SM	Act.	Seq.	IOD	Col.	Comm.	Deploy.
D'Ambrogio						✓				✓
Yacoub						✓				
Rodrigues						✓	✓			
Singh			✓			✓				✓
Pai	✓	✓		✓						✓
Grassi					✓	✓		✓	✓	✓
David						✓				
Cortellessa			✓			✓		✓	✓	✓
Bernardi-a				✓		✓				
Genero	✓									
Allenbi			✓							
Johannessen			✓							
Hansen	✓	✓				✓			✓	✓
Iwu	✓		✓			✓				
Liu				✓		✓			✓	
Hawkins	✓			✓		✓				
Pataricza				✓						
Ober	✓			✓						
Goseva			✓	✓		✓			✓	
Hassan			✓			✓				
Cancila	✓		✓							
Zoughbi	✓									
Lu									✓	✓
Bondavalli	✓	✓	✓	✓		✓		✓		✓
DeMiguel	✓			✓		✓		✓		✓
Leangsuksun										✓
DalCin	✓		✓					✓	✓	
Addouche	✓			✓				✓		
Bernardi-b	✓									
Mustafiz			✓		✓					
Zarras		✓			✓					✓
Jüriens				✓	✓	✓				✓
Bernardi-c	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Total	14	5	12	12	5	18	2	7	8	13

Fig. 3. UML versions.



approaches, but also in the works addressing dependability analysis, mainly, to specify the operational profile (e.g., Singh and Cortellessa).

Observe that, apart from Bernardi-c, which provides support - through a profile - to dependability specification (only) for all UML diagrams, all the other surveyed works consider only a sub-set of UML diagrams. Note that Bondavalli is the contribution that enables to analyze dependability based on the largest sub-set of UML diagrams.

The surveyed works rely upon different UML versions (i.e., 1.4, 1.5 and 2.0), mainly according to the year of publication (Figure 3). In general, the most important changes between UML 1.x and 2.x concern behavioral diagrams, specifically activity and sequence. However, none of the surveyed approaches that rely upon UML 1.x use activity diagrams. The ones that use sequence diagrams (D'Ambrogio, Yacoub, Singh, Hansen, Goseva, Bondavalli and De Miguel) can, in principle, be applied to UML 2.x, since they consider independent execution scenarios - modelled each one by a simple sequential SD (i.e., without alternative, parallel, optional sub-scenarios). Obviously, the software tools that support such approaches and rely on the UML meta-model (e.g., to produce automatically a formal dependability model) would need to be upgraded to the new UML version. UML2.0 supports more types of diagrams than UML1.*, in particular the interaction overview diagrams (IOD), which are a combination of activity and sequence diagrams. IOD allow one to model system scenarios using a hierarchical approach. Nevertheless, even though the majority of the surveyed works support UML2.0, only Rodrigues and Bernardi-c use IOD.

5.1.3. Software development process. Most of the approaches follow the traditional software life-cycle (Table XX). The use-case approach is applied in some works to capture dependability requirements, besides the functional ones.

We can observe that there are several contributions using the component-based software development process. However, only one work, Liu, addresses the software product-line development process. Note also that a few surveyed works apply model-driven development techniques, where software models are the main focus of the development. The use of model transformations to generate not only code, but also analysis models is an intrinsic part of model-driven development.

5.1.4. Software and Application domains. As shown in Figure 4, most of the works either do not focus on a specific software domain or provide specific support to real-time (embedded) systems. Only Zarras addresses the SOA domain. The majority of the surveyed works support reliability analysis of general software systems, possibly fault-tolerant and distributed. We observe that the kind of dependability property to be evaluated is influenced by the software and the application domains considered by a given work. For instance, contributions focusing on real-time (embedded) systems are mainly concerned with safety issues. In particular, considering in detail the application domain

Table XX. Contributions by software development process.

Approach ID	general	UC	CBSE	SPL	MDD
DAmbrogio			✓		
Yacoub			✓		
Rodrigues			✓		✓
Singh			✓		
Pai	✓				
Grassi			✓		✓
David	✓				
Cortellessa			✓		
Bernardi-a	✓				
Genero	✓				
Allenbi		✓			
Johannessen		✓			
Hansen	✓				
Iwu	✓				
Liu			✓	✓	
Hawkins	✓				
Pataricza	✓				
Ober	✓				
Goseva		✓	✓		
Hassan		✓			
Cancila	✓				✓
Zoughbi	✓				✓
Lu	✓				
Bondavalli	✓				✓
Demiguel	✓				
Leangsuksun	✓				
DalCin	✓				
Addouche	✓				
Bernardi-b	✓				
Mustafiz		✓			
Zarras	✓				
Jüriens			✓		
Bernardi-c	✓				✓
Total	20	5	9	1	6

Fig. 4. Software domain

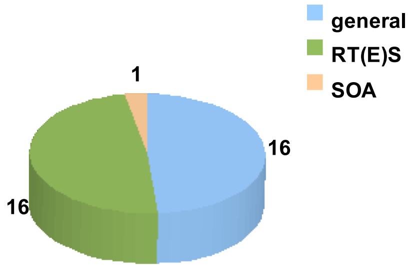


Table XXI.

Approach ID	general	aerospace	automotive	railway	automated	healthcare	transaction
D'Ambrogio	✓						
Yacoub	✓						
Rodrigues	✓						
Singh							✓
Pai		✓					
Grassi	✓						
David	✓						
Cortellessa	✓						
Bernardi-a	✓						
Genero	✓						
Allenbi		✓					
Johannessen			✓				
Hansen	✓						
Iwu		✓					
Liu						✓	
Hawkins	✓						
Pataricza				✓			
Ober		✓					
Goseva						✓	
Hassan						✓	
Cancila				✓			
Zoughbi		✓					
Lu	✓						
Bondavalli	✓						
DeMiguel	✓						
Leangsuksun	✓						
DalCin	✓						
Addouche					✓		
Bernardi-b					✓		
Mustafiz	✓						
Zarras							✓
Jüriens			✓				
Bernardi-c	✓						
Total	17	5	2	2	2	3	2

(Table XXI), we notice that most of the works that address aereospace, automotive, railways control software and healthcare systems are interested in providing support for safety analysis. On the other hand, in the case of transaction applications it is often desirable to guarantee the continuity and the promptness of service delivery, when requested by the end-user. Therefore, reliability and availability are the main issues addressed by the works dealing with this type of applications (Singh, Zarras).

5.1.5. Dependability specification. The specification of dependability requirements and properties can be done by a) providing a specific UML profile; b) a set of UML standard extensions (that is stereotypes and tagged values), not structured in a profile[4]; c) using regular non-extended UML models and d) using OCL (Figure 5). In particular, when regular models are used, they are the same UML diagrams - like use cases applied in the requirement elicitation approaches - or ad hoc ones - like state-machine variants (Mustafiz, Iwu). Other approaches use UML models to extrapolate dependability informations (Genero, Hansen, Goseva). Finally, only Hawkins and Lu use the Object Constraint Language (OCL) to specify safety related requirements and constraints.

Table XXII details the type of specification used by each considered approach. The definition of a UML profile requires more effort with respect to propose a set of extensions, but has the advantage of defining consistent extensions in a structured framework. The majority of the approaches that resort to profiling technique, define the profile in the context of existing standard OMG UML profiles, such as the Schedulability, Performance and Time [UML-SPT 2005] (Rodrigues, Grassi, Cortellessa, Bondavalli, Addouche) and the Modeling and Analysis of Real-Time Embedded System [UML-MARTE 2009] (Cancila, Bernardi-c), which has the advantage of exploiting the specification capabilities of the standard profile. Although a lot of efforts has been devoted to propose UML extensions to support dependability specification in UML-based systems, less attention has been paid to providing a solution for the unification of the different proposals. Indeed, to the best of our knowledge, only Bernardi-c tackled this

[4] Observe that the type of specification *b)* refers to the surveyed works that rely upon UML 1.*.

Fig. 5. Type of specification.

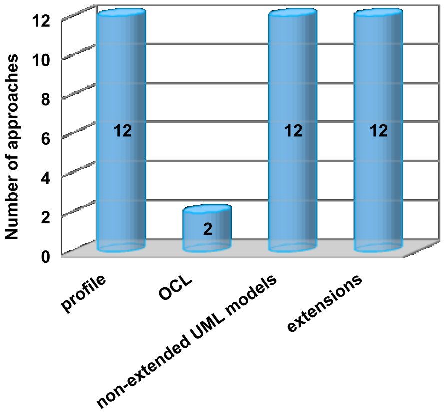


Table XXII.

Approach ID	profile	OCL	non-ext. UML	extensions
D'Ambrogio			✓	
Yacoub				✓
Rodrigues	✓			
Singh				✓
Pai				✓
Grassi	✓			
David			✓	
Cortellessa	✓			
Bernardi-a	✓			✓
Genero			✓	
Allenbi			✓	
Johannessen			✓	
Hansen			✓	
Iwu			✓	
Liu			✓	
Hawkins		✓		
Pataricza				✓
Ober	✓			
Goseva			✓	
Hassan				✓
Cancila	✓			
Zoughbi	✓			
Lu	✓	✓		
Bondavalli	✓			✓
DeMiguel				✓
Leangsuksun				✓
DalCin	✓			
Addouche	✓		✓	
Bernardi-b			✓	
Mustafiz			✓	✓
Zarras				✓
Jüriens				✓
Bernardi-c	✓			

issue. Currently, a standard OMG proposal for a dependability profile does not exist yet. We think that more research should be invested in providing a common UML framework for the modeling and analysis of different NFPs in order to support the consistent specification of different NFPs and their relationships, as well as the trade-off analysis between different NFPs (such as performability, performance and security, security and dependability).

5.1.6. Tool support. As shown in Figure 6, the majority of works provides tool support for the approaches they propose. Although most of the tools are research prototypes that do not cover all the aspects, the potential for building more powerful tool support exists. Many approaches could be automated since they propose either rigorous transformation techniques of UML annotated models into formal dependability mod-

Fig. 6. Tool support

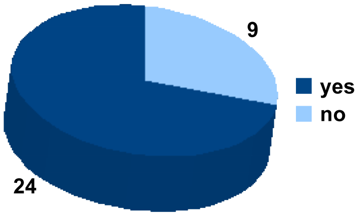
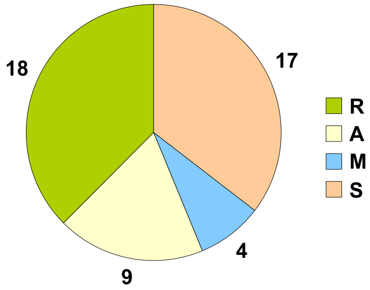


Fig. 7. Dependability attribute



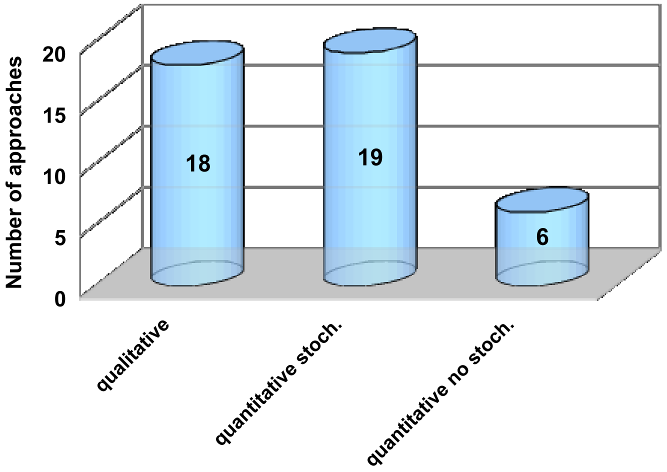
els or dependability annotations through the UML profiling mechanism. Only a few proposals are difficult to implement or do not provide any indication of an existing implementation. These are mainly approaches that address dependability requirements elicitation via use cases (Allenby and Johannessen) or class diagrams (Bernardi-b) or focus on severity analysis (Hassan).

5.2. Dependability characteristics criteria

5.2.1. Attribute. Concerning the type of dependability attribute, most of the surveyed approaches address either reliability or safety issues, while few efforts have been devoted to maintainability and availability (Figure 7). Indeed, the latter are often considered as secondary dependability issues. In particular, the stochastic approaches proposed for reliability analysis can be also used, as claimed by their authors, to compute availability and maintainability measures (e.g., steady state availability, MTTR) given that the additional quantitative characterization of the repair or recovery activities is provided (e.g., repair rate) as input parameter. A unique exception is Genero where a set of size and complexity metrics for UML class diagrams are proposed as indicators of the software specification maintainability.

5.2.2. Analysis type. One of the considered criteria is the type of dependability analysis proposed, that is qualitative or quantitative (Figure 8). Qualitative analysis aims to identify, classify and rank the hazards or failure modes in the software systems, while quantitative analysis mainly aims to compute dependability measures. We notice that safety-related contributions fall basically in the first category (i.e., qualitative) while the works that focus on reliability, maintainability and availability belong to the sec-

Fig. 8. Analysis type



ond one (i.e., quantitative). There are some exceptions that support both types of analysis, like the works on safety of Ober, Goseva, Hassan and Cancila, and the works of Bernardi-b, DalCin, Jüriens and Bernardi-c, providing support for dependability specification.

Considering the approaches aimed at quantitative dependability analysis, the majority of them rely on stochastic (or probabilistic) assumptions. Nevertheless, there are also non-stochastic approaches to the dependability analysis, like Ober and Bernardi-b, that support the verification of time-dependent dependability requirements of real-time systems.

5.2.3. Analysis model. Table XXIII summarizes the techniques adopted by the surveyed works to support dependability analysis of UML-based specifications. This criterion does not apply to the approaches that provide support only for dependability specification (shown in grey in Table XXIII).

Some of the used techniques are those suggested by dependability standards, such as FMEA, HAZOP, Petri Net, Fault Tree, Markov model, Bayesian model and Block diagram [IEC-60300-3-1 2003]. In particular, Fault tree and its variants (e.g., dynamic fault tree) is the mostly used dependability technique, followed by Markov models. Fault trees have been applied in both reliability (D'Ambrogio, Pai, Leangsuksun) and safety works (Iwu, Liu , Hawkins, Hassan), and for both qualitative (Iwu, Liu, Hawkins, Hassan) and quantitative (D'Ambrogio, Pai, Leangsuksun) analysis.

Some contributions propose instead techniques which are not traditionally aimed at dependability analysis, e.g., component dependency graphs (Yacoub). Finally, there are a few approaches, like Hassan and Iwu, that suggest the combined use of several complementary techniques.

5.2.4. Parameters and requirements. Figure 9 shows three histograms that represent the number of surveyed approaches addressing the items of the checklist in Table I, namely the dependability requirements (upper-left histogram), the dependability measures (upper-right histogram) and the other dependability parameters (bottom histogram). We observe that the specification of dependability requirements is supported by few approaches. In particular, most of the works that aim at evaluating the sys-

Table XXIII. Dependability analysis models

Approach	Hazard analysis	FMEA	HAZOP	FFA	Stochastic Petri net	Labeled trans. system	Formal state machine	Dataflow network	Simulation model	Fault Tree	Cost of failure graph	Comp. Dep. Graph	Markov model	Bayesian model	Prob. timed automata	Timed automata	Temporal logics	Block diagram
D'Ambrogio										✓								
Yacoub												✓						
Rodrigues						✓							✓					
Singh														✓				
Pai										✓								
Grassi													✓					
David		✓																
Cortellessa																		
Bernardi-a					✓													
Genero																		
Allenbi			✓															
Johannessen				✓														
Hansen			✓															
Iwu			✓				✓			✓								
Liu										✓								
Hawkins										✓								
Pataricza							✓	✓										
Ober																✓	✓	
Goseva	✓	✓											✓					
Hassan		✓		✓						✓	✓							
Cancila																		
Zoughbi																		
Lu																		
Bondavalli					✓													
DeMiguel									✓									
Leangsuksun										✓			✓					
DalCin					✓													
Addouche															✓			
Bernardi-b																	✓	
Mustafiz													✓					
Zarras													✓					✓
Jüriens																	✓	
Bernardi-c																		

tem reliability do not provide support for the validation of the estimated reliability measures w.r.t. the requirements.

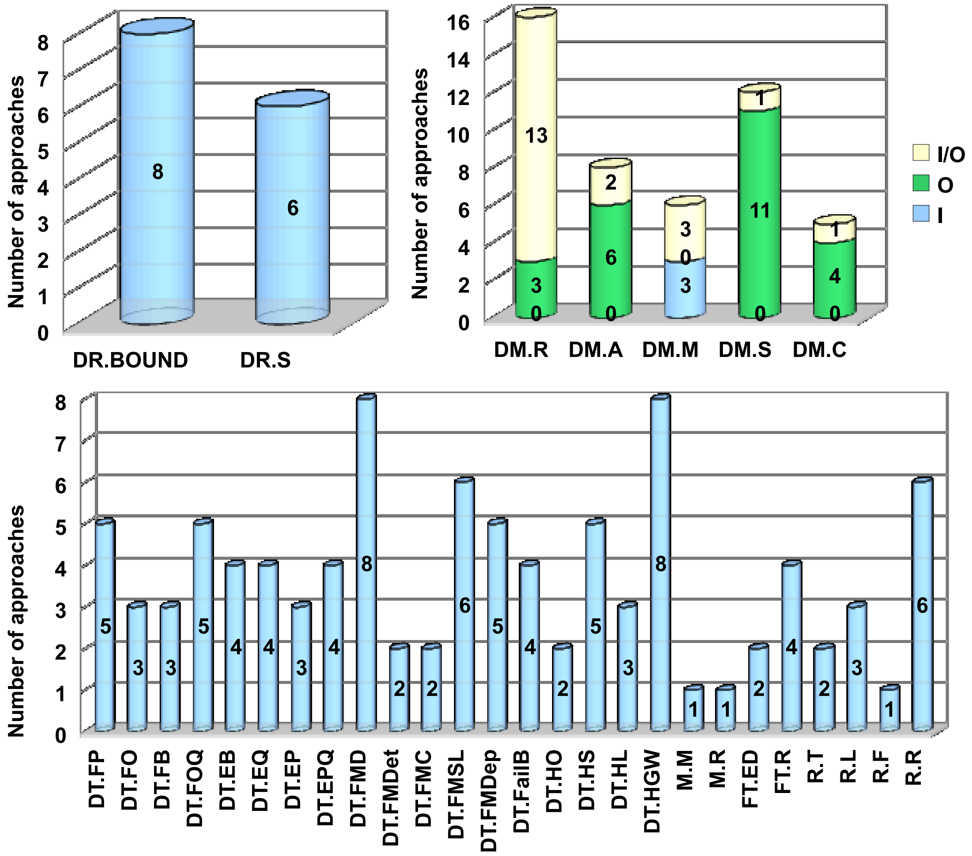
Concerning the dependability measures, they are often considered as both input parameters and output results in a given approach (D'Ambrogio, Yacoub, Rodrigues, Singh, Pai, Grassi, Cortellessa, DeMiguel, Leangsuksun, Addouche, Bernardi-b, Mustafiz, Bernardi-c). For example, the failure occurrence rate is associated to software component/connectors (i.e., input parameters for the method) and to the system level as well (i.e., output result provided by the method).

None of the surveyed approaches provide any indication on how to assign values to the input parameters. Input parameters are simply assumed values. The value assignment can be trivial for some input parameters, such as MTTF of hardware components that is usually provided by the manufacturer, however this is not the case for most of the parameters (e.g., how to assign a MTTF value to a software component ?).

The most frequent items are reliability measures (**DM.R**) and safety properties (**DM.S**): this is not surprising, since most of the surveyed works address reliability and safety issues.

Considering the other dependability parameters, although each item is addressed by at least a work, several of them are marginally dealt with. In particular, special attention has been devoted to the specification of failure modes with respect to the domain (**DT.FMD**) and to the use of hazard guide-words (**DT.HGW**), while few works consider, other classifications of failure modes, such as failure detectability (**DT.FMDet**) and consistency (**DT.FMC**). On the other hand, few efforts have been devoted to maintenance issues, i.e., modifications (**M.M** - Genero) and repair (**M.R** - Bernardi-c), and to supporting a comprehensive specification of redundancy in fault-tolerant systems.

Fig. 9. Number of approaches addressing each checklist item.



For instance, only Zarras provides UML extensions to specify the maximum number of replica failures that can be tolerated (**R.F**).

Figure 10 shows the number of checklist items addressed by each surveyed approach. Such a number is a raw quality metric for the evaluation of the approach itself, giving some insight of its comprehensiveness in providing support for dependability modeling and/or analysis. Obviously, such a metric should not be considered in isolation since other aspects are important as well, as discussed in the next subsection. Note that Bernardi-c is the approach that considers most of the items, since it actually builds on several approaches considered in the survey. Nevertheless, it does not provide a specific method to analyze the system dependability but, rather, supports the dependability specification through a UML profile. On the other hand, Bondavalli is the second approach in the checklist coverage; unlike Bernardi-c, it also proposes a method to derive formal models amenable for dependability analysis.

5.3. Quality criteria

5.3.1. Validation and Compliance with standards. The method validation is not a primary issue (Figure 11(a)). Indeed 12 out of 33 of the surveyed approaches do not consider it at all. On the other hand, when validation is a concern, it is carried out mainly to show the applicability and/or the scalability of the method to realistic examples, i.e.,

Fig. 10. Number of checklist items addressed by each approach

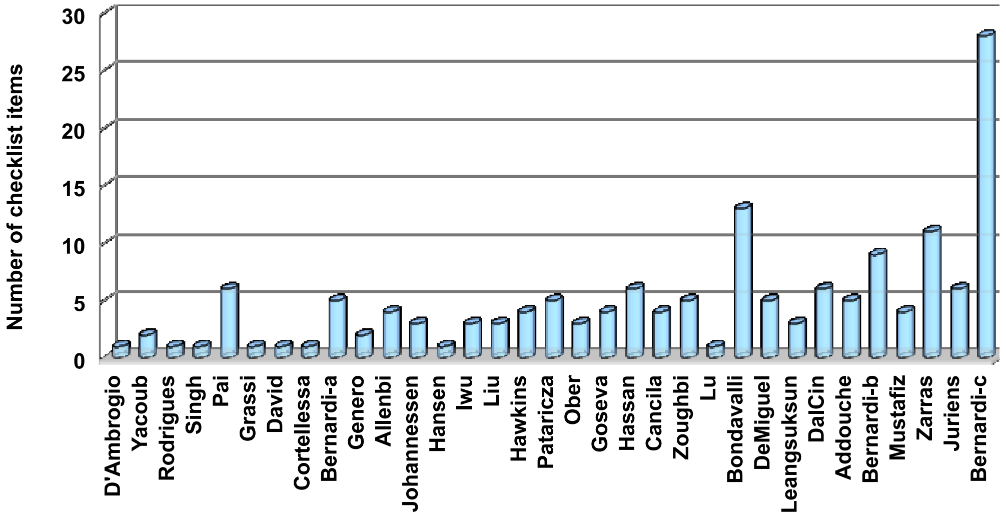
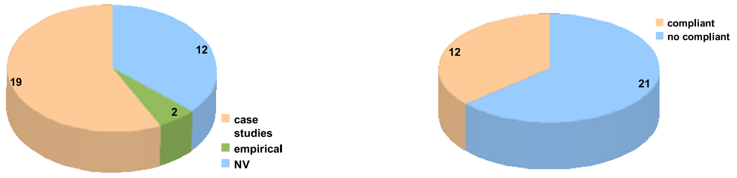


Fig. 11. (a) Validation. (b) Compliance with standards.



(a) (b)

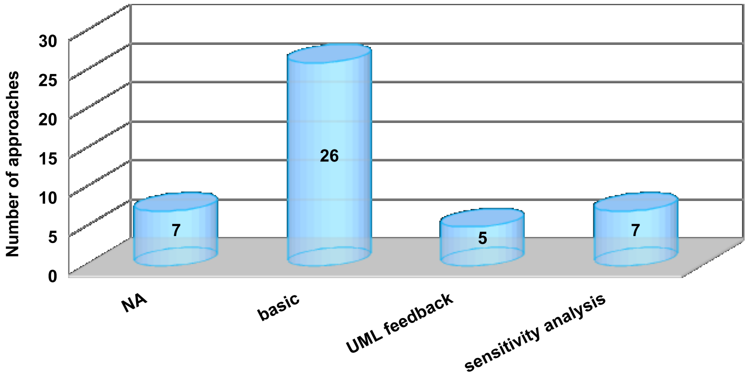
through case studies. The 19 approaches that address a case study are: Singh, Pai, David, Bernardi-a, Allenbi, Johannessen, Hansen, Iwu, Liu, Patarizca, Ober, Goseva, Hassan, Zoughbi, Bondavalli, Bernardi-b, Zarras, Jüriens and Bernardi-c. Only few works (Genero and Mustafiz) conduct empirical analysis in an academic environment, to assess the effectiveness of the proposed approaches beside their applicability.

We observed that all approaches providing support for quantitative dependability analysis are in fact missing the validation of the correctness of the proposed methods. This could be achieved, for example, by comparing the analysis results with the ones obtained in testing activities by injecting faults during the system execution.

Concerning the compliance of the method with respect to dependability engineering standards, only 12 out of 33 approaches adheres to some standard, as shown in Figure 11(b). Most of the compliant approaches focus on safety issues in the development of real-time and embedded systems (Allenbi, Johannessen, Iwu, Goseva, Hassan, Cancila, Zoughbi, Lu) for which a certification from third parties is required.

5.3.2. Presentation of results. The majority of the approaches addressing dependability analysis provide a basic support to present the results of the analysis (Figure 12). The most common way is textual presentation, followed by graphical and tabular one. We observed that sensitivity analysis is supported by several works that address reliabil-

Fig. 12. Presentation of the results.



ity and availability analysis. The most promising approaches are those that feedback the results to the original UML specification (Rodriguez, Liu, Pataricza, Hassan and Jüriens); this makes the analysis process transparent to the software analyst.

However, further research is needed to address this issue. In particular, the problem of how to identify the critical elements of the UML specification that cause dependability properties not to be satisfied is still open. Good solutions are those that provide useful information to the software engineers for changing the design accordingly. Finally, in Figure 12, there are seven approaches classified as not available (i.e., NA): they are either aimed at dependability specification, rather than the analysis, or provide transformation techniques without focusing on the analysis of the derived formal models.

5.3.3. Limitations. Almost all the surveyed approaches present limitations, as summarized in detail in Table XXIV. In particular, several proposals aimed at reliability analysis assume failure independence of system components (Yacoub, Rodrigues, Singh, Grassi, Zarras, Leangsuksun). However, this assumption, which facilitates the analysis of the derived reliability model, may not hold for systems characterized by tightly coupled components. An example of such systems are the even demanding complex, large scale ICT infrastructures that control distributed embedded systems (e.g., distributed SCADA systems controlling power production and distribution plants located in a given geographical area). Another limitation that is common to some approaches is low scalability (Bernardi-a, Hansen, Hawkins) that can make the validation and verification activities time consuming and risky from the point of view of software development process management or, even worse, unfeasible.

6. CONCLUSION

We have surveyed approaches in the literature addressing dependability modeling and analysis of software systems specified with UML. The survey covers contributions published in the last decade that focus on different facets of dependability, namely reliability, availability, maintainability and safety. Several open research issues emerged from the study. Firstly, most of the works focus on reliability and safety and fewer efforts have been devoted to availability and maintainability modelling and analysis. Moreover, we have not found any work addressing specifically how to extend UML with integrity NFP, which is also a dependability concern.

Table XXIV. Limitations

Approach ID	Limitations
D'Ambrogio	Lack of UML extensions. Informal treatment of spatial redundancy.
Yacoub	Execution scenario independent assumption. Parallel execution of components is not supported. Component failure independence assumption
Rodrigues	Component failure independence assumption.
Singh	Component/connector failure independence assumption. Time-independent failure probability.
Pai	Use of class diagrams to represent hardware components and explicit error propagation associations between hardware components.
Grassi	Failure independence assumption.
David	A Dysfunctional Behavior dababase needs to be constructed.
Cortellessa	No annotations for hw failure supported.
Bernardi-a	Limited scalability that may lead to the generation of intractable dependability models from the analysis point of view.
Genero	Lack of guidelines about how to use the proposed metrics to evaluate the maintainability of the UML specifications.
Allenbi	Operations in emergency/degraded states and multiple failure identification are not supported.
Johannessen	
Hansen	Limited scalability that may lead to a time consuming activity.
Iwu	Lack of relationships between UML specification and PFS requirements.
Liu	The state-based modeling technique is not suitable for testing border (exact) time values.
Hawkins	Limited scalability in the hazard detection approach that may lead to an uncontrolled generation of mutant transitions.
Pataricza	
Ober	
Goseva	Use case/scenario independent assumption.
Hassan	Low traceability of the results derived from each applied severity technique.
Cancila	
Zoughbi	Use of dynamic concepts (defined with the profile) to extend static concepts. This leads to mixed static/dynamic views in the same diagram (CD).
Lu	Timing specification issues are vaguely dealt.
Bondavalli	Introduction on unnecessary redundant information in the UML models, since some input parameters require the joint use of more than one stereotype.
DeMiguel	
Leangsuksun	Node failure independence assumption, single-failure assumption.
DalCin	Lack of support to the modeling of the interaction among dependability mechanisms and the system components.
Addouche	Poor separation of concerns (new classes need to be defined and introduced in the system model, beside the classes representing the system components).
Bernardi-b	Expertise of the modeler required to specify the predicates/axioms in TRIO language.
Mustafiz	Failure assumptions limited to failures coming from the system environment (hw sensor failure). Use of no standard state-charts (DA-Chart).
Zarras	Object failure independence assumption.
Jüriens	
Bernardi-c	Lack of support for the specification of path properties. Limited support for the specification of FT mechanism (only redundancy aspects are dealt).

Secondly, the surveyed works provide support mainly in the early phases of the software life-cycle (i.e., from requirement to design), while there is a lack of support for later phases, as for example for testing dependability NFPs guided by the use cases.

Thirdly, those contributions that support model transformation mainly focus on obtaining formal models which are amenable for dependability analysis. However, only a few go one step further to provide a feedback from the analysis results to the original UML model specification, in order to pinpoint to requirement inconsistencies or design flaws. It is also worth noticing that tool support and method validation are crucial factors to make an approach effectively applicable. Although the majority of the surveyed approaches are characterized by a high automation degree, most of them are not fully supported by a software tool. Moreover, in many cases method validation consists only in applying the proposed method to a case study. Considering the approaches that provide support for quantitative dependability analysis, the validation of the correctness of the proposed methods is in fact missing. More efforts should be devoted to the validation of the methods themselves.

Last but not least, more research work should be invested in providing a standard common UML framework for the modelling and analysis of several NFPs, in order to support the consistent specification of different NFPs and their relationships, as well as the trade-off analysis between different NFPs (such as performability, performance and security, security and dependability).

Finally, an important issue to consider for the future work is to investigate the current request for proposals (RFPs) for UML that are now being considered and their potential impact to the area of dependability modeling with UML.

APPENDIX

The Unified Modeling Language [UML 2005] is a general purpose standardized modeling language used for software development. It proposes a set of diagrams that allow description of the structural and behavioral views of a system as well as the hardware platform where the same is deployed. UML has an extensive tool support.

The system structure may be described in UML by a component diagram and/or a class diagram. The first contains components and connectors that can be packaged or grouped to form subsystems, which in turn are grouped to form higher level subsystems and eventually a system. Components can be logical or physical. A component provides services offered through its interfaces and may require services from other components. A class diagram allows for representing in detail the internal structure of the system components. The behavioral view of the system is specified using use cases, activity diagrams, sequence diagrams and state machines, or a combination of them. Section A.1 offers some examples of UML diagrams and a brief explanation to understand their basic features.

UML 2 introduced the profile mechanism as a meta-modeling technique to extend and adapt the language for different purposes. Reasons to extend UML are of different nature, for example to introduce terminology adapted to a given platform or domain (e.g., the EJB profile [UML-EDOC 2001]) or to add semantics not already present in UML that can be used for model transformation purposes (e.g., the profile [UML-MARTE 2009] extends UML with concepts from the real-time and embedded systems domain, facilitating the derivation of performance or schedulability analysis models from UML+MARTE models). Since profiles are standard extension mechanisms, they are recognized by standard UML tools and can be exchanged among them.

Stereotypes and tagged-values are extension mechanisms used to define a profile. A stereotype extends one or more UML meta-classes and can be applied to UML model elements (components, states, transitions, etc.). For example, MARTE introduces a stereotype for a well-known concept from the the real-time domain, that of “type of clock” , which extends the UML meta-class “Class”. Figure 13 (a) depicts the definition of the stereotype “clockType”. The stereotype can be applied to any Class instance in a UML+MARTE model by labeling it with clockType, as shown in Figure 13 (b). Last, tagged-values represent the attributes of the stereotypes; for instance, in Figure 13 (a) are given the attributes of “clockType” stereotype (nature, unitType and so on).

A.1. Examples of UML-annotated models

The purpose of this sub-section is twofold: on the one hand, a reader non-familiar with UML can learn a few essential aspects of some of the most important UML diagrams; on the other hand, we show excerpts of some of the works surveyed in this paper. This way the reader can see how stereotypes and tagged-values are used to model dependability concepts within UML models (it is informally known as “UML-annotated models for dependability”).

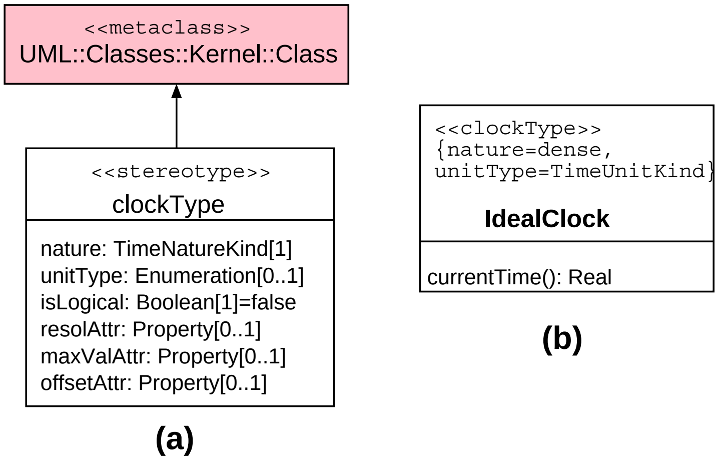

<<metaclass>>
UML::Classes::Kernel::Class
<<stereotype>>
clockType
nature: TimeNatureKind[1]
unitType: Enumeration[0..1]
isLogical: Boolean[1]=false
resolAttr: Property[0..1]
maxValAttr: Property[0..1]
offsetAttr: Property[0..1]
(a)
<<clockType>>
{nature=dense,
unitType=TimeUnitKind}
IdealClock
currentTime(): Real
(b)

Fig. 13. UML stereotypes and tagged-values.

A use case diagram (UCD) identifies the functionalities of the system at a high abstraction level. Each functionality is depicted as an ellipse (called use case) that interacts with actors (humans or other systems) to carry out the system responsibilities. Figure 14 depicts the UCD of a system meant to provide reliable communication; it has only one use case and two actors, the client sending the messages and the receiver. The use case is stereotyped as a *reliability handler* following [Mustafiz et al. 2008], which means that the use case addresses exceptional situations that threaten system reliability. The diagram also depicts an annotation from [Bernardi et al. 2009], which defines the use case as a service. This annotation assesses that service availability should be predicted (by analysis) as a rate of the successful messages (`$Xack`) out of all delivered messages (`$Xrequest`).

The deployment diagram (DD) identifies the system software components as well as the hardware nodes in which the former are deployed. In this case, we have used the proposal of [Bondavalli et al. 2001; Majzik et al. 2003], see Figure 14. It identifies which software and hardware are *stateful* or *stateless* and which components are working as *redundancy managers*, *variants* or *adjudicators* in a fault tolerance architecture.

The sequence diagram shows the messages exchanged between the system components. It provides useful constructors such as loops, alternatives or parallel execution. The example illustrated in Figure 15 corresponds to [Cortellessa and Pompei 2004]. It offers roles for the components, connectors and actors as well as probabilities of execution or failure for all these elements. This sequence diagram partially describes the system scenario represented by the previous use case, i.e., the message replication service.

The activity diagram, see Figure 16, specifies the control flow of a component, subsystem or system. It is widely used for modeling business processes, workflows or system low level processes. It features most of the common control flow structures such as decision, fork, join, loop or merge. Figure 16 models a partial behavior of a reliable communication system and illustrate the failure specification using the proposal of [Bernardi et al. 2009].

ACKNOWLEDGMENT

The authors thank the anonymous reviewers for their valuable help to improve this work.

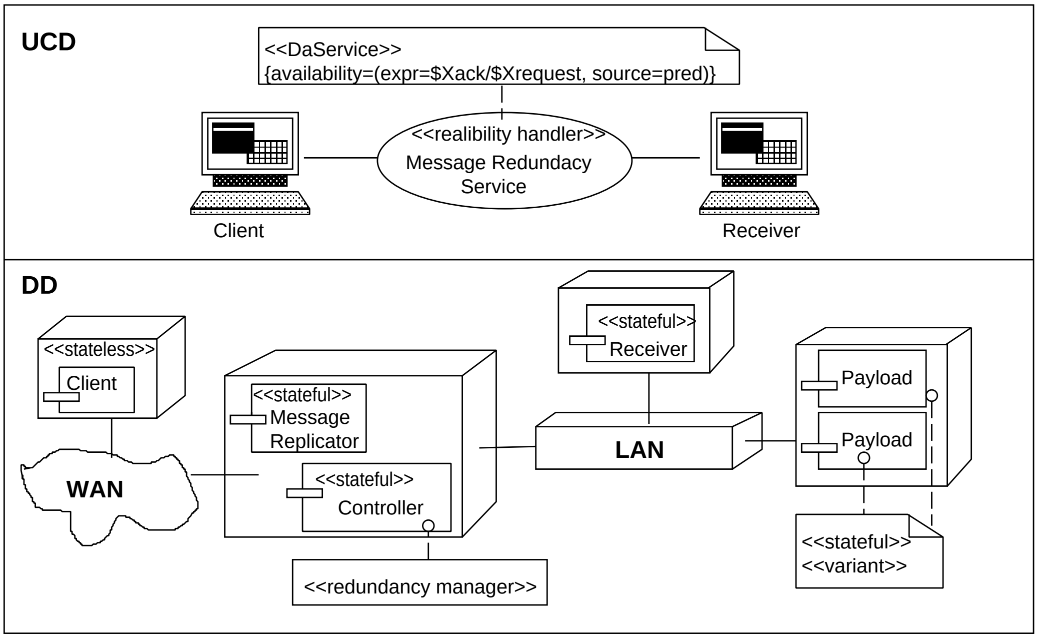


Fig. 14. Use case diagram (UCD) and deployment diagram (DD).

REFERENCES

ADDOUCHE, N., ANTOINE, C., AND MONTMAIN, J. 2006. 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)*. IEEE Computer Society, Sydney, Australia, 17.

ADDOUCHE, N. AND ANTOINE, C.AND MONTMAIN, J. 2004. UML models for dependability analysis of real-time systems. In *Proc. International Conference on Systems, Man and Cybernetics*. Vol. 6. IEEE CS., The Hague, Netherlands, 5209–5214.

AJMONE-MARSAN, M., BALBO, G., CONTE, G., DONATELLI, S., AND FRANCESCHINIS, G. 1995. *Modelling with Generalized Stochastic Petri Nets*. Wiley Series in Parallel Computing. John Wiley and Sons, England.

ALLENBY, K. AND KELLY, T. 2001. Deriving safety requirements using scenarios. In *5th IEEE International Symposium on Requirements Engineering*. IEEE Computer Society, Washington, DC, USA, 228–235.

ANSI/IEEE. 1991. Standard glossary of Software Engineering Terminology. Tech. Rep. STD-729-1991, ANSI/IEEE.

ARNOLD, T. 1973. The concept of coverage and its effect on the reliability model of a repairable system. *IEEE Transactions on Computers 22*, 251–254.

ARP-4754 1994. Certification considerations for highly-integrated or complex aircraft systems. Society of Automotive Engineers.

ARP-4761 1995. Guidelines and methods for conducting the safety assessment of civil airbone systems and equipment. Society of Automotive Engineers.

AVIŽIENIS, A. 1967. Design of fault-tolerant computers. In *Proceedings of the Fall Joint Computer Conference*. AFIPS '67 (Fall). ACM, New York, NY, USA, 733–743.

AVIZIENIS, A. 1985. The N-Version approach to Fault-Tolerant software. *IEEE Transactions on Software Engineering SE-11,* 12, 1491–1501.

AVIZIENIS, A., LAPRIE, J.-C., RANDELL, B., AND LANDWEHR, C. 2004. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing 01,* 1, 11–33.

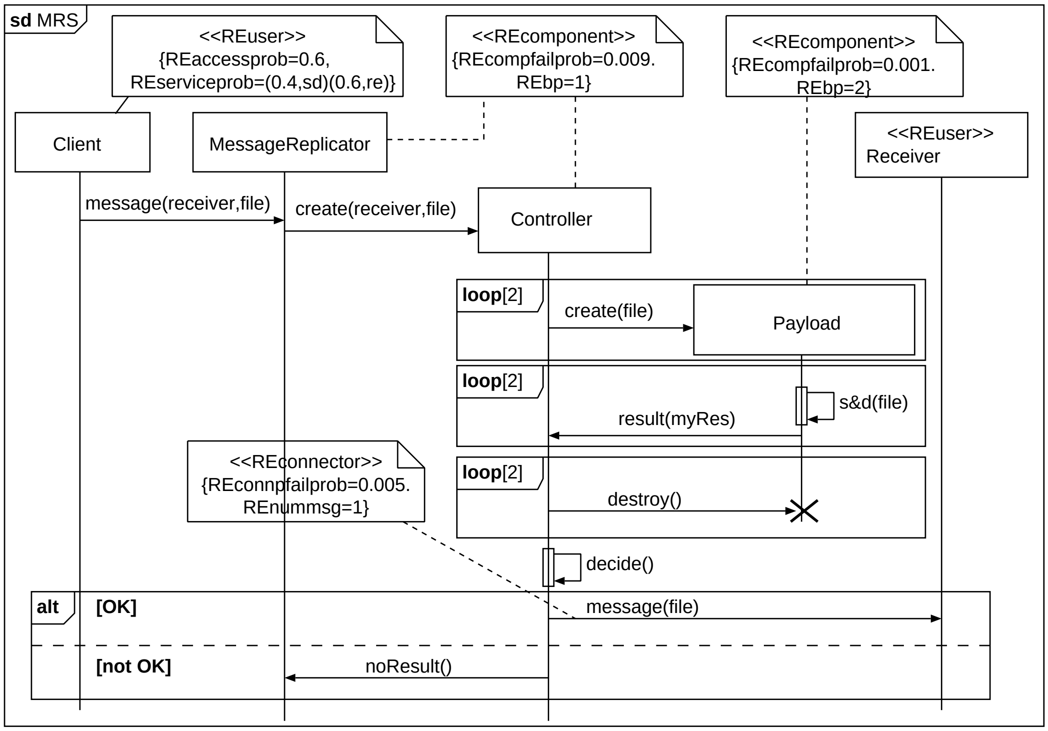


Fig. 15. Sequence diagram.

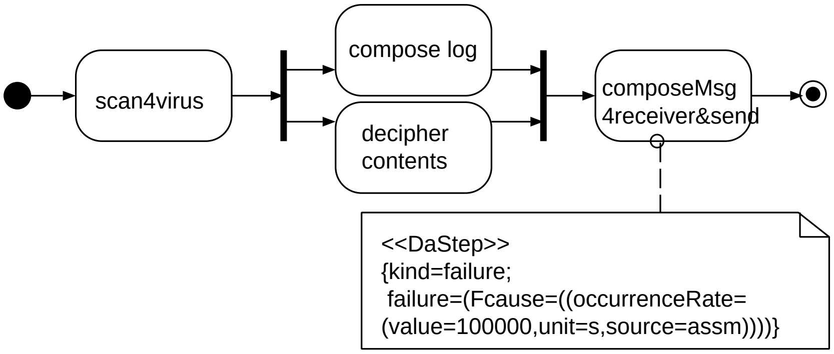


Fig. 16. Activity diagram.

BALSAMO, S., MARCO, A. D., INVERARDI, P., AND SIMEONI, M. 2004. Model-based performance prediction in software development: a survey. *IEEE Transactions on Software Engineering 30,* 5, 295–310.

BELL, M. 2008. *Service-Oriented Modeling (SOA): Service Analysis, Design and Architecture*. Wiley & Sons., Hoboken, New Jersey (US).

BERNARDI, S., DONATELLI, S., AND DONDOSSOLA, G. 2004a. A class diagram framework for collecting dependability requirements in automation systems. In *Proc. of the 1st International Symposium on Leveraging Applications of Formal Methods (ISOLA'04)*. Dept. of Computer Science, University of Cyprus, Paphos (Cyprus).

BERNARDI, S., DONATELLI, S., AND DONDOSSOLA, G. 2004b. Towards a methodological Approach to Specification and Analysis of Dependable Automation Systems. In *Proc. of the* 1^{st} *International Joint Conference on Formal Modelling and Analysis of Timed Systems (FORMATS) and on Formal Techniques in Real-Time and Fault Tolerant System (FTRTFT)*. Springer, Grenoble (France), 36–51.

BERNARDI, S. AND MERSEGUER, J. 2006. QoS Assessment via Stochastic Analysis. *IEEE Internet Computing 10,* 3, 32–42.

BERNARDI, S., MERSEGUER, J., AND PETRIU, D. 2009. A dependability profile within MARTE. *Software and Systems Modeling*, 1–24. 10.1007/s10270-009-0128-1.

BIBA, K. J. 1977. Integrity considerations for secure computer systems. Tech. Rep. MTR-3153, Mitre Corporation, Bedford MA. April.

BILLINTON, R. AND ALLAN, R. N. 1992. *Reliability evaluation of engineering systems: concepts and techniques*. Springer.

BOEHM, B. 1984. Verifying and validating software requirements and design specifications. *IEEE Software 1*, 75–88.

BONDAVALLI, A., DAL CIN, M., LATELLA, D., MAJZIK, I., PATARICZA, A., AND SAVOIA, G. 2001. Dependability analysis in the early phases of UML-based system design. *Int. Journal of Computer Systems Science & Engineering 16,* 5, 265–275.

BPEL 2007. Web Services Business Process Execution Language. Version 2.0.

CANCILA, D., TERRIER, F., BELMONTE, F., DUBOIS, H., ESPINOZA, H., GRARD, S., AND CUCCURU, A. 2009. Sophia: a modeling language for model-based safety engineering. In *2nd International Workshop On Model Based Architecting And Construction Of Embedded Systems*, S. Van Baelen, T. Weigert, I. Ober, and H. Espinoza, Eds. CEUR, Denver, Colorado, USA, 11–26.

CEA-LIST. 2008. Papyrus: open source tool for graphical UML modelling. Available at: http://www.papyrusuml.org/.

CHILLAREGE, R., BHANDARI, I. S., CHAAR, J. K., HALLIDAY, M. J., MOEBUS, D. S., RAY, B. K., AND WONG, M.-Y. 1992. Orthogonal defect classification-a concept for in-process measurements. *IEEE Trans. Softw. Eng. 18*, 943–956.

CLARK, D. D. AND WILSON, D. R. 1987. A comparison of commercial and military computer security policies. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE CS Press, Oakland, California, USA, 184–195.

CLEMENTS, P. AND NORTHROP, L. 2001. *Software Product Lines: Practice and Patterns*. Software Engineering Institute. Addison-Wesley, US.

CORTELLESSA, V. AND GRASSI, V. 2007. A modeling approach to analyze the impact of error propagation on reliability of component-based systems. In *Proceedings of the 10th international conference on Component-based software engineering*. CBSE'07. Springer-Verlag, Berlin, Heidelberg, 140–156.

CORTELLESSA, V. AND POMPEI, A. 2004. 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)*. ACM, New York, NY, USA, 197–206.

CORTELLESSA, V., SINGH, H., AND CUKIC, B. 2002. Early reliability assessment of UML based software models. In *Workshop on Software and Performance*. ACM, New York, NY, USA, 302–309.

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

D'AMBROGIO, A., IAZEOLLA, G., AND MIRANDOLA, R. 2002. A method for the prediction of software reliability. In *Proc. of the 6-th IASTED Software Engineering and Applications Conference (SEA2002)*. ACTA press, Cambridge, MA, USA.

DAVID, P., IDASIAK, V., AND KRATZ, F. 2009. Improving reliability sudies with SysML. In *RAMS09: Proceedings of the Reliability and Maintainability Symposium*. IEEE Computer Society, Fort Worth, Texas, (USA).

DE SOUZA E SILVA, E. AND GAIL, H. R. 1989. Calculating availability and performability measures of repairable computer systems using randomization. *J. ACM 36*, 171–193.

DEMIGUEL, M., LAMBOLAIS, T., PIEKAREC, S., BETGÉ-BREZETZ, S., AND PÉQUERY, J. 2001. Automatic generation of simulation models for the evaluation of performance and reliability of architectures specified in UML. In *EDO'00: Revised Papers from the Second International Workshop on Engineering Distributed Objects*. Springer-Verlag, London, UK, 83–101.

EN-50126 1999. Application ferroviaires - Spécification et démonstration de Fiabilité, Disponibilité, Maintenabilit'e et Sécurité (FMDS). Norme.

EN-50128 2001. Applications ferroviaires - Système de signalisation, de télécommunication et de traitement - Logiciels pour systèmes de commande et de protection ferroviaire. Norme.

EN-50129 2001. Application ferroviaires - Système de signalisation, de télécommunication et de traitement - systèmes électroniques relatifs à la sécurité pour la signalisation. Norme.

FOKKING, W. 2000. *Introduction to Process Algebra*. Springer-Verlag, Berling-Heidelberg.

GENERO, M., MANSO, E., VISAGGIO, A., CANOFRA, G., AND PIATTINI, M. 2007. Building measure-based prediction models for UML class diagram maintainability. *Empirical Software Engineering 12*, 517–549.

GENERO, M., PIATTINI, M., MANSO, E., AND CANTONE, G. 2003. Building UML class diagram maintainability prediction models based on early metrics. In *METRICS '03: Proceedings of the 9th International Symposium on Software Metrics*. IEEE Computer Society, Washington, DC, USA, 263.

GHEZZI, C., MANDRIOLI, D., AND MORZENTI, A. 1990. Trio: A logic language for executable specifications of real-time systems. *J. Syst. Softw. 12,* 2, 107–123.

GOKHALE, S. S. 2007. Architecture-based software reliability analysis: Overview and limitations. *IEEE Transactions on Dependable and Secure Computing 4,* 1, 32–40.

GOSEVA-POPSTOJANOVA, K., HASSAN, A., GUEDEM, A., ABDELMOEZ, W., NASSAR, D. E. M., AMMAR, H., AND MILI, A. 2003. Architectural-level Risk Analysis Using UML. *IEEE Transactions on Software Engineering 29,* 10, 946–960.

GRASSI, V., MIRANDOLA, R., AND SABETTA, A. 2005. 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)*. ACM, New York, NY, USA, 25–36.

GRASSI, V., MIRANDOLA, R., AND SABETTA, A. 2007. 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.

HANSEN, K., WELLS, L., AND MAIER, T. 2004. HAZOP analysis of UML-based software architecture description of safety-critical systems. In *Second Nordic Workshop on UML, Modeling, Methods and Tools*, K. Koskimies, L. Kuzniarz, J. Lilius, and I. Porres, Eds. TUCS, Turku, Finland.

HASSAN, A., GOSEVA-POPSTOJANOVA, K., AND AMMAR, H. 2005. UML Based Severity Analysis Methodology. In *Proc. of Annual Reliability and Maintainability Symposium (RAMS 2005)*. IEEE, Alexandria, VA.

HAWKINGS, R., TOYN, I., AND BATE, I. 2003. An approach to Designing Safety Critical Systems using the Unified Modelling Language. In *Workshop on Critical Systems Development with UML*. San Francisco (USA), 3–18.

HOSFORD, J. 1960. Measures of dependability. *Operations Researchs 8,* 1, 204–206.

HUANG, Y. AND KINDALA, C. 1996. Software fault tolerance in the application layer. In *Software Fault Tolerance*, M. R. Lyu, Ed. John Wiley and Sons Ltd., Chapter 10, 231–248.

IEC-60300-3-1 2003. Dependability Management. Part 3: Application Guide, Section 1: Analysis Techniques for dependability: Guide on methodology.

IEC-61131-1 1992. Programmable controllers, part 3: Programming languages. International Electrotechnical Commission.

IEC-61508 1998. Functional safety of electrical/electronic/programmable electronic safety-related systems. International Electro-technical Commission.

IMMONEN, A. AND NIEMELÄ, E. 2008. Survey of reliability and availability prediction methods from the viewpoint of software architecture. *Software and System Modeling 7,* 1, 49–65.

ISO/IEC 14764 2006. Standard for Software Engineering – Software Life Cycle Processes - Maintenance. International Organization for Standardization/International Electro-technical Commission.

ISO/IEC9126-1.2 2001. Information technology - software product quality. part 1: quality model. International Electro-technical Commission.

IWU, F., GALLOWAY, A., MCDERMID, J., AND TOYN, I. 2007. Integrating safety and formal analyses using UML and PFS. *Reliability Engineering and System Safety 92,* 2, 156–170.

JACOBSON, I. 1995. *Object-Oriented Software Engineering: a Use Case driven Approach*. Addison–Wesley, Wokingham, England.

JüRJENS, J. AND WAGNER, S. 2005. Component-based Development of Dependable Systems with UML. In *Component-Based Software Development*, A. et al., Ed. LNCS Series, vol. 3778. Springer-Verlag, Berlin/Heidelberg, 320–344.

JOHANNESSEN, P., GRANTE, C., ALMINGER, A., EKLUND, U., AND TORIN, J. 2001. Hazard analysis in object-oriented design of dependable systems. In *Proc. of the International Conference on Dependable Systems and Networks (DSN01)*. IEEE Computer Society, Washington, DC, USA, 507–512.

JOHNSON, B. W. 1989. *Design and analysis of fault-tolerant digital systems*. Addison-Wesley.

JUNGCLAUS, R., SAAKE, G., HARTMANN, T., AND SERNADAS, C. 1996. Troll: a language for object-oriented specification of information systems. *ACM Trans. Inf. Syst. 14*, 175–211.

JÜRJENS, J. 2003. Developing safety-critical systems with UML. In *UML 2003, San Francisco*. LNCS Series, vol. 2863. Springer-Verlag, Berlin/ Heidelberg, 360–372.

LAMPORT, L., SHOSTAK, R., AND PEASE, M. 1982. The byzantine generals problem. *ACM Trans. Program. Lang. Syst. 4*, 382–401.

LAZOWSKA, E., ZAHORJAN, J., SCOTT GRAHAM, G., AND SEVCIK, C. 1984. *Quantitative System Performance: Computer System Analysis Using Queueing Network models*. Prentice-Hall, New Jersey (USA).

LEANGSUKSUN, C., SONG, H., AND SHEN, L. 2003. Reliability modeling using UML. In *Proceedings of the International Conference on Software Engineering Research and Practice (SERP03)*, B. Al-Ani, H. R. Arabnia, and Y. Mun, Eds. CSREA Press, Las Vegas, Nevada (USA), 259–262.

LEVESON, N. G. 1995. *Safeware*. Addison-Wesley, USA.

LITTLEWOOD, B. AND STRIGINI, L. 1993. Validation of ultrahigh dependability for software-based systems. *Commun. ACM 36*, 69–80.

LIU, J., DEHLINGER, J., AND LUTZ, R. R. 2007. Safety analysis of software product lines using state-based modeling. *Journal of Systems and Software 80,* 11, 1879–1892.

LIU, J. W. 2000. *Real-time Systems*. Prentice Hall, Upper Saddle River, New York.

LU, S. AND HALANG, W. A. 2007. A UML profile to model safety-critical embedded real-time control systems. In *Contributions to Ubiquitous Computing*, B. J. Krämer and W. A. Halang, Eds. Studies in Computational Intelligence Series, vol. 42. Springer, Berlin, Heidelberg, 197–218.

LYU, M. 1995. *Software Fault Tolerance*. John Wiley & Sons, Ltd.

LYU, M. R., Ed. 1996. *Handbook of Software Reliability Engineering*. IEEE Computer Society Press, NY.

MAJZIK, I., PATARICZA, A., AND BONDAVALLI, A. 2003. Stochastic Dependability Analysis of System Architecture Based on UML Models. In *Architecting Dependable Systems, LNCS 2677*, R. De Lemos, C. Gacek, and A. Romanovsky, Eds. Lecture Notes in Computer Science. Springer-Verlag, Berlin, Heidelberg, New York, 219–244.

MEYER, J. F. 1980. On evaluating the performability of degradable computing systems. *IEEE Trans. Comput. 29*, 720–731.

MIL-STD-1629A 1984. Procedures for performing failure mode effects and criticality analysis. US Military standard, MIL-STD-1629A/ notice 2.

MIL-STD-882d 1999. System safety program requirements. MIL-STD-882, United States of America.

MUSTAFIZ, S. AND KIENZLE, J. 2009. DREP: A requirements engineering process for dependable reactive systems. In *Methods, Models and Tools for Fault Tolerance*, M. J. Butler, C. B. Jones, A. Romanovsky, and E. Troubitsyna, Eds. Lecture Notes in Computer Science Series, vol. 5454. Springer, Berlin / Heidelberg, 220–250.

MUSTAFIZ, S., KIENZLE, J., AND BERLIZEV, A. 2008. Addressing degraded service outcomes and exceptional modes of operation in behavioural models. In *SERENE '08: Proceedings of the 2008 RISE/EFTS Joint International Workshop on Software Engineering for Resilient Systems*. ACM, New York, NY, USA, 19–28.

MUSTAFIZ, S., SUN, X., KIENZLE, J., AND VANGHELUWE, H. 2008. Model-driven assessment of system dependability. *Software and System Modeling 7,* 4, 487–502.

OBER, I., GRAF, S., AND OBER, I. 2006. Validating timed UML models by simulation and verification. *STTT 8,* 2, 128–145.

OCL 2010. Object Constraint Languageo. Version 2.2.

OpNet 1999. OpNet modeler. http://www.opnet.com/solutions/network_rd/modeler.html.

PAI, G. J. AND DUGAN, J. 2002. Automatic Synthesis of Dynamic Fault Trees from UML System Models. In *Proc. of 13th International Symposium on Software Reliability Engineering (ISSRE-02)*. IEEE Computer Society, Annapolis, MD, USA, 243–256.

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

PATARICZA, A. AND GYÖR, F. 2004. Towards unified dependability modeling and analysis. In *Workshops Proceedings Organic and Pervasive Computing*. Lecture Notes in Informatics. GI, Gesellschaft für Informatik, Bonn, Germany, 113–122.

PATARICZA, A., MAJZIK, I., HUSZERL, G., AND V'ARNAY, G. 2003. UML-based design and formal analysis of a safety-critical railway control software module. In *In Proc. of Symposium Formal Methods for Railway*

Operation and Control Systems (FORMS03), G. Tarnai and E. Schnieder, Eds. Budapest (Hungary), 125–132.

POWELL, D. 1992. Failure mode assumptions and assumption coverage. In *Fault-Tolerant Computing, 1992. FTCS-22. Digest of Papers., Twenty-Second International Symposium on*. IEEE Computer Society, Boston, MA, USA, 386 –395.

RODRIGUES, G. N., ROSENBLUM, D. S., AND UCHITEL, S. 2005. Reliability prediction in model-driven development. In *Model Driven Engineering Languages and Systems, 8th International Conference (MoDELS 2005)*, L. C. Briand and C. Williams, Eds. Lecture Notes in Computer Science Series, vol. 3713. Springer, Montego Bay, Jamaica, 339–354.

RTCA. 1992. Software considerations in airbone systems and equipment certification. Radio Technical Commission for Aeronautics (RTCA), European Organization for Civil Aviation Electronics (EUROCAE), no.DO-178B/ED-12B.

RUMBAUGH, J. E., BLAHA, M. R., PREMERLANI, W. J., EDDY, F., AND LORENSEN, W. E. 1991. *Object-Oriented Modeling and Design*. Prentice-Hall.

SAILER, R., ZHANG, X., JAEGER, T., AND VAN DOORN, L. 2004. Design and implementation of a tcg-based integrity measurement architecture. In *Proceedings of the 13th conference on USENIX Security Symposium - Volume 13*. SSYM'04. USENIX Association, Berkeley, CA, USA, 223–238.

SCHMIDT, D. C. 2006. Guest editor's introduction: Model-driven engineering. *IEEE Computer 39,* 2, 25–31.

SINGH, H., CORTELLESSA, V., CUKIC, B., GUNEL, E., AND BHARADWAJ, V. 2001. A Bayesian approach to reliability prediction and assessment of component based systems. In *12th International Symposium on Software Reliability Engineering (ISSRE 2001), 27-30 November 2001, Hong Kong, China*. IEEE Computer Society, Washington, DC, USA, 12–21.

STAHL, T. AND VÖLTER, M. 2006. *Model-driven software development*. John Wiley & Sons, Ltd., New York.

SysML 2010. System Modeling Language. Version 1.2, formal/2010-06-01.

SZYPERSKI, C. 1998. *Component Software: Beyond Object-Oriented Programming*. ACM Press and Addison-Wesley, New York, NY.

TCG 2011. http://www.trustedcomputinggroup.org.

TRIVEDI, K. 2001. *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*. John Wiley and Sons, NY.

UK Ministry of Defence 2000. *HAZOP Studies on Systems Containing Programmable Electronics*. UK Ministry of Defence. Glasgow (UK).

UML 2005. Unified Modeling Language: Superstructure. Version 2.0, formal/05-07-04.

UML-EDOC 2001. UML Profile for Enterprise Distributed Object Computing. Version 1.0.

UML-MARTE 2009. UML profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE). Version 1.0, OMG document formal/2009-11-02.

UML-QoS&FT 2008. UML Profile for Modeling Quality of Service and Fault Tolerant Characteristics and Mechanisms. V1.1, formal/08-04-05.

UML-SPT 2005. UML Profile for Schedulabibity, Performance and Time Specification. Version 1.1, formal/05-01-02.

VESELY, W., GOLDBERG, F., ROBERTS, N., AND HAASL, D. 1981. Fault Tree Handbook. System and Reliability Research, Office of Nuclear Regulatory Research, U.S. Nuclear Regulatory Commission, Washington, D.C. 20555.

WEYUKER, E. J. 1982. On Testing Non-Testable Programs. *The Computer Journal 25,* 4, 465–470.

YACOUB, S. M., CUKIC, B., AND AMMAR, H. H. 2004. A scenario-based reliability analysis approach for component-based software. *IEEE Transactions on Reliability 53,* 4, 465–480.

Z 2002. Z Formal Specification Notation: Syntax, Type System and Semantics. ISO/IEC 13568:2002 ed.

ZARRAS, A., VASSILIADIS, P., AND ISSARNY, V. 2004. Model-driven dependability analysis of webservices. In *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE, OTM Confederated International Conferences, Agia Napa, Cyprus, October 25-29, 2004, Proceedings, Part II*, R. Meersman and Z. Tari, Eds. Lecture Notes in Computer Science Series, vol. 3291. Springer, Berlin / Heidelberg, 1608–1625.

ZOUGHBI, G., BRIAND, L., AND LABICHE, Y. 2006. A UML profile for developing airworthiness-compliant (RTCA DO-178B) safety-critical software. Tech. rep., Carleton University, Canada, tech.rep.SCE-05-19.

ZOUGHBI, G., BRIAND, L., AND LABICHE, Y. 2007. A UML Profile for Developing Airworthiness-Compliant (RTCA DO-178B), Safety-Critical Software. In *Proceedings of Models 2007*, G. Engels, Ed. LNCS Series, vol. 4735. Springer-Verlag, Berlin, Heidelberg, 574–588.

Received 0000; revised 0000; accepted 0000