Timing-Failure Risk Assessment of UML Design Using Time Petri Net Bound Techniques

Simona Bernardi, Javier Campos, and José Merseguer

***Abstract*—Software systems that do not meet their timing constraints can cause risks. In this work, we propose a comprehensive method for assessing the risk of timing failure by evaluating the software design. We show how to apply best practises in software engineering and well-known Time Petri Net (TPN) modeling and analysis techniques, and we demonstrate the effectiveness of the method with reference to a case study in the domain of real-time embedded systems. The method customizes the Australian standard risk management process, where the system context is the UML-based software specification, enriched with standard MARTE profile annotations to capture nonfunctional system properties. During the risk analysis, a TPN is derived, via model transformation, from the software design specification and TPN bound techniques are applied to estimate the probability of timing failure. TPN bound techniques are also exploited, within the risk evaluation and treatment steps, to identify the risk causes in the software design.**

***Index Terms*—MARTE profile, risk assessment, time Petri net (TPN) bound techniques, unified modeling language (UML).**

I. Introduction

THE quantitative evaluation of software systems early in the life cycle is not yet a common practice for most of the software projects. The software engineering community lacks of quantitative evaluation techniques properly integrated within the software standards and the current development methodologies. On the other hand, formal quantitative methods, such queueing networks [1], timed automata [2] or Petri nets [3], [4], when adequately applied to software system design, have been proved to be useful to predict and validate a large number of their nonfunctional properties, e.g., performance, timeliness or reliability.

In particular, the assessment of timing constraints is crucial for real-time systems, since the inability of the system to meet a deadline may result in an incorrect system behavior then leading to an unpredictable consequence. Then, the consequences of software failures have to be analyzed not only early in the life cycle but also using an appropriate paradigm.

In this work, we consider the early development stages of soft real-time systems, when the detection of timing violations is aimed at reducing the number of missed deadlines. We show how to apply best practises in software engineering and well-known Time Petri Net (TPN) [5] modeling and analysis techniques to propose a comprehensive and low-cost method for assessing the risk of timing failures by evaluating the software design. We also demonstrate the effectiveness of the method with reference to a case study in the domain of real-time embedded system.

In order to make the proposed method useful, from the software engineers point-of-view: 1) we fit our proposal into a standard risk assessment methodology and 2) we assume the software system specified using standard OMG [6] languages. Concerning the first point, we have identified several standards which aim at managing mainly security risks in different software contexts. However, since there is not a widely accepted *software* risk process, we have decided to learn from other areas where the activities involved in risk assessment have been also standardized, such as business or chemical industry. Finally, among the different processes, we have chosen the standard for Risk Management Process (RMP) [7] that provides a waterfall model easy to apply for risk assessment and to customize in the software domain. Regarding the second point, UML [8] is used for the software and hardware platform specification, and the "Modeling and Analysis of Real Time and Embedded Systems" profile [9] (MARTE) is applied for the definition of nonfunctional properties, input values and parameters.

According to RMP, the risk analysis consists in computing the risk as the function of two factors: the likelihood and the consequence. In our context, the former is the probability that the service is delivered too late (or too early), and the latter is the impact of the late (early) service delivery on the software system users and/or environment. Indeed, Avizienis *et al.* [10] classify a timing failure as a type of service failure that occurs when the service is delivered either too late or too early, i.e., the system does not meet the timing constraints. For the *likelihood* estimation, we apply TPN and their efficient bound techniques [11]. Regarding the *consequence* factor, our method resorts to standard analysis techniques (e.g., Functional Failure Analysis or Preliminary Hazard Analysis) used in the software domain to determine it [12]–[14].

We choose TPN since they are suited for modeling real-time systems early in the life-cycle, where the software timing specifications are not deterministic, but still can be expressed as

Manuscript received March 18, 2010; revised August 03, 2010 and October 18, 2010; accepted December 01, 2010. Date of publication December 17, 2010; date of current version February 04, 2011. This work was supported in part by the Distributed Supervisory Control of Large Plants Project (DISC n.INFSO-ICT-224498), under the Seventh Framework European Program. Paper no. TII-10-03-0055.

S. Bernardi is with the Centro Universitario de la Defensa, Academia General Militar, 50018 Zaragoza, Spain (e-mail: simonab@unizar.es).

J. Campos and J. Merseguer are with the Departamento de Informática e Ingeniería de Sistemas, Universidad de Zaragoza, 50090 Zaragoza, Spain (e-mail: jcampos@unizar.es; jmerse@unizar.es).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TII.2010.2098415

interval (i.e., min/max) values. Besides, the TPN bound techniques are based on the formulation and solving of linear programming problems and the computational effort to get results is significantly lower than using conventional enumerative techniques, so also complex systems can be analyzed. The results from the TPN bound analysis will be also exploited by identifying and evaluating critical elements in the design. The risk likelihood is then estimated as the probability that the service response time is not greater (less) than the maximum (minimum) timing constraint. We assume the service response time uniformly distributed between the computed upper and lower bounds. In probabilistic risk assessment, the uniform distribution is usually recommended to determine likelihood when no information about the shape of the random variable is known [15], [16]. It represents the state of knowledge for the situations where little *a priori* information exists. The selection of uniform is based on the Laplace's "principle of insufficient reason" (first enunciated by Jakob Bernoulli): *The uniform distribution leads to the most conservative estimate of uncertainty; i.e., it gives the largest standard deviation*. From the usefulness point-of-view, the uniform assumption permits to provide not trivial estimations with respect to the deterministic one.

The proposed technique supports a preliminary risk evaluation in the software life-cycle; as system measures become available, later in the life-cycle, other assumptions on the type of distribution over the bound interval could be more appropriate. In order to analyze the sensitivity of the risk likelihood, with respect to the distribution assumption, we have compared the probability of service timing-failure of the running example under the uniform and normal hypothesis. The analysis results show that both distribution lead to similar outcomes. In general, the difference in the risk likelihood evaluation depends not only on the timing constraint but also on the mapping of the quantitative values onto likelihood categories.

This paper is structured as follows:Section II reviews the related works; Section III gives an overview of the method and introduces basic concepts. Section IV addresses the risk context and identification steps; Section V describes the approach to estimate the risk likelihood; Section VI addresses the risk evaluation and treatment steps; Section VII presents the case study; finally, conclusions are given in Section VIII.

II. Related Work

Our approach conforms to the RMP standard [7] to support the risk assessment in the early stages of the software life-cycle. The risk assessment metamodel, defined in the UML profile for "Modeling QoS and Fault Tolerance Characteristics and Mechanisms" (QoS&FT) [17], also relies on the RMP standard. Nevertheless, we use the UML profile MARTE [9] due to two main reasons. First, the specification of nonfunctional properties (NFPs) is more rich in MARTE than in QoS&FT; indeed, the latter is mainly based on use case diagrams, that are not enough for our purposes. Second, the specification of the system functional properties in QoS&FT is independent of the risk specification, then leading to the introduction of new UML model elements to represent NFPs.

Some standards deserve to be discussed and compared with RPM. Hence, ISO has proposed the 2700× series, the closest to RPM is the ISO 27005 [18] which is devoted to information security risk management, however no specific method for risk management is prescribed. In the American context, the NIST [19] standard is a comprehensive guide for risk management of IT systems and it encompasses risk assessment as well as risk mitigation. In the German context, BSI [20] is an standard for IT information security managers that describes step by step how an information security management system can be designed. As discussed in [21], these standards propose qualitative evaluation, which differs from the quantitative needs of our proposal, that are indeed offered by RPM. Other nonstandard proposals also deserve to be mentioned such as CORAS [22], OCTAVE [23], EBIOS [24], and CRAMM [25]. All of them are related to security management but do not deal with timing aspects as it is the focus of our work. Among them, only CORAS offers a quantitative evaluation. CORAS, as our proposal, follows the RPM steps and uses UML diagrams to model the system behavior. OCTAVE is a comprehensive approach, compliant with the U.S. Department of Defense and more related to procedures such as organizational data. EBIOS was developed for the French National Defense and offers a detailed guide about how to identify security needs, characterize attacks or identify vulnerabilities. CRAMM undertakes risk analysis of information systems and networks, and can be used during analysis and design of information systems, as well as for existing systems.

Concerning proposals for risk analysis of software systems based on a quantitative estimation, it is worth mentioning the works [26], [27]. Both, like our proposal, can be applied early in the life-cycle and use UML for software specification. In [26] a methodology is proposed to assess software risks introduced by the environment and it is aimed at identifying potentially unreliable software components. The risk is computed as the combination of two parameters: the probability and the consequence of malfunctioning. UML state charts and sequence diagrams are used to identify the risky components and connectors, respectively, in the software architecture. From the risk associated with components and connectors, scenario risk factors are calculated by creating and solving a Markov model. The work [27] proposes a methodology for the estimation of the performance based risk factor, which originates from violations of performance properties. Annotated sequence and deployment UML diagrams are elaborated to estimate the scenario failure probability using the classical analysis in [1]. The approach supports the identification of risky scenarios and risky software components when the timing specification is expressed by mean values. Our approach, instead, is suited to soft real-time systems where timing specifications are given as min/max values. Moreover, in [27] only standalone analysis is supported (i.e., no model of concurrency is introduced), while our method can manage the concurrency and the resource contention. TPNs also promote the advantage for our method to identify critical hardware resources. Finally, another important difference is that, being built on a standard risk process, our proposal benefits from the well-established knowledge in this area.

The risk analysis involves decisions about which translation method, from UML to Petri Nets, we should use. Our work concerns the translation of UML sequence and deployment diagrams. A lot of efforts have been spent by the researchers during

the last ten years to derive formal models for the quantitative analysis of UML-based specifications, as surveyed in [28]–[30], respectively, for performance, timing and dependability assessment. We will discuss the proposals that use Petri Nets as formal model [31]–[38] as in our approach. Some of them [31]–[34] are mainly oriented to performance analysis.

For our purposes, the choice should be driven by the following requirements: the translation should be simple enough to be automatically supported; the resulting Petri net has to satisfy some "good properties" (e.g., liveness and boundedness) since the goodness of the bound techniques is sensitive to the net structure.

The work [31] proposes a systematic approach to transform UML's Collaboration and Statecharts to Generalized Stochastic Petri Nets (GSPNs), while the work [32] derives a GSPN model from UML's Sequence Diagrams (SDs) and StateCharts, using a compositional approach. Both the approaches need the StateChart specification to get a GSPN model, that we do not use in this paper. The PUMA [33] approach translates the UML diagrams into an intermediate model, the Core Scenario Model [39] (CSM), that separates the functional from the nonfunctional information. The CSM can be translated into different performance formalisms, i.e., Petri nets or queuing networks. We discarded this choice since the CSM would introduce a significant complexity to our proposal, which is comprehensive enough.

The proposals in [34] and [38] translate all the richness of the UML 2 SD into Stochastic Well-formed Nets and Colored Petri Nets, respectively. However, we discarded such approaches since in [34] the UML specification has to be enriched, as in [32], with one UML state machine per lifeline in the SD. On the other hand, in [38] the lifelines in the SD, and then their execution specifications, are not explicitly modeled. The works [35]–[37] propose a similar translation approach from either Message Sequence Charts (precursor of SD) or SD to untimed Petri Nets. Indeed, each object participating in an interaction is represented by a net that captures the sequence of events and these nets are connected through places representing mailboxes, then preserving the *weak sequencing semantics* introduced in UML 2 for SDs. Such works are mainly aimed at assessing qualitative properties of the system design. In particular, the work [35] provides a formal semantics of the ordering actions in a SD through untimed Petri nets. Kluge [36] and Eichner *et al.* [37] obtain untimed high-level Petri nets from Message Sequence Charts and SD, respectively, by adopting a compositional approach. Actually, the translation proposed in [37] is similar to the Kluge's proposal but it considers most of the UML-2 SD constructs. We have chosen, then, the approach [37] to get the Petri net structure from the SD.

TPNs have been extensively used for the validation of real-time systems and their analysis is mainly based on enumerative techniques [4], [40], [41], which suffer the well-known state-space explosion problem. In [42], a stochastic extension of TPNs is proposed to support the performance evaluation of real-time systems, that consists in associating a probability density-function to the static firing interval of each nondeterministic transition. The main drawback of the approach is the computational complexity that is by far beyond the limit that our approach aims to address. Our uniform assumption, for the estimation of the risk likelihood, corresponds to the computation of a simple ratio and allows us to avoid both the introduction of non-Markovian stochastic Petri net models and the use of analysis techniques which are much more complex than the TPN bound techniques.

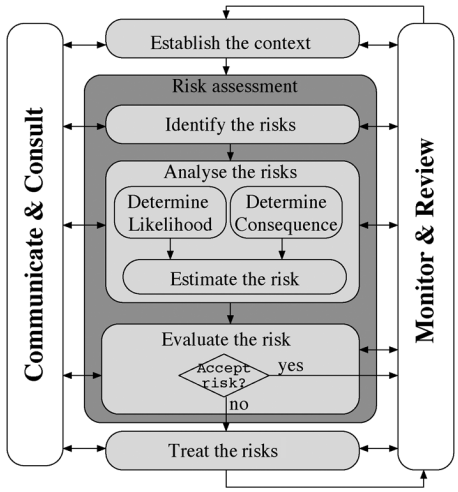


Fig. 1. Risk management process AS/NZS 4360:1999.

III. Overview of the Method and Background

The standard for Risk Management Process (RMP) [7] provides a waterfall model easy to apply for risk assessment and to customize in the software domain. As shown in Fig. 1, the RMP consists of sequential steps (gray ones) and transversal steps (white ones) that involve the whole process. The objective of our proposal is to tailor the main activities of the RMP carried out in the sequential steps to the tasks for the assessment of the timing-failure software risk, as detailed in the following.

1) Establish the Context: The activities performed in this step establish the boundaries within which the risk process will apply, i.e., the internal and external contexts, and the "risk criteria" (e.g., safety, media exposure, timing or financial impact). When it comes to tailor these activities to our method, they mainly consist in the specification of the software system. The internal context will be modeled by UML use case and sequence diagrams to define the system scenarios at a high level and detailed level, respectively. The UML notation is rich enough to express the functional system properties. However, to express all the necessary parameters to carry out risk assessment, it has to be enriched with non functional properties which will be specified using the MARTE profile [9]. The external context will be represented by a UML deployment of the software system environment or hardware platform, where the software components involved in a given scenario will be deployed and operate. Finally, our method is concerned with the adoption of only one risk criterion: the risk associated with the timing failure of system scenarios.

2) Identify the Risks: The purpose of this activity is the identification of the current risks, according to the risk criterion, affecting the project objectives. In our context, this task consists in the definition of the timing constraints associated with the system scenarios. In particular, for each high-level scenario,

TABLE I
RISK MATRIX

Likelihood \ Consequence	negligible	marginal	critical	catastrophic
frequent [0.1,1]	intolerable	intolerable	intolerable	intolerable
moderate [0.01,0.1)	undesirable	undesirable	intolerable	intolerable
occasional [1E-3,0.01)	tolerable	undesirable	undesirable	intolerable
remote [1E-4,1E-3)	negligible	tolerable	undesirable	undesirable
unlikely [1E-5,1E-4)	negligible	negligible	tolerable	undesirable
impossible [0,1E-5)	negligible	negligible	negligible	tolerable

the software engineer should specify the maximum (or the minimum) threshold for the response time as well as the type of timing failure (i.e., early or late). Such specifications will be expressed, using the MARTE profile, in the use case diagrams created in the previous step.

3) Analyze the Risks: This step assists the analyst in providing an estimation of the risks, previously identified, in order to make a decision about committing resources to control them. The RMP provides a simple formula for the estimation of the risk, that is the product of the possible consequence, or impact, of an event with the likelihood of that event occurrence. One of the main contributions of this work concerns the estimation of the likelihood factor, that is the probability of the scenario timing failure. Indeed, we provide a derivation method of a TPN model from the UML-MARTE specification of the system and a technique to compute such a probability that exploits efficient TPN bound techniques.

4) Evaluate the Risks: The risk evaluation consists in deciding whether the risk is either tolerable or not. In the latter case, the risk treatment is required. The risk is evaluated with the help of a *risk matrix* that combines the likelihood and the consequence factors estimated in the analysis step. Such type of matrices are often used in safety analysis and risk assessment [14] since they can express the nonlinearity of the risk metric with respect to the likelihood and consequence categorization. Their definition (i.e., the consequence and likelihood categories and their mapping to quantitative values) depends on the application domain. In the example presented in this work, we have used the matrix in Table I, which is inspired by [43]. Observe that this choice does not affect the applicability of our approach, since the matrix can be changed according to the application domain.

5) Treat the Risks: This step is carried out when the risk is not tolerable. The treatment activities identify and evaluate strategies to treat or control the risk, in order to either reduce the likelihood or to reduce/eliminate the negative consequences associated with the risk. In our approach, the root causes of a scenario timing failure concern with the software and hardware resources and their configuration. We will exploit the results from the TPN bound analysis to pinpoint such causes. Then, we will propose general guidelines about how to modify and improve the design to reduce the probability of scenario timing failure.

A. Background on UML and MARTE Profile

UML [8] is a general purpose visual modeling language that provides several types of diagrams to capture different aspects of the system. Structural diagrams (i.e., class, object, component, collaboration and deployment) represent the system from a static point-of-view. In particular, we will use deployment diagrams that describe the execution architecture of the system specifying the assignment of software components to nodes. Behavioral diagrams (i.e., use case, interaction, state machine, activity) are used instead to model the dynamic of the system. We will consider use case and sequence diagrams in our approach. Use case diagrams are typically used to capture the functional requirements of a system (that is, what a system is supposed to do) by means of use cases and actors. Sequence diagrams are, instead, a kind of interaction diagrams that can be used to model the use case realization. They show the temporal interaction of the system components through message exchange. Sequence diagrams may contain combined fragments (e.g., alternative, parallel) to represent complex control flows. UML provides a lightweight mechanism, the *profiling*, to extend its metamodel for different purposes. The profiling is a metamodeling technique where the profiles are kinds of packages that extends a reference metamodel with the purpose of tailoring it to a specific platform or domain. A profile contains a set of stereotypes that define how specific metaclasses are extended. Like classes, stereotypes may have properties (tags) and constraints. When a stereotype is applied to a model element, the values of the properties are called tagged-values.

MARTE [9] is a UML profile that supports the modeling and analysis of systems that need to verify timing constraints. In particular, MARTE allows one to specify nonfunctional properties (NFPs) according to a well-defined Value Specification Language (VSL) syntax. The "Dependability Analysis and Modeling" profile [30] is a MARTE specialization, then a MARTE-DAM annotation *stereotypes* the design model element it affects in the way UML proposes, i.e., by extending its semantics.

Fig. 2 depicts an excerpt of the MARTE (a) and DAM stereotypes (b) and tagged values used in this work. According to UML, each stereotype is made of a set of tags which define its properties. For example, *GaScenario* stereotype has *respT* and *hostDemand* as tags which are used to specify the scenario response time and the CPU demand on the processing host, respectively. The types of the tags are basic UML types (e.g., integer, enumeration) or MARTE NFP types (e.g., *NFP_Duration*). Moreover, DAM enriches the MARTE types library with basic and complex dependability types (c). The latter (e.g., *DaFailure*) are composed of attributes that can be MARTE NFP types (e.g., *NFP_Real*), basic dependability types (e.g., *DaCriticalLevel*) or simple types (e.g., *CriticalLevel* and *Domain*). MARTE NFP types are data-types of special importance

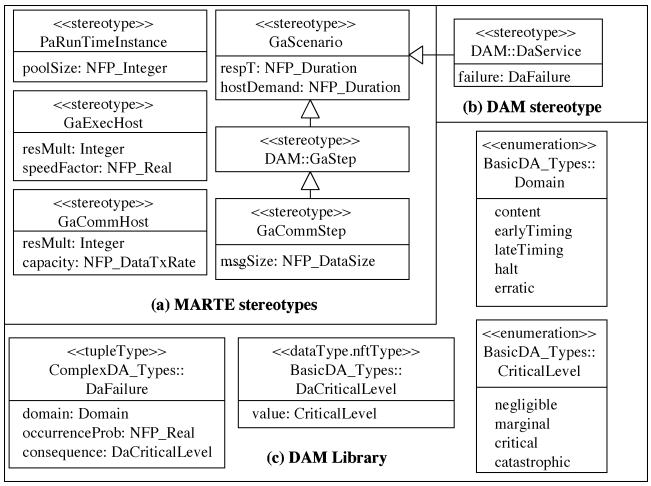


Fig. 2. MARTE-DAM extensions.

since they enable the description of relevant NFP aspects using properties such as: *value*, a value or parameter name (prefixed by the dollar symbol); *expr*, a VSL expression; *source*, the origin of the NFP—such as required (*req*), or estimated (*est*) parameter; and *statQ*, the type of statistical measure (e.g., maximum, minimum, and mean).

IV. Establish the Context and Identify the Risks

The first two steps of RMP provide the necessary UML specification where to identify the property to be fulfilled, the input parameters and the metrics to be estimated. Such system description consists of a use case, a sequence and a deployment diagram. In the use case diagram, each use case (UC) represents a high-level scenario of the system, where the risk associated with a timing failure will be identified and annotated. When a use case diagram contains more than one UC, i.e., high-level scenarios, each UC is considered independently of each other assuming that each one involves different timing failure risks for the system. Each use case is refined with a sequence diagram, which represents the sequence of actions that the software performs, as well as the messages exchanged in the scenario. A sequence diagram can contain *alternative*, *optional* and *parallel* combined fragments. The other types of combined fragments (such as *loop* fragments) are instead not supported by our method. The deployment diagram is used to specify the execution platform.

Observe that, in the current conventional software development methodologies as well as in more agile ones (e.g., Agile Unified Process), the output of the design consists of a set of artifacts that normally includes the UML diagrams our method uses as input specification. Therefore, our proposal could be easily integrated within such methodologies.

The properties, input parameters and metrics will be defined using the MARTE profile. Concretely, we use the Generic Quantitative Analysis Modeling (GQAM) and Performance Analysis (PA) extensions of MARTE (stereotypes prefixed as *Ga* and *Pa*, respectively) and the Value Specification Language (VSL) to specify the input parameters. The properties and the metrics to be estimated are specified instead using the extensions of the Dependability Modeling and Analysis (DAM) profile [30] (stereotype prefixed as *Da*).

Fig. 3 shows the running case with the complete and necessary subset of MARTE-DAM annotations for our method to work. In each UC, stereotyped as *DaService*, the software engineer identifies the risk of a timing failure by specifying the timing constraint (*respT*) and the corresponding type of failure (*failure.domain*). In particular, the former expresses the maximum or the minimum time allowed for the scenario to be executed. Accordingly, the latter can be either a late or early timing failure. In Fig. 3(a), the maximum scenario response time is of 30 s, then the risk associated is a late timing failure. Moreover, the two risk factors are annotated as parameters to be estimated during the risk analysis, that is the probability (*failure.occurrenceProb*) and the consequence (*failure.consequence*) of a timing failure. In Fig. 3(a), such parameters are expressed with the two variables *Prob* and *Cons*.

The input parameters are defined in the sequence and in the deployment diagrams, as exemplified in Fig. 3(b) and (c). They are system assumed values, variables or expressions that will define or parameterize the formal model, i.e., the places and the transitions of the resulting TPN (described in Section V-A). When the input parameters are given as variables, then the software engineer can assign them with different values, so enabling the method to carry out the system sensitivity analysis. Expressions are functions of the input variables. The sequence diagram accommodates the annotations to define: 1) the CPU demand (*hostDemand*), in unit of time, necessary to execute a system activity, specified in the *GaStep* execution occurrences of object lifelines and 2) the size of the *GaCommStep* messages (*msgSize*), that expresses the amount of information exchanged between two objects during an interaction.

Using the VSL syntax, both kinds of annotations are specified with a pair of tuples which express a minimum $(statQ = min)$ and a maximum $(statQ = max)$ value. The CPU demands are scaled by the CPU speed factor, properly annotated in the execution node. The *msgSize* tagged-value is annotated to a message only when a *GaCommHost* is used in the communication. For example, in Fig. 3(c), this is the case of messages *m2* and *m3*, that are exchanged by the components B and C which run on different CPUs and use the LAN to communicate (Fig. 3(b)). Finally, the population of software components executing the sequence diagram is specified by stereotyping each object lifeline as *PaRunTimeInstance* and by setting the *poolSize* tag to either a value or a parameter. We assume that there is a single lifeline that starts the interaction. However, since a different pool size can be associated to each lifeline, the execution of software components can be performed sequentially or in parallel, depending of the population configuration (and, obviously, on the hardware resource restrictions). In the example, there are $N1$, $N2$, and $N3$ objects of class A, B and C, respectively. If we assume $N1 = N2, = N3 = 2$, then the two object triplets (a_1, b_1, c_1) and (a_2, b_2, c_2) execute concurrently the interactions. On the other hand, if we assume $N1 = 2$ and $N2 = N3 = 1$, like in a client-server model, then the two object triplets (a_1, b_1, c_1) and (a_2, b_1, c_1) execute the scenario almost sequentially.

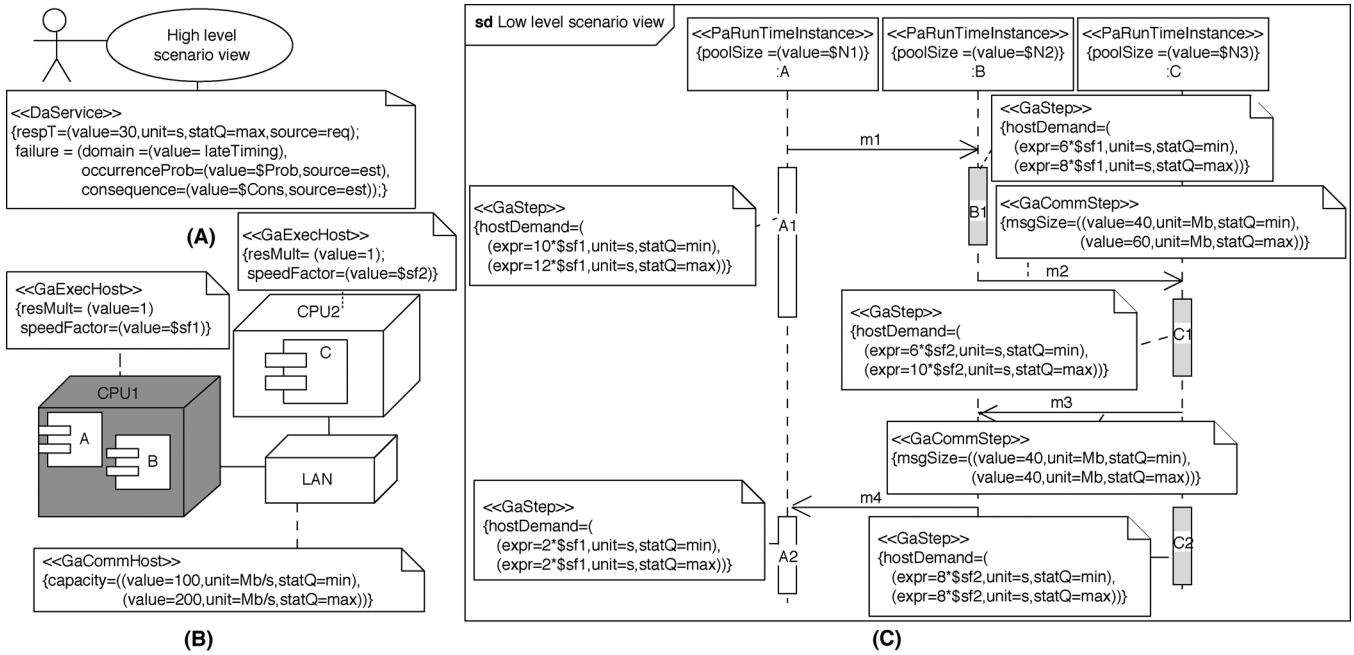


Fig. 3. Example of system specification.

The deployment diagram contains the input parameters related with the resource platform characteristics. We use the *resMult* and the *speedFactor* tags. The former specifies the multiplicity of nodes, while the latter defines the processing speed of the node as a ratio to the speed of a reference processor for the system under consideration. The network capacity (*capacity*) tag is annotated to *GaCommHost* nodes and it is expressed as an interval of values (i.e., min/max values). It will be used, in the derivation of the TPN model, to assign a time duration to the transitions representing message transmission, together with the message size tagged-values specified in the sequence diagram.

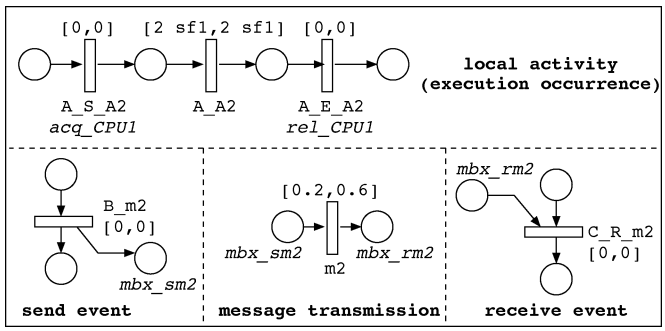


Fig. 4. TPN subnets.

V. Determine the Risks Likelihood

The goal of the risk analysis is to estimate the risk associated with a scenario timing failure. The risk depends on two factors, i.e., the probability and the consequence of the scenario timing-failure. In this section, we will define the former as a function of the required scenario response time, specified in the UC, and of the scenario response time upper and lower bounds, that are computed using the TPN bound techniques.

The construction of a TPN model of the system is a prerequisite for the estimation of the probability of timing failure. In the following, we will illustrate how to get the TPN model from the UML system specification. Sequence diagrams with alternative, optional and parallel combined fragments are also supported by our method, since the approach [37] we build on provides a translation of such constructs. However, for space reasons, we will consider only simple constructs (i.e., no combined fragments). Then, we will describe how to compute the scenario response time bounds by applying the TPN bound techniques [11], [44]. Finally, the estimation of the risk likelihood is described.

A. Derivation of the TPN Model

As discussed in Section II, we rely on the approach in [37] to get the Petri net model from the UML specification. Nevertheless, we have to customize it for different reasons. First, to include the timing specification that lead to a TPN model. Then, to represent resource contention in the system. Concerning the latter issue, we exploit the compositional properties of Petri nets to translate the sequence and the deployment diagrams in two separate TPNs, that will be then composed over common label transitions to get the final analyzable TPN model.

In the approach [37] each event (e.g., send and receive) and local activity, in the SD, is converted into a PN subnet. We consider also an additional PN subnet to model message transmission, when the communication causes delays. We have then four types of *labeled* TPN subnets, as shown in Fig. 4: the execution occurrence of the local activity $A2$, the send/receive events and the transmission related to the message $m2$. The local activity subnet is characterized by three transitions representing, respectively, the starting (`A_S_A2`), the execution (`A_A2`), and the termination (`A_E_A2`) of the activity $A2$ performed by the

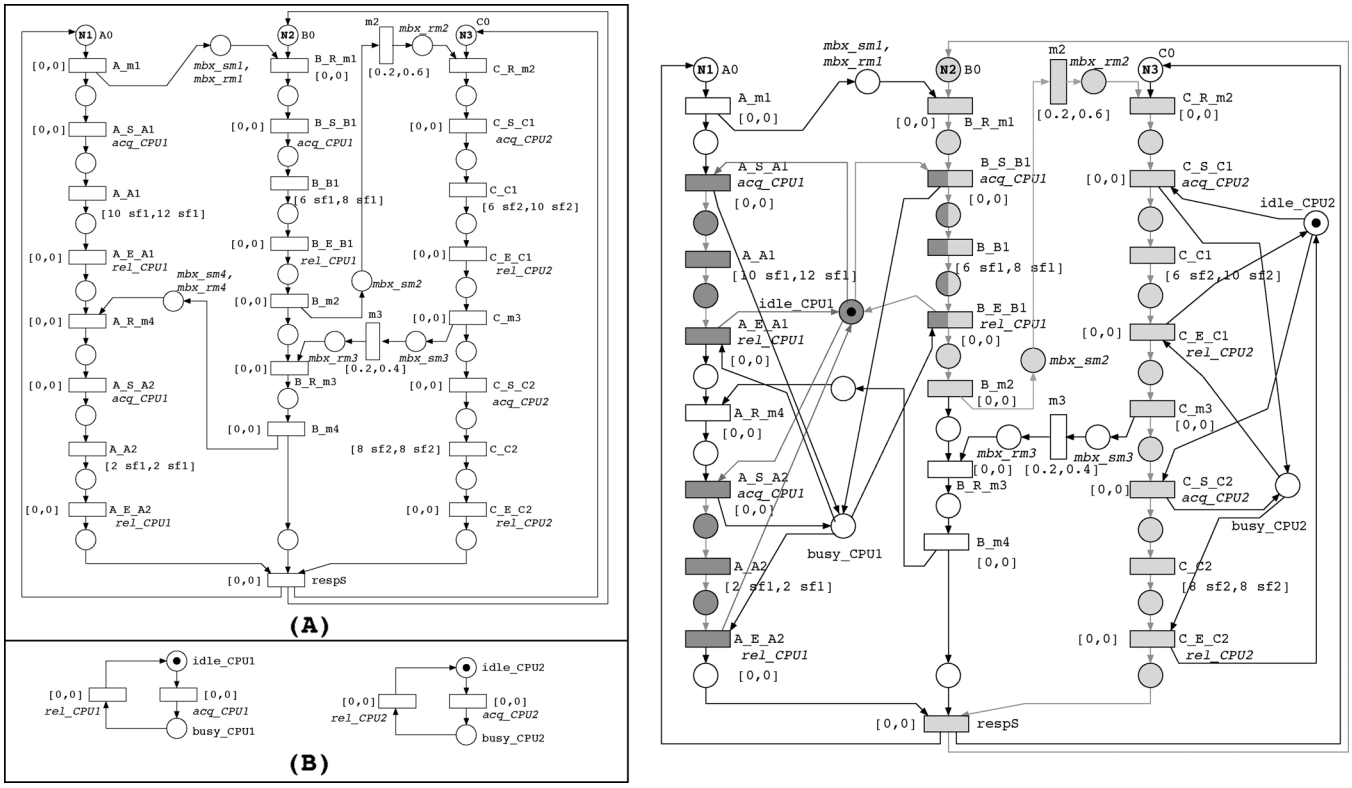


Fig. 5. Left: (a) scenario-TPN model and (b) resource-TPN model. Right: final TPN model.

component A. The send/receive event subnets are characterized, instead, by a place representing the send/receive mailbox of message $m2$ (labeled mbx_sm2 and mbx_rm2). The transmission of $m2$ is modeled by a transition with the send/receive mailbox places as input/output places, respectively. The transitions labeled as *acq_res* or as *rel_res* represent the acquisition or the release of a resource *res*, respectively, and they have no firing delay. Also, the transitions modeling the send/receive events have no firing delay associated (e.g., `B_m2,C_R_m2`). The timing specification of a transition `L_act`, representing the execution of an activity *act* performed by component L, is given by the *hostDemand* tagged values associated with the activity. For example, the transition A_A2 (Fig. 4) is characterized by the following min/max firing times:

$$\mathbf{a}[A\_A2] = hostDemand.value_{\min} = 2 \cdot sf1 \quad (1)$$

$$\mathbf{b}[A\_A2] = hostDemand.value_{\max} = 2 \cdot sf1 \quad (2)$$

where $hostDemand.value_{\min}$ and $hostDemand.value_{\max}$ are the min/max tagged-values of $hostDemand$ associated with activity $A2$, in Fig. 3(c), and $sf1$ is the speed factor parameter of the processing node *CPU1*.

The time associated to a transition `msg`, representing the transmission of a message (*msg*) through a communication resource (*res*) is given by the ratio of the *msgSize* and *capacity* associated with the message (in the SD) and to the communication resource (in the deployment diagram), respectively. Since both the message size and the net capacity parameters are expressed as min/max tagged values, the division operator of interval arithmetic for non-negative intervals is used [45]. For example, the transition $m2$ (Fig. 4) is characterized by the following min/max firing times:

$$\mathbf{a}[m2] = \frac{msgSize.value_{\min}}{capacity.value_{\max}} = 0.2 \quad (3)$$

$$\mathbf{b}[m2] = \frac{msgSize.value_{\max}}{capacity.value_{\min}} = 0.6 \quad (4)$$

where $msgSize.value_{\min} = 40$ and $msgSize.value_{\max} = 60$ are the min/max tagged-values of $msgSize$ associated with message $m2$, in Fig. 3(c), and $capacity_{\min} = 100$ and $capacity_{\max} = 200$ are the min/max tagged-values of $capacity$. In the formulas, we assume tagged values with homogeneous metric units.

The TPN subnets are then connected via source/sink places, according to the partial order sequencing of the corresponding events, as proposed by [37], then obtaining a TPN model (*scenario-TPN model*) that captures the *weak sequencing of events* [8] as well as the timing delays due to the execution of local activities and to the message transmission. The scenario-TPN model for the SD, in Fig. 3(c), is shown in Fig. 5(a). The initial marking of the scenario-TPN model is defined according to the population specification annotated to each object lifeline. Then, the places `A0,B0,C0` are characterized by an initial marking equal to the *poolSize* tagged values of lifelines A, B and C, respectively (i.e., the parameters $N1$, $N2$, and $N3$). A transition `respS` (with zero firing delay) is also added, that brings back the model to its initial marking after a scenario execution.

The deployment diagram provides indications on the system hardware resources, then a resource-TPN model will be derived. Fig. 5(b) shows the resource-TPN model of the deploy-

ment diagram in Fig. 3(b). A resource-TPN model consists of a set of disjoint TPN subnets, one for each *GaExecHost* node. A TPN subnet models the behavior of a hardware resource *res*. It consists of two places, `idle_res` and `busy_res`, and two causally connected transitions, labeled as *acq_res* and *rel_res*, that represent the resource acquisition and release (with zero firing delay). The place `idle_res` is initially marked, and its marking is set to the multiplicity tagged-value associated with the node.

The final analyzable TPN model is obtained by composing the scenario-TPN model and the resource-TPN model over transitions with equal label. The transition composition operator [46] is the usual one that generates the Cartesian product of the transitions with common labels of the two composing nets. The synchronized transitions may represent either resource acquisition (transitions labeled as *acq_res*) or resource release (transitions labeled as *rel_res*). The final analyzable TPN model of the example in Fig. 3(b) and (c) is depicted in Fig. 5 (right side): it has been obtained by composing the TPNs in Fig. 5(a) and (b) over transitions with common labels (*acq_CPU1*, *res_CPU1*, *acq_CPU2*, *res_CPU2*).

B. Computation of Scenario Response Time Bounds

The TPN model, derived from the sequence and deployment diagrams, is characterized by *good* properties such as boundedness and liveness, and it is used to compute the bounds of the scenario response time. When the scenario is characterized by a single tuple of interacting components (that is one object per lifeline), the scenario response time corresponds to the time duration between two consecutive firings of the synchronization transition $respS$ (namely, the inter-firing time $\Gamma[respS]$), whose firing models the termination of the scenario execution by the tuple. When, instead, several tuples of interacting components execute concurrently, we consider the number of objects associated to the lifeline of reference, which corresponds to the one that starts the interaction. In the example of Fig. 3, the lifeline of reference is A and it is populated with $N1$ objects. The scenario execution is completed when all the $N1$ tuples have terminated, that is after $N1$ consecutive firings of $respS$.

The lower bound of $\Gamma[respS]$ is computed by applying techniques based on the Little's law. In particular, when the TPN has the unique minimal T-semiflow[1] $\mathbf{x} = \mathbf{1}$ (*1-consistency*), all the transitions are characterized by the same visit ratio and we can state and solve the following linear programming problem (LPP):

$$\begin{aligned} \Gamma^L[respS] = maximum \quad & \mathbf{y} \cdot \mathbf{Pre} \cdot \mathbf{a} \\ subject\ to \quad & \mathbf{y} \cdot \mathbf{C} = 0 \\ & \mathbf{y} \cdot \mathbf{M_0} = 1 \\ & \mathbf{y} \geq 0 \end{aligned} \tag{5}$$

where $\mathbf{y}$ is a vector of variables, $\mathbf{Pre}$ and $\mathbf{C}$ are the pre- and incidence matrices, respectively, $\mathbf{a}$ is the transition *static earliest firing time* vector, and $\mathbf{M_0}$ is the initial marking vector. The LPP (5) is a straightforward extension of the one stated for TPN marked graphs in [44] to one-consistent TPNs.

The upper bound of $\Gamma[respS]$ is computed, instead, considering a complete sequentialization of all the timed transition firings

$$\Gamma^U[respS] = \sum_{t \in T} \mathbf{b}[t] \tag{6}$$

where $\mathbf{b}$ is the transition *static latest firing time* vector.

The lower bound $\Gamma^L[respS]$ represents the inter-firing time of $respS$ under *best case behavior* assumption, that is considering the real concurrency among the component activities and the transition minimum firing delays and, in particular, depends on the initial marking. The upper bound $\Gamma^U[respS]$ represents, instead, the inter-firing time of $respS$ under *worst case behavior* assumption, and it does not depend on the initial marking. The scenario response time lower and upper bounds are then given by

$$RT^L(N) = N \cdot \Gamma^L[respS] \tag{7}$$

$$RT^U(N) = N \cdot \Gamma^U[respS] \tag{8}$$

where N is the number of objects associated to the lifeline of reference.

When the sequence diagram contains alternative/optional fragments, we resort to general LPP techniques to compute the bounds [11]. It is worth noting that the time required to compute the scenario response time bounds does not depend on either the number of tuples of interacting objects or the number of hardware resources (i.e., the TPN initial marking). Indeed, the LPP techniques used to compute the bounds have polynomial time complexity on the net size, i.e., number of places and transitions.

C. Risk Likelihood Estimation

The risk likelihood is estimated as the probability that the scenario response time is not greater (less) than the maximum (minimum) timing constraint annotated in the UC. Similarly to [27], we define such a probability as a function of the required response time and the scenario response time bounds, previously computed. In particular, when a maximum is specified, such as in the UC in Fig. 3(a), we use the following formula:

$$P_{fail}(x) = \begin{cases} 0, & \text{if } x > RT^U \\ \frac{RT^U - x}{RT^U - RT^L}, & \text{if } RT^L \leq x \leq RT^U \\ 1, & \text{otherwise} \end{cases}. \tag{9}$$

The probability value in the extreme cases is reasonable by definition of bounds. Indeed, if the maximum value is greater than the upper bound, the (unknown) response time will be always lower than the maximum specified, hence the probability of timing failure is zero. On the other hand, if the maximum value is lower than the lower bound, the (unknown) response time will be always greater than the maximum specified, so the probability of timing failure is set to 1. When, instead, the maximum requirement falls in the interval $x \in [RT^L, RT^U]$, we estimate the probability of timing failure as the ratio between the distance of the upper bound from the maximum value (i.e., failure range) and the distance between the bounds (i.e., whole range of

[1]T-semiflows (P-semiflows) are right (left) integer natural annullers of the TPN incidence matrix.

values). This ratio corresponds to consider each response time value, within the bound interval, with the same probability: this is a reasonable assumption when there is absence of information on the time duration distribution of the system activities, as in our case. Indeed, for each activity only min/max durations are given as input to the specification. Formula (9), and correspondingly $1 - P_{fail}(x)$ when the specification indicates a minimum threshold, corresponds to assume that the response time is uniformly distributed between the lower and upper bounds. The main reasons of this choice has been to provide an efficient and not trivial risk estimation technique for the early stages of the software life-cycle. In fact, concerning the efficiency, we resort to the TPN bound techniques and formula (9) that are characterized by low computational costs and, in particular, scale very well with respect to the number of software and hardware resources in the system. Unlike the exact stochastic analysis techniques, such as [42], that require the knowledge of probability distributions of the single activity durations, the proposed approach can be used early in the life-cycle, which is often characterized by uncertainties due to the lack of complete knowledge of the whole system and of the external factors. From the usefulness point-of-view, the uniform assumption permits to provide not trivial estimations with respect to the deterministic one. Indeed, if we assume a deterministic distribution, then $P_{fail}(x)$ can be either set to 1 or 0 depending whether the maximum value specified x is less than the upper bound RT^U or not. This assumption leads to consider only the boundary categories of the risk likelihood classification, e.g., in Table I either *frequent* or *impossible*.

As system measures become available, later in the life-cycle, other assumptions on the type of distribution over the interval $[RT^L, RT^U]$ could be more appropriate. In order to analyze the sensitivity of the risk likelihood, with respect to the distribution assumption, we have compared the $P_{fail}(x)$ of the running example under the uniform and normal hypothesis. In particular, according to the 2σ and 3σ-rules [47], the following normal distributions are considered:

1) $\mathcal{N}(\mu, \sigma_2)$, where $\mu = (RT^U + RT^L)/2$ and $\sigma_2 = (RT^U - RT^L)/4$ (i.e., 2σ-rule);
2) $\mathcal{N}(\mu, \sigma_3)$, where $\sigma_3 = (RT^U f - RT^L)/6$ (i.e., 3σ-rule).

Default values are set to the CPU speed factors, (i.e., $sf1 = sf2 = 1$), the lifeline A is populated with $N1 \in [1 \dots 10]$ objects and one object is assigned to each lifeline B and C. First, we have observed that the difference between the $P_{fail}(x)$ values under the uniform and the normal cases is not affected by the bound interval; in particular, the maximum error is 9.15% for the 2σ-rule (18.66% for the 3σ-rule) independently of the pool size value $N1$. Fig. 6 plots the three curves of $P_{fail}(x)$, when $N1 = 1$. Second, the uniform assumption can either underestimate or overestimate $P_{fail}(x)$, with respect to the normal assumption, depending whether the maximum value specified x is less or (respectively) greater than the mean value μ. While the probability value is the same when $x = \mu$. In Fig. 3(a), $x = 30 < \mu$, and the probability of the scenario timing-failure is $P_{fail}(30) = 0.5288$ under the uniform assumption. It is an underestimate with respect to the normal assumption with a difference of 1.71% (2σ-rule) and 3.99% (3σ-rule). Finally, the difference in the risk likelihood evaluation depends not only on the maximum value specified but also on the mapping of the probability value onto the qualitative property according to the defined likelihood classification. Indeed, considering the one defined in Table I, under both the uniform and the normal assumptions the risk likelihood of the running example is finally evaluated as *frequent*.

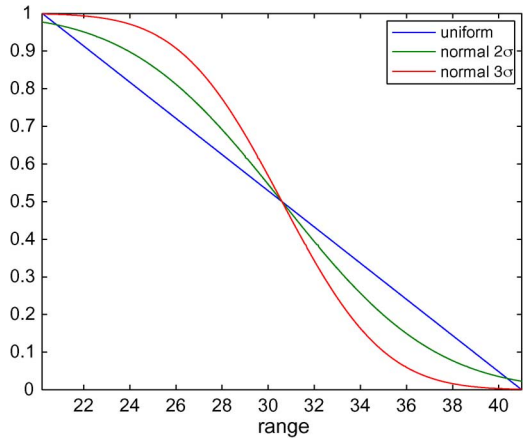


Fig. 6. Sensitivity of $P_{fail}(x)$ under different uniform and normal distributions ($RT^L = 20.2$, $RT^U = 41$).

VI. Evaluate and Treat the Risks

Beside the risk likelihood, the consequence associated with a system failure should be determined also. The consequence is a characterization of the impact of the failure on the system users and environment. Many standard techniques exist that support the consequence analysis [12]–[14]. We resort to those techniques that can be applied early in the software life-cycle (such as Preliminary Hazard Analysis [13] or Functional Hazard Assessment [48] or Functional Failure Analysis [49]). Actually, the only condition we impose to the selected technique concerns with the consequence classification that should correspond to the ones in the adopted risk matrix: e.g., in Table I, the following categories are considered: *negligible*, *marginal*, *critical*, and *catastrophic* [13], [14].

Once the two risk factors have been estimated, the risk evaluation activity consists in verifying whether the estimated risk associated with the scenario timing-failure is tolerable, with the help of the risk matrix in Table I. In case of positive result, then the risk assessment for the current scenario can be considered terminated and we can repeat the risk assessment process for other system scenarios. On the other hand, when the estimated risk is not tolerable, the UML design needs to be further analyzed to identify the potential causes of the risk and, in case, undertake corrective actions.

The first step of the risk treatment consists in identifying the *critical* elements, in the UML design, that contribute to the risk estimated during the analysis.

For this purpose, we exploit the TPN bound technique used for the computation of the scenario response time lower bound. In particular, the optimal solution of the LPP (5), stated for

the TPN system, is a minimal P-semiflow[2] and the TPN subnet generated by its support is the *slowest* subnet $\mathcal{N}^*$ of the TPN system. In fact, it has the highest cycle time (i.e., the sum of the minimum firing delays of its transitions) that corresponds to the computed bound.

Considering the backward mapping of the TPN model onto the UML design, we define the demands associated to a lifeline L of the SD, to message transmission, and to a resource r of the DD, respectively, as follows:

$$D_L = \sum_{L\_act_j \in T^*} \mathbf{a}[L\_act_j]$$

$$D_{tr} = \sum_{m_k \in T^*} \mathbf{a}[m_k]$$

$$D_r = \begin{cases} \sum_{L_i\_act_j \in T^*} \mathbf{a}[L_i\_act_j] & \text{if } idle\_r \in P^* \\ 0 & \text{otherwise} \end{cases}$$

where transitions L_act_j ($L_i\_act_j$) model local activities act_j of L (L_i), transitions m_k model the transmission of messages m_k, T^*, and P^* are the sets of transitions/places of the slowest subnet $\mathcal{N}^*$. Then, it is possible to identify the software components (i.e., object lifelines) and resources with the highest demand. Moreover, $\mathcal{N}^*$ may correspond to one of the following critical elements in the UML design.

- **Software components**. The set of timed transitions $\{L\_act_j\} \subseteq T^*$ represent the local activities executed by a single lifeline L. Then, the critical elements are the software components represented by the lifeline L, since they are the components with the highest demand.
- **Hardware resource**. The set of timed transitions $\{L_i\_act_j\} \subseteq T^*$ represent the lifeline local activities in the sequence diagram which require a hardware resource r to be executed. Then, the critical element of the UML design is the node r in the deployment diagram, since it is the most demanded resource within the considered scenario.
- **Interaction path**. The set of timed transitions $\{L_i\_act_j, m_k\} \subseteq T^*$ represent local activities executed by different lifelines L_i and messages transmission. Then, the critical element is the most time consuming path in the scenario.

It is worth noting that the slowest TPN subnets are sensitive not only to the change of the timing parameters but also to the population of software components executing the sequence diagram. In the running example, assuming the default values for the two CPU speed factors and a single object per lifeline, the slowest TPN subnet (Fig. 5, on the right, in light gray) corresponds to the scenario longest path emphasized in light in Fig. 3(c). The component with highest demand in the scenario longest path is C since its demand $D_C = \mathbf{a}[C\_C1] + \mathbf{a}[C\_C2] = 14$ s, while $D_B = \mathbf{a}[B\_B1] = 6$ s and $D_A = 0$. On the other hand, if the lifelines A and B are populated with two objects, the slowest TPN subnet (Fig. 5, on the right, in dark gray) identifies the *CPU1* in Fig. 3(b) as the critical resource, with $D_{CPU1} = 18$ s. The high-risky components that use *CPU1* are the A objects with $D_A = \mathbf{a}[A\_A1] + \mathbf{a}[A\_A2] = 12$ s, while $D_B = \mathbf{a}[B\_B1] = 6$ s and $D_C = 0$.

[2]A semiflow is *minimal* when its support, i.e., the set of the nonzero components of the incidence matrix annuller, is not a proper superset of the support of any other, and the greatest common divisor of its elements is one.

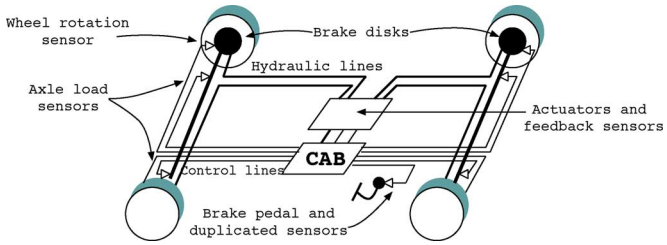


Fig. 7. CAB system context.

The second step in the risk treatment offers some simple guidelines to reduce the probability of the scenario timing-failure. These guidelines, that have to be applied in the UML design, give advice on how to remove or alleviate the identified *critical* elements. When the critical element is a hardware resource, the simplest actions concern either with the replacement with a faster one or with the increment in its multiplicity, i.e., with its replication. The activities carried out by the software components with the highest demands should be studied in the sequence diagram trying to reduce their delays and demands and/or trying to execute them in parallel whenever possible.

In the case of a path in the scenario that goes beyond the timing constraint, then the previous advice can be applied to each software component participating in the path. Moreover, an alternative network configuration could be studied in the deployment diagram for the hardware nodes hosting the involved software components and/or incrementing the capacity of the networks, these actions will tend to reduce the demands of the messages exchanged in the longest path. After some of these changes have been introduced in the UML design, the method has to be applied again to check the success.

VII. A Case Study From the Real-Time and Embedded System Domain

Even if the proposed method is more fitted into soft real-time domain, we will apply it to a hard real-time one, a Computer-Assisted Braking (CAB) system for vehicles introduced in [50]. The intention is not on obtaining so accurate results, but on producing a blueprint for a complex and realistic case study. CAB provides three functionalities in addition to traditional vehicle brakes: 1) *Anti-lock braking* (ABS), that detects the onset of wheel lock up and releases the brakes to allow the wheel to turn and regain grip. 2) *Emergency stop detection and enhancement*, that detects the rapid pedal movement associated with an emergency stop and maximizes the braking used. 3) *Load-compensated braking* that measures the weight on the vehicle's suspension to ensure that a given pressure on the brake pedal provides the same degree of braking, regardless of how heavily the vehicle is loaded, or how the load is distributed.

Fig. 7 shows the context of the CAB system. Each hydraulic line is fitted on to each wheel, with a feedback pressure sensor for closed-loop control. The brake pedal has two sensors, each returning a value indicating how hard the pedal has been pressed. Axles of the vehicle have two pressure transducers to

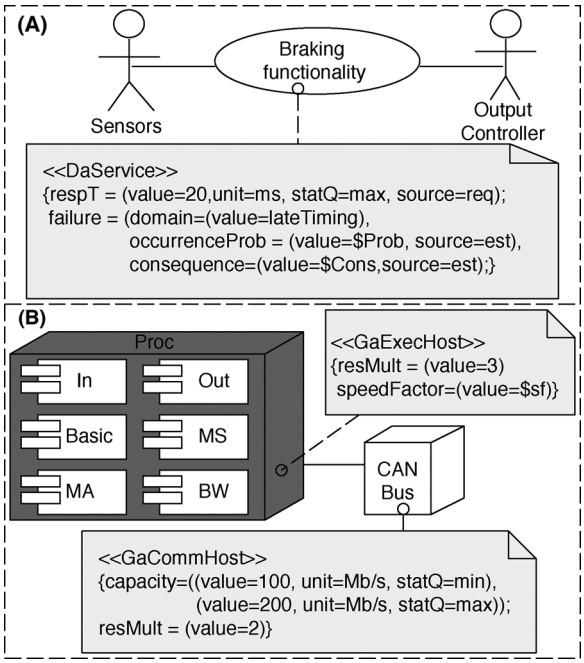


Fig. 8. CAB use (a) case diagram and (b) deployment diagram.

measure the load on the vehicle. Finally, each wheel has a rotation sensor to be used for lookup detection for anti-lock braking function. Two output controllers drive the hydraulic actuators. Each one controls the actuators for a diagonal pair of wheels, takes required commands over a duplicated Controller Area Network (CAN) bus link, and converts them to the required electrical outputs to the actuators. Each controller works on a cyclic basis, and will only alter its outputs once per period of the cycle. Mauri [50] presents a detailed safety assessment of the CAB by applying several standard techniques (e.g., PHA and FMEA). Moreover, he points out the importance of timing assessment and clearly establish the requirements for the CAB at this regard, however, he prefers not to report results since his aim just targets safety concerns.

A. Establish the Context and Identify the Risk

CAB has to meet timing requirements derived from vehicle dynamics. In particular, the latency from the pedal movement to brake effect should be at most 20 ms. So, we will assess the risk due to a late response time from demand to brake effect. Thereby, we consider the CAB specification in [50], from which we modeled the architecture and behavior using UML. Consider that we left out, only and intentionally, the specification of the safety requirements also reported in [50].

In Fig. 8(a), use case annotation captures the timing requirement: the risk (i.e., late timing failure) as well as the two risk factor parameters (i.e., probability and consequence of a scenario timing failure). Deployment diagram in Fig. 8(b) represents the CAB redundant architecture: there are three identical processor nodes, $Proc_i$ $(i = 1, 2, 3)$, that communicate via a duplicated CAN bus. The buses are also used for sending output values to the output modules.

Fig. 9 shows the braking scenario of the CAB system. The scenario represents the concurrent execution of the components running on processor $Proc_i$, namely, channel i, as well as interactions between the modifier selector MS_i with the bus watchers running on the three processors BW_j $(j = 1, 2, 3)$. It starts with component In_i which reads all sensor values and uses data from both pedal sensors to form a single pedal value (*R_E*). Then, In_i sends the outputs to the $Basic_i$, the modifier selector MS_i and the Out_i components. $Basic_i$ calculates a basic braking pressure for each wheel based only on the pedal sensor value (*BBP*) and sends a record containing the four values to MS_i. The latter uses all the sensor information to determine which modifiers are required (*CAS*), calculates modifier values and creates a record containing basic and modifier values in pool (*MV*) to be used by the modifier adder MA_i. MS_i broadcasts then the votes to all the bus watcher components, running on the three processors, in order to identify which modifiers are required. Observe that a *msgSize* tagged-value has been assigned to the *Vote* messages sent by the MS_i to the bus watchers BW_j $(i \neq j)$, since the CAN buses are used in the communication. No annotation has been assigned, instead, to the *Vote* message sent to the BW_i, since the latter runs on the same processor of the sender. The BW_i builds up a record of votes (*BRV*) and sends all the votes to the MA_i. The latter determines which modifiers to add to the basic braking value from the record of all votes assembled by the BW_i (*Vot*) and sends the final braking pressure values to the Out_i. Finally, Out_i calculates the braking pressure (*BR*), adjusted according to sensor feedback, and sends the braking actuator drive values to the two output controllers.

B. Determine the Risk Likelihood

During risk analysis, the risk factor parameters (probability and consequence) will be estimated. The first step consists in deriving a TPN model (Fig. 10) from the UML specification (Figs. 8 and 9) according to our method. To improve readability, broken arcs are used in Fig. 10, where the name associated with a broken arc is the name of the transition/place from which the arc is originated or to which the arc is addressed (e.g., *idle_Proc* is an input place of transition *IN_S_R_E*). Immediate transitions are drawn as thin black bars, while timed transitions as thick white bars. The CAB resource is represented by a pair of places (*idle_Proc*, *busy_Proc*), which are either input or output for a set of transitions like *IN_S_R_E*, *IN_E_R_E* and so on. It is worth noting that place *idle_Proc* allows parallel execution of the three processors. Finally, the slowest subnet of the TPN is emphasized in gray and it will be discussed in the next subsection.

Considering the interval-based annotations in the UML models (Figs. 8 and 9), the interval firing times of timed transitions were calculated according to the mapping described in Section V-A. In particular, the min/max firing times assigned to transitions that represent message transmission were calculated considering formulas (3) and (4).

We have used the TPN-PerfBound tool [51] to compute the upper and lower bound inter-firing times for transition *respS*. In particular for the lower bound computation, the generated LPP is characterized by 64 variables and 46 equality constraints. The automatic generation and solution of the LPP took few seconds on a Pentium 4 PC with 1.60 GHz CPU. The lower and upper bound response times of the braking scenario were calculated

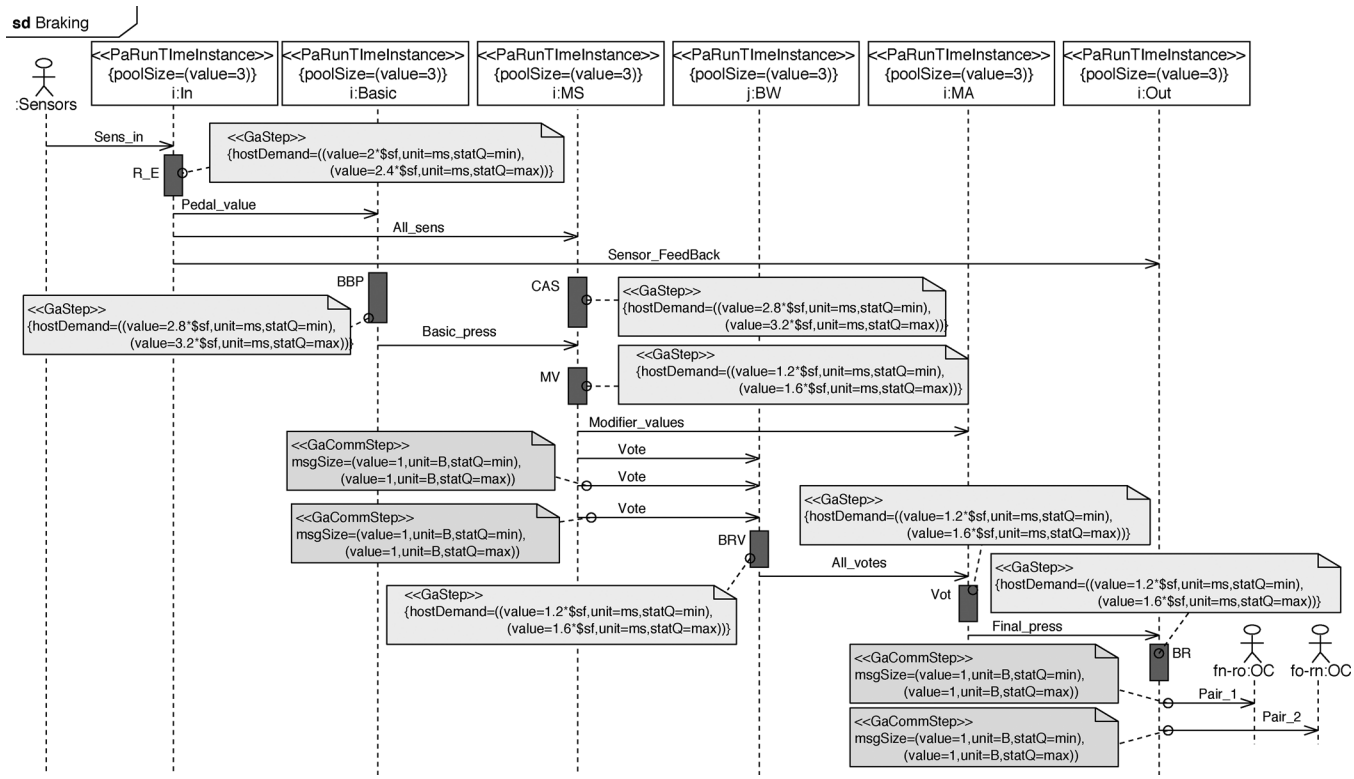


Fig. 9. CAB braking scenario.

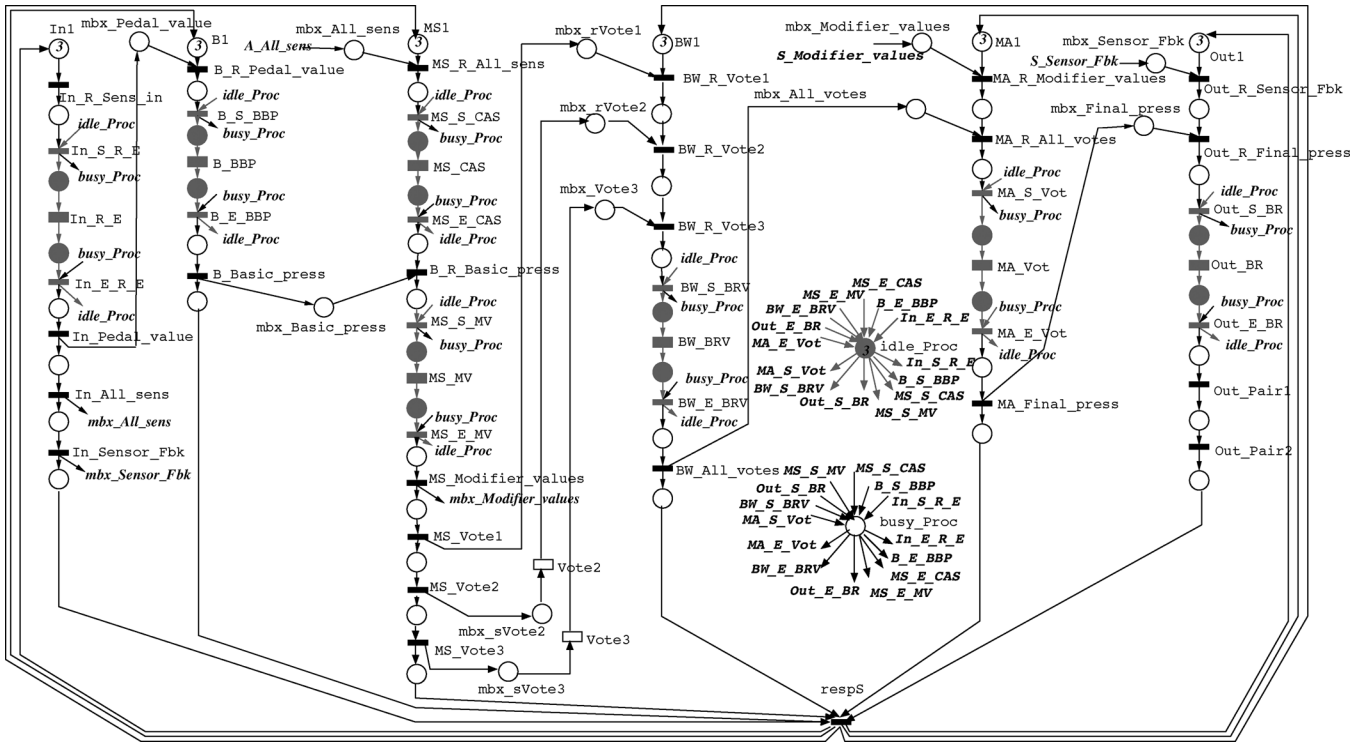


Fig. 10. TPN model of the CAB system.

by applying formulas (7) and (8) in Section V-B, respectively, then obtaining $RT^L(3) = 12.4$ ms and $RT^U(3) = 45.6$ ms.

The timing requirement states that the response time of the braking scenario should be at most 20 ms. We can then estimate the probability of timing failure by using formula (9) in Section V-C: $Prob = (45.6 - 20)/(45.6 - 12.4) = 0.77$. According to Table I, this value falls in the *frequent* likelihood category.

The preliminary hazard analysis of the CAB system, carried out in [50], led to the classification of the timing-failure mode of the braking scenario as *critical*.

C. Evaluate and Treat the Risk

The estimate of the timing-failure risk corresponds to an intolerable risk considering Table I, so it needs treatment. We then proceed with the identification of the critical elements in the UML design to treat the risk. In particular, from the solution of the LPP (5) stated for the computation of the lower bound, it is possible to identify the slowest subnet of the TPN model. Observe that the slowest subnet (in gray in Fig. 10) contains the timed transitions representing the local activities in the CAB braking scenario (in gray in Fig. 9) which are executed by the components running on the same processor. Then, the processors modeled by the node *Proc* in the deployment diagram in Fig. 8(b) are the critical elements in the UML design since they have the highest demand.

By applying guidelines in Section VI, a trivial solution for risk reduction is to use processors with higher speed. For example, if we double the processor speed and apply the risk analysis method again, we obtain a risk estimation of 0.13. According to Table I, that risk would still be intolerable. Applying the method once more, it can be computed that a speed factor of 2.3 is needed in order to reduce the likelihood under the threshold of 10^{-5}, thus leading to a negligible risk.

VIII. Conclusion

In this paper, we have applied best practises in software engineering (i.e., UML, profiling and model driven transformation) and well-known formal techniques in the literature (i.e., Time Petri Nets and bound techniques) to propose a comprehensive method for assessing the risk of timing failure by evaluating the software design. The main contribution of this work is the customization of the activities proposed by the standard risk management process [7] for the assessment of timing-failure risks, in early stages of the software life-cycle.

Our system context is the UML-based software specification, enriched with MARTE [9] profile annotations to capture the nonfunctional system properties. During the risk analysis, we exploit the TPN and bound techniques to estimate the probability of scenario timing-failure. First, a TPN model is derived from a UML-based design, according to a transformation approach which is an adaptation of the Eichner's one [37]. Later, the TPN bound techniques are applied to compute the scenario response time bounds, which are used then to estimate the probability of timing-failure. Within the risk evaluation and treatment steps, our method provides a support in the identification of the critical elements in the design which contribute to the timing-failure risk of the system scenario, i.e., resource bottlenecks, most time consuming paths in the scenario and software components with the highest demand.

We have developed a TPN tool [51] for the bound computation and the identification of the slowest subnet of a TPN model, that has been used in the running example and in the case study. Currently, we are implementing the most time-consuming step, namely, the derivation of the TPN model from the UML system specification.

With respect to the accuracy of the computed bounds, a general theoretical result has not been obtained [11]. Nevertheless, as in the case of qualitative structural theory of net systems, the derived performance-oriented results are specially powerful for some well-known subclasses of nets, like the ones obtained by applying the UML2TPN translation of UML sequence and deployment diagrams proposed in this work. For more general classes, the quality of the bounds could be poor, specially for throughput lower bounds.

Acknowledgment

The authors would like to thank the anonymous referees and the associate editor for their helpful comments to improve the paper.

References

[1] E. Lazowska, J. Zahorjan, G. Scott Graham, and C. Sevcik, *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*. New York: Prentice-Hall, 1984.

[2] R. Alur and D. Dill, "A theory of timed automata," *Theor. Comput. Sci.*, no. 126, pp. 183–235, 1994.

[3] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis, *Modelling With Generalized Stochastic Petri Nets*, ser. John Wiley Series in Parallel Computing. Chichester, U.K.: Wiley, 1995.

[4] B. Berthomieu and M. Diaz, "Modeling and verification of time dependent systems using time petri nets," *IEEE Trans. Soft. Eng.*, vol. 12, no. 3, pp. 259–273, Mar. 1991.

[5] P. Merlin and D. Faber, "Recoverability of communication protocols," *IEEE Trans. Commun.*, vol. COM-24, no. 9, pp. 1036–1043, Sep. 1976.

[6] Object Management Group, [Online]. Available: http://www.omg.org

[7] *Australian Standard AZ/NZS4360: Risk Management*, AZ/NZS4360:, 1999.

[8] *Unified Modeling Language: Superstructure* OMG, May 2010, ver. 2.3, formal/10-05-05.

[9] *A UML Profile for Modeling and Analysis of Real Time Embedded Systems (MARTE),* OMG, 2009, document ptc/09-11-02.

[10] A. Avizienis, J. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, Jan.–Mar. 2004.

[11] S. Bernardi and J. Campos, "Computation of performance bounds for real-time systems using time petri nets," *Trans. Ind. Informat.*, vol. 5, no. 2, May 2009.

[12] "IEC-60300-3-1: Dependability Management," 2001, International Electrotechnical Commission, 3 rue de Varembé CH 1211. Geneva, Switzerland.

[13] "MIL-STD-882: System Safety Program Requirements," 1999, Dept. Defense Military Standard, USA.

[14] N. Leveson, *SAFEWARE: System Safety and Computers*. Reading, MA: Addison-Wesley, 1995.

[15] M. Stamatelatos and H. Dezfuli, "Probabilistic Risk Assessment Procedures Guide for NASA Managers and Practitioners," Aug., 2002, ver. 1.1, Prepared for Office of Safety and Mission Assurance, NASA Headquarters. Washington, DC.

[16] C. Rodger and J. Petch, "Uncertainty & Risk Analysis," Apr. 1999, Business Dynamics, PriceWaterHouseCoopers.

[17] *UML Profile for Modeling Quality of Service and Fault Tolerant Characteristics and Mechanisms,* OMG, Apr. 2008, ver. 1.1, formal/08-04-05.

[18] *Information Technology. Security Techniques: Information Security Risk Management*, iSO/IEC 27005:2008, International Electrotechnical Commission, 2008.

[19] *NIST National Vulnerability Database*, , National Institute of Standards and Technology. [Online]. Available: http://nvd.nist.gov/

[20] *BSI Standard 100-1: Information Security Management Systems (ISMS)*, Standard 100-1, Federal Office for Information Security. [Online]. Available: http://www.bsi.de/english/gshb/

[21] E. Zambon, S. Etalle, R. J. Wieringa, and P. Hartel, "Model-based qualitative risk assessment for availability of IT infrastructures," *Softw. Syst. Modeling*, 10.1007/s10270-010-0166-8, to appear.

[22] F. den Braber, I. Hogganvik, M. Lund, K. Stolen, and F. Vraalsen, "Model-based security analysis in seven steps. A guided tour to the CORAS method," *BT Technol. J.*, vol. 1, no. 25, pp. 101–117, 2007.

[23] C. Alberts and A. Dorofee, "OCTAVE Criteria," Carnegie Mellon-Software Engineering Institute, Pittsburgh, PA, Tech. Rep. ESC-TR-2001-016, Dec. 2001.
[24] "EBIOS: Expression des Besoins et Identification des Objectifs de Sécurité," Agence Nationale de la Sécurité des Systèmes d'information. [Online]. Available: http://www.ssi.gouv.fr/en/
[25] "CRAMM v5.1 Information Security Toolkit," Siemens [Online]. Available: http://www.cramm.com
[26] K. Goseva-Popstojanova, A. E. Hassan, A. Guedem, W. Abdelmoez, D. E. M. Nassar, H. H. Ammar, and A. Mili, "Architectural-level risk analysis using UML," *IEEE Trans. Softw. Eng.*, vol. 29, no. 10, pp. 946–960, 2003.
[27] V. Cortellessa, K. Goseva-Popstojanova, K. Appukkutty, A. Guedem, A. Hassan, R. Elnaggar, W. Abdelmoez, and H. Ammar, "Model-based performance risk analysis," *IEEE Trans. Softw. Eng.*, vol. 31, no. 1, pp. 3–20, Jan. 2005.
[28] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni, "Model-based performance prediction in software development: A survey," *IEEE Trans. Softw. Eng.*, vol. 30, no. 5, pp. 295–310, May 2004.
[29] I. Ober, S. Graf, and I. Ober, "Validating timed UML models by simulation and verification," *Int. J. Softw. Tools for Technol.*, vol. 8, no. 2, pp. 128–145, 2006.
[30] S. Bernardi, J. Merseguer, and D. Petriu, "A dependability profile within MARTE," *J. Softw. Syst. Modeling*, 10.1007/s10270-009-0128-1, to appear.
[31] P. King and R. Pooley, "Derivation of Petri Net performance models from UML specifications of communications software," in *Proc. 15th U.K. Performance Engineering Workshop, Specifications of Communication Software,* , 2000, pp. 262–276, Springer.
[32] S. Bernardi, S. Donatelli, and J. Merseguer, "From UML sequence diagrams and statecharts to analyzable Petri Net models," in *Proc. 3rd Workshop on Software and Performance (WOSP'02)*, Roma, Italy, July 2002, pp. 35–45, ACM.
[33] M. Woodside, D. Petriu, D. Petriu, H. Shen, T. Israr, and J. Merseguer, "Performance by Unified Model Analysis (PUMA)," in *Proc. 5th Int. Workshop on Software and Performance (WOSP'05)*, Palma de Mallorca, Spain, Jul. 2005, pp. 1–12.
[34] S. Bernardi and J. Merseguer, "Performance evaluation of UML design with stochastic well-formed nets," *J. Syst. Softw.*, vol. 80, no. 11, pp. 1843–1865, Nov. 2007.
[35] J. Cardoso and C. Sibertin-Blanc, "Ordering actions in sequence diagrams of UML," in *Proc. 23th Int. Conf. Inform. Technol. Interfaces*, Pula, Croatia, 2001, pp. 3–14.
[36] O. Kluge, "Petri Nets as a semantic model for message sequence charts specification," in *Proc. 2nd Int. Workshop on Integration of Specification Techniques for Applications in Engineering (INT02)*, Grenoble, France, Apr. 2002, pp. 138–147.
[37] C. Eichner, H. Fleischhack, R. Meyer, U. Schrimpf, and C. Stehno, "Compositional semantics for UML 2.0 sequence diagrams using Petri Nets," in *Proc. 12th Int. SDL Forum, SDL 2005: Model Driven*, A. Prinz, R. Reed, and J. Reed, Eds., Grimstad, Norway, Jun. 20–23, 2005, vol. 3530, Lecture Notes in Computer Science, pp. 133–148.
[38] J. Fernandes, S. Tjell, J. Jørgensen, and O. Ribeiro, "Designing tool support for translating use cases and UML2.0 sequence diagrams into a Coloured Petri Net," in *Proc. 6th International Workshop on Scenarios and State Machines (SCESM07)*, Minneapolis, MN, 2007, p. 2.
[39] D. Petriu and M. Woodside, "An intermediate metamodel with scenarios and resources for generating performance models from UML designs," *Softw. Syst. Modeling*, vol. 6, no. 2, pp. 163–184, 2007, 10.1007/s10270-006-0026-8.
[40] E. Vicario, "Static analysis and dynamic steering of time-dependent systems," *IEEE Trans. Softw. Eng.*, vol. 27, no. 8, pp. 728–748, Aug. 2001.
[41] D. Xu, X. He, and Y. Deng, "Compositional schedulability analysis of real-time systems using Time Petri Nets," *IEEE Trans. Softw. Eng.*, vol. 28, no. 10, pp. 984–996, Oct. 2002.
[42] E. Vicario, L. Sassoli, and L. Carnevali, "Using stochastic state classes in quantitative evaluation of dense-time reactive-time reactive systems," *IEEE Trans. Softw. Eng.*, vol. 35, no. 5, pp. 703–719, Nov. 2009.
[43] D. Cancila, F. Terrier, F. Belmonte, H. Dubois, H. Espinoza, S. Girard, and A. Cucurru, "Sophia: A modeling language for model-based safety engineering," in *Proc. 2nd Int. Workshop on Model Based Architecting and Construction of Embedded Systems*, Denver, CO, Oct. 6, 2009, pp. 11–26.
[44] S. Bernardi and J. Campos, "On performance bounds for interval Time Petri Nets," in *Proc. 1st Int. Conf. Quantitative Evaluation of Systems (QEST'04)*, Enschede, The Netherlands, Sep. 2004, pp. 50–59.
[45] R. Kearfott, "Interval computations: Introduction, uses, and resources," *Euromath Bulletin*, vol. 2, no. 1, pp. 95–112, 1996.
[46] S. Donatelli and G. Franceschinis, "The PSR methodology: Integrating hardware and software models," in *Proc. 17th Int. Conf. Application and Theory of Petri Nets (ICATPN96)*, Osaka, Japan, Jun. 1996, vol. 1091, LNCS.
[47] G. Upton and I. Cook, *The Oxford Dictionary of Statistics*. Oxford, U.K.: Oxford Univ. Press, 2002.
[48] *ARP4761: Aerospace Recommended Practise: Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airbone Systems and Equipment*, 12 ed. Warrendale, PA: Society of Automotive Engineers, 1996.
[49] Y. Papadopolous and J. McDermid, "Hierarchically performed hazard origin and propagation studies," in *Proc. SAFECOMP'99*, 1999, vol. 1698, LNCS, pp. 139–152.
[50] G. Mauri, "Integrating safety analysis techniques, supporting identification of common cause failures," Ph.D. dissertation, Dept. Comput. Sci., Univ. York, York, 2000.
[51] E. Pacini, S. Bernardi, and M. Gribaudo, "ITPN-PerfBound: A performance bound tool for Interval Time Petri Nets," in *Proc. 15th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2009)*, S. Kowalewski and A. Philippou, Eds., York, U.K., Mar. 2009, pp. 50–53.

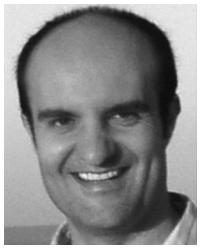
Simona Bernardi received the M.S. degree in mathematics and the Ph.D. degree in computer science from the University of Torino, Torino, Italy, in 1997 and 2003, respectively.

She is Professor at the Centro Universitario de la Defensa, General Militar Academy of Zaragoza, Zaragoza, Spain. From 2005 to 2010, she held a researcher position at the University of Torino. She has been a Visiting Researcher at the Department of Computer Science and System Engineering, University of Zaragoza, and at the Department of System and Computer Engineering, Carleton University, Canada. She has been serving as a referee for international journals and as a program committee member for several international conferences and workshops. Her research interests include software performance, dependability and security engineering, UML and object-oriented software development methodologies, formal methods for the modeling and analysis of software systems.

Javier Campos was born in Jaca, Spain, in 1963. He received the M.Sc. degree in applied mathematics and the Ph.D. degree in systems engineering and computer science (with Extraordinary Doctorate Award) from the University of Zaragoza, Zaragoza, Spain, in 1986 and 1990, respectively.

In 1986, he joined the Department of Computer Science and Systems Engineering, University of Zaragoza, where he was the Director from 2001 to 2003. In 2005, he was named Full Professor of Languages and Computing Systems at the Department of Computer Science and Systems Engineering, after winning one of the first two national competitive habilitation positions announced. His research interests include modeling and performance evaluation of distributed and concurrent systems, Petri nets, software performance engineering, and discrete-event systems in automation. He has supervised the completion of 100 M.S. thesis students and two Ph.D. students. Since 1989, he coauthored about 80 papers published in refereed journals and conferences. He has been plenary session invited speaker and tutorial invited speaker in several important meetings and has served o the Program Committee, sometimes as a Chair, of several international conferences. He is a Founding Member of the Aragón Institute for Engineering Research and a member of the Aragonese Informatics Engineering Association.

Dr. Campos has been a member of the IEEE IES Technical Committee on Factory Automation, Co-Chair of the IEEE IES Technical Sub-Committee on Industrial Automated Systems and Controls, Associate Editor of the IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, and Guest Editor of the Special Section on Formal Methods in Manufacturing in the same Journal.



José Merseguer received the B.S. and M.S. degrees in computer science and software engineering from the Technical University of Valencia, Valencia, Spain, and the Ph.D. degree in computer science from the University of Zaragoza, Zaragoza, Spain.

He is currently the Director of the Master in Computer Science and Systems Engineering at the University of Zaragoza, Zaragoza, Spain. He is also an Assistant Professor in the Department of Computer Science and Systems Engineering, University of Zaragoza. He teaches software engineering courses at graduate and undergraduate levels. He has developed postdoctoral research with Prof. M. Woodside at Carleton University, Ottawa, Canada, and with Prof. R. Lutz at Iowa State University. He has also been a Visiting Researcher at the Universities of Torino and Cagliari, Italy. He cooperates with different software companies, among them Akhela S.R.L. through a European research project. His main research interests include performance and dependability analysis of software systems, UML semantics, and service-oriented software engineering.

Dr. Merseguer is a member of the Aragón Institute for Engineering Research (I3A). He has been serving as a referee for international journals and as a program committee member for several international conferences and workshops. Among them the International Conference on Performance Engineering (ICPE). He is now serving as Publication Chair for ICPE'11.