

Available online at www.sciencedirect.com

Expert Systems with Applications 00 (2021) 1–1

Expert Systems with
Applicationswww.elsevier.com/locate/procedia

Reliability-driven Automotive Software Deployment based on a Parametrizable Probabilistic Model Checking

Abdelhakim Baouya^{a,*}, Otmane Ait Mohamed^b, Samir Ouchani^c, Djamal Bennouar^d

^aUniversity of Grenoble Alpes, Verimag Lab, Grenoble, France

^bConcordia University, Montréal, Canada, ECE Department

^cEcole d'Ingénieur CESI, Aix-en-Provence, France

^dHead of LIMPAF Lab, University of Bouira, Algeria

*Corresponding author at : Verimag Lab, University of Grenoble Alpes, Grenoble, France

Email addresses: abdelhakim.baouya@univ-grenoble-alpes.fr (Abdelhakim Baouya),
otmane.aitmohamed@concordia.ca (Otmane Ait Mohamed), souchani@cesi.fr (Samir Ouchani),
dbennouar@univ-bouira.dz (Djamal Bennouar)



2

Abstract

Embedded systems span a wide range from a small platform of sensors and actuators to distributed systems combining several interacting nodes. Designing such systems includes hardware parts and software parts. The software part acquires in importance since it handles the resources and services to interact with the hardware part. The paper introduces a novel deployment-decision making based on PRISM probabilistic model checker that takes software components and the physical platform to produce a set of deployment candidates. Starting from System Modeling Language (SysML), the process includes mechanisms to extract hardware and software features and carry out a set of deployment candidates. Each candidate should satisfy the reliability property written in Probabilistic Computation Tree Logic. Formally, we capture the underlying semantics of software blocks behaviour expressed as an activity diagram and their generated PRISM code to prove the approach soundness. Illustration relies on the automotive control system to show the applicability of the proposed approach.

Keywords: SysML internal block diagrams, Activity diagrams, Reliability, Model checking, Deployment

1. Introduction

Many cities are more and more overcrowded and lead to growing accidents and unpredicted emergencies. In response to that, the engineers have to open a way to improve safety and efficiency. Several innovative and cost-effective solutions are emerging to simplify our daily-life so-called Automotive Control Systems (ACS) (Mattes, 2014). The evolution of ACS is enabled by recent advances in computing and sensing technologies, as well as advances in estimation and control theory. However, restrictions on such systems in terms of reliability, safety, security are becoming more stringent. On one side, these systems are increasingly complex (i.e., encompassing several artefacts), and the hardware platform resources are often limited (e.g., memory, many hardware interfaces, communication bandwidth,..., etc.), on the other side, a quality metric such as reliability relevant to achieve the deployment relies on a specific set of features (Rashid et al., 2015). So, looking for a compromise between physical resources and reliability is obvious to manage the deployment plan.

Software deployment is known as NP-hard (Garey and Johnson, 1979); it consists of the deployment of software components on different physical locations. The term *component* refers to a software artefact where a composed one using interfaces is a system (Carlson et al., 2006; Herrera et al., 2014). (Arcangeli et al., 2015) and (Saxena and Karsai, 2011) give state of the art related to the deployment concepts and existed strategies. Reliability is one of the quality attributes for the achievement of the deployment. This issue is addressed by (Baouya et al., 2016) to increase reliability and to plan maintenance strategies of multi-processing systems. In our approach, we want to optimize the deployment regarding the reliability computed from a set of components features. Numerous publications such as (Zhang et al., 2013), (Peng et al., 2014), (Lu et al., 2015) and (Cherfi et al., 2014) discussed about reliability and state that the methods rely on the application of Markov models. One of the interesting approaches is probabilistic model checking

(Kwiatkowska et al., 2011), a quantitative analysis technique based on Markov models. It has been proven its usefulness in case of analyzing a wide range of reliability and availability properties (Zhang et al., 2013; Hoque et al., 2014; Baouya et al., 2019, 2020).

In our study, we mainly deal with this kind of relationship between the software components (i.e., blocks) and executing platform represented by processors and buses, and we study the architectural deployment alternatives. For this purpose, we specify our electronic system using SysML diagrams (OMG, 2012), and the satisfaction of our properties depends on the behaviour of its core software.

The basic idea is first to build a parameterizable Markov model (Filieri et al., 2016) that captures the software behaviour expressed in SysML activity diagrams, and then to check the property on the model using temporal logic. The input parameters represent the reliability computed from the components features of our automotive electronic system and the results of the software-hardware allocations. In order to allow an automatic verification using PRISM (Kwiatkowska et al., 2011), the underlying *Discrete-time Markov Chains* (DTMC) semantic model of system behaviour is constructed using the semantic rules. DTMC is used under the assumption that the system's behaviour meets with some tolerable approximation of the Markov property, i.e., the transition probability depends only on the current state, not on the history that leads to that state (Kwiatkowska et al., 2012). Figure 1 depicts all the steps for the deployment-space exploration (DSE). The specification consists of the SysML Internal Block Diagram (IBD) of both software and hardware components with its behaviour expressed in activity diagrams. The instances of blocks called *parts* are enriched with MARTE features (Modeling and Analysis of Real-Time and Embedded systems) (Mallet and de Simone, 2008) to express software and hardware characteristics such as workloads. To illustrate the use of transformations in this process, we study the deployment of subparts of the automotive control system.

The remainder of this paper is organized as follows: Section 2 discusses briefly Automotive Control Systems. Section 3 shows how the deployment is resolved according to the designer parameters. Section 4 describes the SysML block diagrams and their associated behaviour and then formalized. The PRISM model checker and its semantics are presented in section 5. Section 6 provides a mapping algorithm of SysML Activity Diagrams into PRISM code with its soundness proof. Section 7 illustrates the application of our framework in a case study. Section 8 reviews the related work. In section 9, our main findings and threats to validity are discussed. Section 10 draws conclusions and lays out future work.

2. Automotive control systems

The Automotive Control Systems are the *active safety functions* (Mattes, 2014) that encompass all features intended to prevent accidents. The main active functions that are studied in our paper are Anti-lock Brake (ABS) and Adaptive-Cruise Control (ACC) Subsystems.

2.1. Adaptive cruise control

ACC (Adaptive Cruise Control) (Mattes, 2014) simplifies the task of driving a car because it relieves the driver of the mentally demanding task of keeping a check on the car's speed. The primary function of ACC is keeping the speed constant at the setting selected by the driver (i.e. Absence of vehicle in front). In Figure 2, if the vehicle in front is travelling at a constant speed, a car fitted with ACC will follow it at the same speed and a virtually constant distance. That is because the distance between the two vehicles is at least within a broad speed range. Here the speed change is automatic without the need for driver intervention.

2.2. Anti lock brake system

Wheels of a vehicle may lock up under braking due to wet or slippery road surfaces then the vehicle become uncontrollable and leave the road. The anti-lock braking system (ABS) detects if one or more wheels are about to lock up under braking and if so makes sure that the brake pressure remains constant or is reduced. As a consequence, the vehicle can be braked or stopped quickly and safely. The core unit in the system processes the information received from the brake paddle according to defined mathematical procedures. The results of those calculations form the basis for the control signals sent to the emergency stop detection (see Section 7).

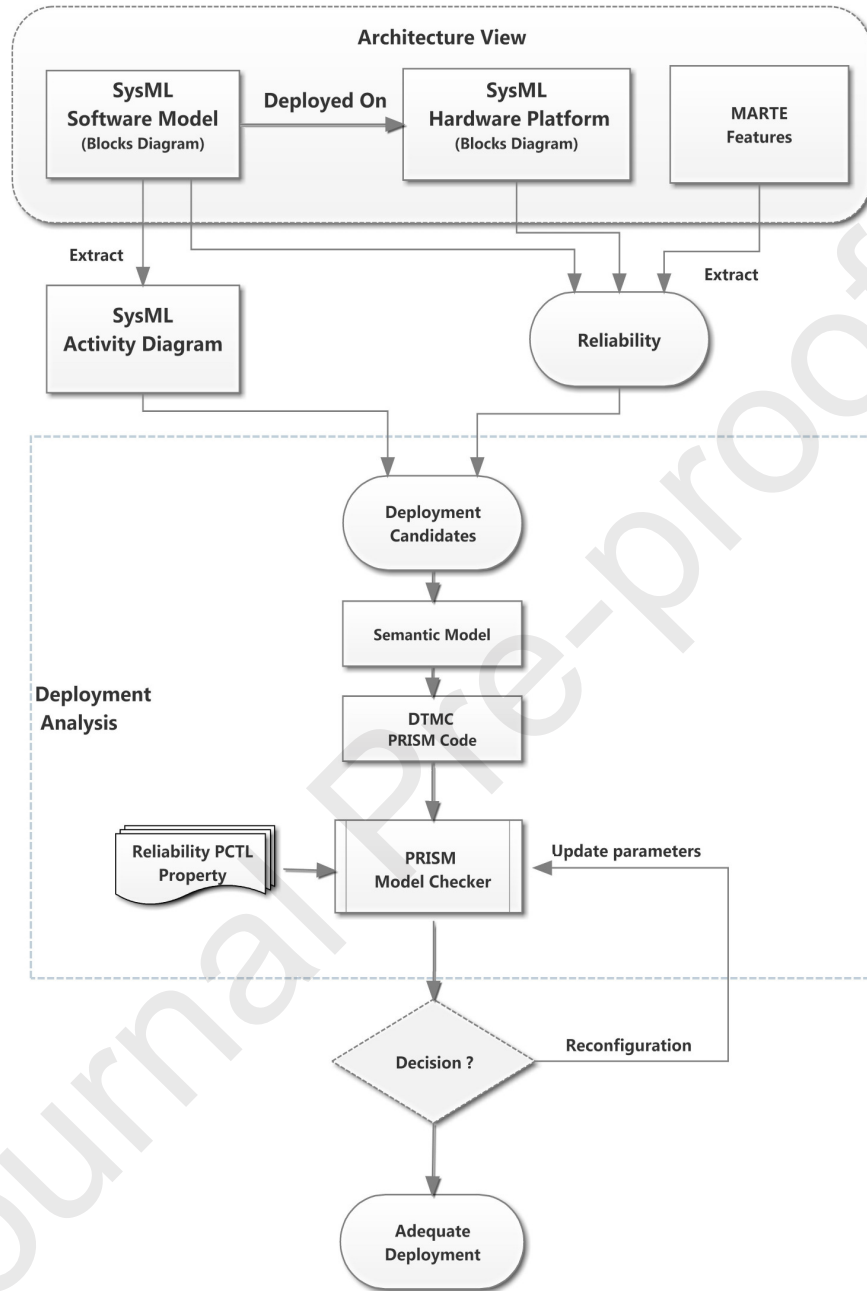


Figure 1: Reliability-driven deployment approach

3. Solving the deployment problem

According to (Arcangeli et al., 2015), the deployment is a post-production process that consists of software components (Carlson et al., 2006) for use and keeping them operational and up-to-date. Thus, the deployment plan must satisfy a set of constraints before production. Such constraints are imposed over the

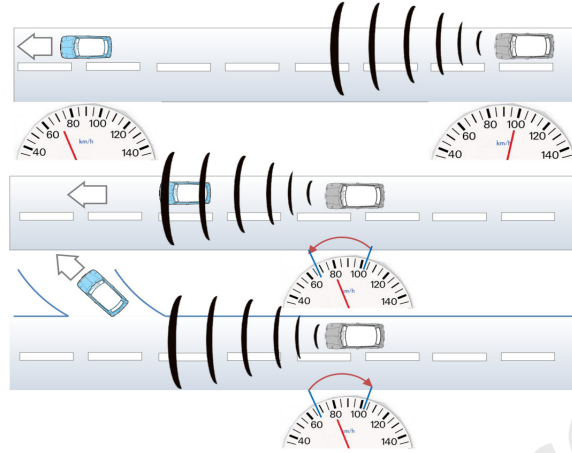


Figure 2: Adaptive cruise control system

dependencies between components themselves (i.e., Scheduling and communication) and between software components and their environment (e.g., Execution platform, smartphone). Papers referring to software deployment are those of (Flissi et al., 2008) and (Juve and Deelman, 2011). (Flissi et al., 2008) propose *DeployWare* for the deployment of software systems on nodes of grids. The deployment scheme is described using an architecture description language (ADL), which captures the system configuration. Validation consists of fulfilling all dependencies between software components, avoiding errors, and resource conflicts. (Juve and Deelman, 2011) target an automatic deployment of distributed applications on clouds using *Wrangler*. It allows users to send a simple XML description of the desired deployment consisting of several nodes and services to the cloud coordinator. It manages the provisioning of virtual machines, the installation and the configuration of software and services. The coordinator ensures that the request is valid, and all dependencies can be resolved.

In embedded systems, the deployment process consists of automatically explores the states' space of possible allocations of software blocks to hardware blocks according to the constraints addressed in this section and returns the set of near-optimal candidates. In our paper, software deployment is driven by the reliability of the system services that must be maximized. Service refers to the flow of actions and data at the software level. We assume that software failures due to programming imperfections are unlikely influencing the deployment process, then we may abstract away from the hardware-independent software reliability. Also, the software and hardware architecture is constant during the deployment.

Our approach assumes that processors are failing silent (Meedeniya et al., 2011), (Heiner and Thurner, 1998); It means that the processor detects all errors and switch immediately to a passive state, which leads to an exception at the software level. When failures occur during the execution of software blocks, it impacts the reliability of our system. With a fixed and deterministic scheduling strategy, any failure that happens on the processor leads to service execution failure.

To evaluate a single deployment, we use some specification metrics for the software and the platform architecture to capture the system reliability. The constraints established in this section correspond to the attributes of components in the SysML specification. For instance, the processor capacity corresponds to the `<< HwProcessor >>` featured with MIPS as MARTE annotation (see Section 4).

3.1. Hardware Architecture

The set of execution components represented by processor are denoted by $U = \{ u_1, u_2, \dots, u_m \}$, where $m \in \mathbb{N}$ and the parameters of hardware architecture are given as follows:

- Processor speed, $ps : U \rightarrow \mathbb{N}$; the instruction-processing capacity of the hardware component in MIPS(million instructions per second).

- Processor failure rate, $\lambda : U \rightarrow \mathbb{N}$; the rate parameters of the Poisson distribution that characterizes the probability of a single processor failure.

3.2. Software Architecture

The set of software components that must be allocated to processor are denoted by $C = \{ c_1, c_2, \dots, c_n \}$, where $n \in \mathbb{N}$ and the parameters of software components are given as follows:

- Component workload, $wl : C \rightarrow \mathbb{N}$; computational requirement for component expressed in MI (millions instructions)

Let $D = \{ d \mid d : C \rightarrow U \}$ denotes the set of all functions assigning software components to hardware processors, and we write a single deployment alternative $d_j = \{ (c_1, u_{j1}), (c_2, u_{j2}), \dots, (c_n, u_{jm}) \} \subseteq D$ as $d_j = \{ u_{j1}, u_{j2}, \dots, u_{jm} \}$.

The quality metric of the approach depends on the reliability of physical hardware (Processors and buses). The reliability of the physical elements in our paper is computed according to the failures of execution elements (Processors). The reliability of single element i (Meedeniya et al., 2011) can be modeled with exponential distribution as:

$$R_i = e^{-\lambda \cdot T} \quad (1)$$

where λ is the failure rate of the element and T is the time elapsed in it.

In this model, failure rates of execution elements are obtained from the hardware architecture parameters, and the time taken for the execution relies on the component workload and processing speed (Meedeniya et al., 2011). In time triggered-architecture (Heiner and Thurner, 1998), processing depends on the fixed scheduling algorithm. Thus, the requests are queued until their time slot arrives. Based on the software workload and processing speed of the processors, the scheduling length can be calculated as follows:

$$sl(u_i) = \sum_{c \in d^{-1}(u_i)} \frac{wl(c)}{ps(u_i)} \quad (2)$$

The reliability of individual software block are computed as :

$$R_i = e^{-\lambda(d(c_i))sl(d(c_i))/2} \quad (3)$$

The result R_i refers to the reliability of each allocation of software component that allows populating our activity diagram with probabilities or could be parameters for the PRISM model checker. We note that the probabilities on the activity diagram are unset (i.e., variables). When the reliability of individual software blocks is known, the full reliability is computed using the PRISM model checker applied over the parameterizable activity diagram, and we refer to the full reliability as R_{d_j} .

Having the reliability measure R_{d_j} for each candidate under deployment $d \in D$, the reliability R of the best candidate is calculates as :

$$R = \text{Max}\{R_{d_0}, \dots, R_{d_m}\}, \quad m \in \mathbb{N} \quad (4)$$

4. Modelling automotive systems using SysML

The system specification described in this paper is characterized by following a component-oriented approach (Szyperski, 2002). In a component-based approach, the system is built by the composition of multiples blocks (i.e., component) interacting with each other through a well-defined interface. Also, the specification of behaviour is based on activity diagrams.

4.1. SysML internal blocks diagram

The **block** (Friedenthal et al., 2008) is the fundamental modular unit for describing system structure in SysML. The internal block diagram (IBD) conveys how the parts of a primitive block must be assembled to create a valid instance of the main block (Delligatti, 2013). The graphical notation of SysML helps designers for the specification of embedded systems using IBD in easy way (Nejati et al., 2012). However, the description needs a piece of more information about the software and hardware blocks to apply analysis such as the processor speed. To provide capabilities that are missing in SysML, the MARTE profile is used to specify the required platform resources and their quality of service characteristics.

In the case of our approach, the processor is the main component in the hardware board stereotyped using MARTE “HW_Processor” from sub-profile Hardware Resource Modeling (HRM) (See Figure 3) and some attributes can be added like *mips* that characterize the instruction-processing capacity. To annotate each processor and bus components with the probability of success and failure of data transmission, we use “GaStep” annotation from the Generic Quantitative Analysis Modeling package (GQAM) with attribute *prob*. Figure 3 represents a multi-processing platform used in our case study where blocks are stereotyped with MARTE annotations. In our paper, *prob* represents the success of processing and the correct service execution. In Table 1, we show the features studied in the previous section with its corresponding MARTE annotations. In addition, when the blocks are defined, they are endowed with behaviour. There are three main behavioural formalisms in SysML: activities, state machines, and interaction diagrams.

Deployment features	MARTE annotations
Processor speed	<i>mips</i>
Processor failure	<i>Prob</i>
Component workload	<i>hostDemandOps</i>

Table 1: Software/Hardware deployment features with their corresponding MARTE annotations.

4.2. SysML activity diagram

Activity diagram is one of the behavioural diagrams in SysML (OMG, 2012) encompassing a set of artefacts. A subset of them are considered in the paper as portrayed in Fig 4 are grouped into four categories: nodes, edges, controls and partitions. Activity nodes consists of activity invocation, objects, and actions nodes. Edges could be control object or object edges. Controls include initial node, flow final, activity final, fork, merge, and join node, whereas, activity invocation includes call behaviour, send/receive signals (objects). Object edges flow the data from actions output pin to input pins. Control edges trigger the execution of activity nodes where control flows are routed by control nodes. A set of activity diagram artefacts can be grouped into activity partitions (also known as a *swimlane*). A typical case is when an activity partition represents a block or a part and indicates that any behaviours invoked in that partition are the responsibility of the block or the part.

In the paper, we link each subset of actions to each block of the IBD. Figure 5 presents an example of the electric motor inverter (Cherfi et al., 2014) where different activity artefacts are allocated to blocks. The flows crossing the partitions are a kind of data and events transmission, and they are annotated with probabilities. An exception could be generated and handled by the *interruptible region* in the first block.

4.3. SysML activity diagram formalization

Our methodology enables the search of the best software-hardware blocks deployment from a parameterizable activity diagram which means; for different deployment candidates, we have different probabilities. The only change occurs on the activity diagram probability values. To have the best deployment, each candidate with reliability values is checked. Based on (Ouchani et al., 2014b; Baouya et al., 2015), we formalize the deployment by developing its calculus : *Deployment Activity Calculus* (DAC). In Table 2, we formally rewrite each SysML activity diagram artifact and each formal notation is identified by a label *l* that belongs to the set of labels denoted by \mathcal{L} . We use $Ex(p, \mathcal{N}_1, \mathcal{N}_2)$ to express an exception that could happen with

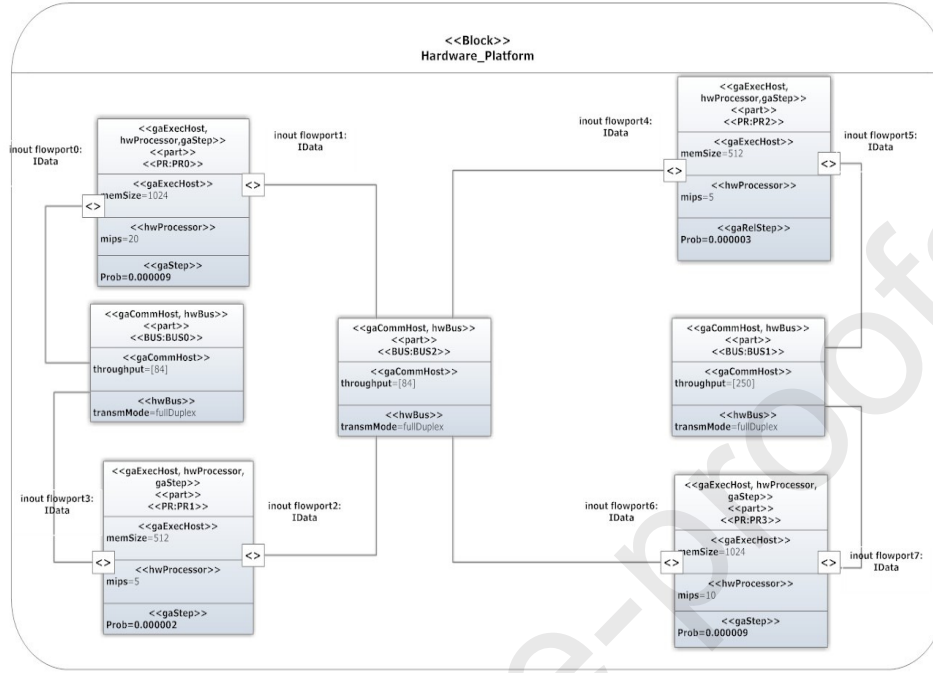


Figure 3: Example of hardware platform.

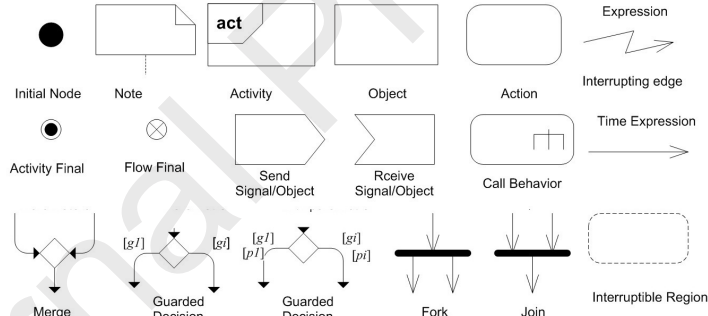


Figure 4: A sub set of SysML activity diagram artefacts

probability $1 - p$. The symbol \mathcal{N}_1 refers to the next SysML node except \mathcal{N}_2 which refers to the SysML receiving signal node. Marked and Unmarked terms are two syntactic concepts in this calculus. A marked terms are active component with tokens. The unmarked terms correspond to the static structure of the service workflow. To describe better our approach, in Table 2 we formally rewrite each SysML activity diagram artifact and each formal notation is identified by a label l that belongs to the set of labels denoted by \mathcal{L} .

For the workflow observations, we use structural operational semantics (Milner, 1999) in Table 3 to formally describe how the computation steps of DAC atomic terms take place. The semantics of parameterizable activity diagram can be described in terms of DTMC as stipulated by Definition 1.

Definition 1 (DAC-DTMC). A Discrete-time Markov Chain of DAC term \mathcal{A} is the tuple $M_{\mathcal{A}} = \langle \bar{s}, S, P, L \rangle$, where:

- \bar{s} is an initial state, such that $L(\bar{s}) = \{l : \bar{i} \mapsto \mathcal{N}\}$,

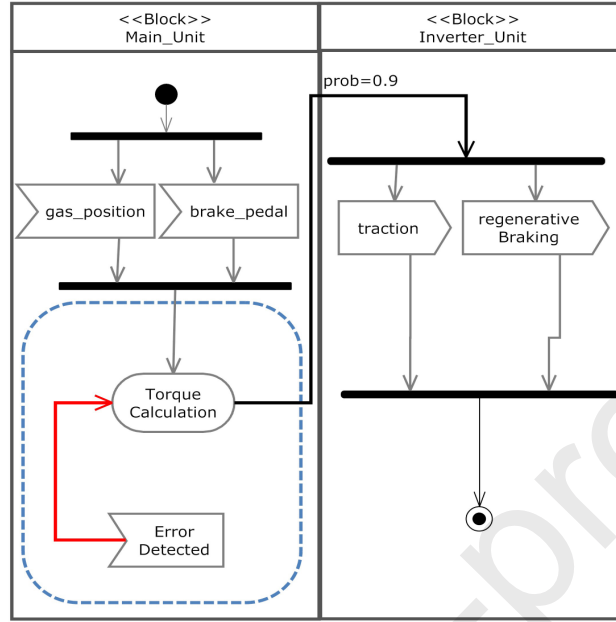


Figure 5: An example of activity allocated to the blocks

- S is a finite set of states reachable from \bar{s} , such that, $S = \{s_i : 0 \leq i \leq n | L(s_i) = \{\bar{N}\}\}$,
- $P: S \times S' \rightarrow [0, 1]$ is a probability function assigning for each transition from s to s' a positive real value p where: For each $S' \subseteq S$ such that $S' = \{s_i : 0 \leq i \leq n : s \rightarrow_{p_i} s_i\}$, each transition $s \rightarrow_{p_i} s_i$ satisfies one DAC semantic rule and $\sum_{i=0}^n p_i = 1$.
- $L: S \rightarrow 2^{[[\mathcal{L}]]}$ is a labeling function where: $[[\mathcal{L}]]: \mathcal{L} \rightarrow \{true, false\}$.

5. Probabilistic model checking

Probabilistic model checking is based on the construction and the analysis of a probabilistic model of a system, typically a Markov chain. In this paper, we focus on Discrete-Time Markov chains (DTMCs) with proved usefulness in reliability and availability analysis (Franco et al., 2016) (Kwiatkowska et al., 2009b). A DTMC involves a set of states S showing the values of the system's variables and a transition probability matrix $\mathbf{P}: S \times S \rightarrow [0, 1]$ describing the system's changes through states. Definition 2 formally describes a DTMC.

Definition 2 (Discrete-Time Markov chains (DTMC)). A DTMC is a tuple $M = \langle \bar{s}, S, P, L \rangle$, where:

- \bar{s} is an initial state, such that $\bar{s} \in S$,
- S is a finite set of states,
- $P: S \times S \rightarrow [0, 1]$ is a function assigning to each transition between two states $s, s' \in S$ a positive real value $P(s, s') \in [0, 1]$ where $\sum_{s' \in S} P(s, s') = 1$,
- $L: S \rightarrow 2^{AP}$ is a labeling function that assigns each state $s \in S$ to a set of atomic propositions taken from the set of atomic propositions (AP).




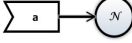
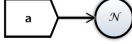
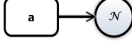
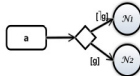



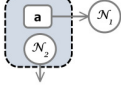
Artifacts	DAC terms	Description
	$l : i \rightarrow N$	When a diagram is invoked <i>Initial node</i> is activated .
	$l : \odot$	<i>Activity final node</i> ends the diagram' execution.
	$l : \otimes$	<i>Flow final node</i> kills its related path' execution.
	$l : a?v \rightarrow N$	Receive a signal/object using <i>Receive node</i> .
	$l : a!v \rightarrow N$	Send a signal/object using <i>Send node</i> .
	$l : a \rightarrow N$	Atomic action by the way of <i>Action node</i> .
	$l : D(g, N_1, N_2)$	<i>Decision node</i> with guarded edges $\{g, \neg g\}$.
	$l : M(x) \rightarrow N$	<i>Merge node</i> is a routing node and $x = \{x_1, x_2\}$ is a set of input flows.
	$l : J(x) \rightarrow N$	<i>Join node</i> shows the synchronization where $x = \{x_1, x_2\}$ is a set of input pins.
	$l : F(N_1, N_2)$	<i>Fork node</i> triggers multiple parallel nodes.
	$l : Ex(p_1, N_1, N_2)$	<i>Interruptible region</i> models the interruption when errors occurs with $1 - p_1$

Table 2: DAC terms of SysML activity diagram artifacts.

Let's consider the case of a Vehicle Management Unit (VMU) (Cherfi et al., 2014). In an electric vehicle, a VMU is responsible for commanding the electric motor inverter, among other functions. With a given specific inputs (gas and brake pedal positions), the VMU sends a torque set point to the inverter that in turn commands the electric motor (traction and regenerative braking), as illustrated in Figure 5. In order to prevent unexpected errors, a watchdog (i.e. a software module) is added, which is in charge of bringing the system to a last safe state when errors are detected. The partial corresponding PRISM code is illustrated in Listing 1, the failure is a signal received at the exception node "Error_Detected". P and $1 - P$ corresponds to the correct and failure signal transmission, respectively.

INT-1	$l : i \mapsto \mathcal{N} \xrightarrow{l} l : \bar{i} \mapsto \mathcal{N}$	Execution of \mathcal{A} by putting a token on i .
INT-2	$l : \bar{i} \mapsto \mathcal{N} \xrightarrow{l} l : i \mapsto \mathcal{N}$	Token propagation from the marked term i to \mathcal{N} .
ACT-1	$\forall n > 0, m \geq 0 \ l : \overline{a^m} \mapsto \mathcal{N}^n \xrightarrow{l} l : \overline{a^{m+1}} \mapsto \mathcal{N}^{n-1}$	Token propagation from the global marked term to a .
ACT-2	$l : \overline{a^{m+1}} \mapsto \mathcal{N}^n \xrightarrow{l} l : \overline{a^m} \mapsto \mathcal{N}^n$	Token propagation from the marked term a to \mathcal{N} .
FRK-1	$\forall n > 0 \ l : \overline{F(\mathcal{N}_1, \mathcal{N}_2)}^n \xrightarrow{l} l : \overline{F(\mathcal{N}_1, \mathcal{N}_2)}^{n-1}$	FRK-1 propagates token to the outgoing sub-terms.
DEC-1	$\forall n > 0 \ l : \overline{D(g, \mathcal{N}_1, \mathcal{N}_2)}^n \xrightarrow{g} l : \overline{D(g, \mathcal{N}_1, \mathcal{N}_2)}^{n-1}$	DEC-1 describes a token flows through the edge satisfying its guard.
EXP-1	$l : \overline{Ex(\mathcal{N}_1, \mathcal{N}_2)}^n \xrightarrow{l} l : \overline{Ex(\mathcal{N}_1, \mathcal{N}_2)}^{n-1}$	Node \mathcal{N}_2 is triggered after an exception.
EXP-2	$l : \overline{Ex(p, \mathcal{N}_1, \mathcal{N}_2)}^n \xrightarrow{l_{1-p}} l : \overline{Ex(p, \mathcal{N}_1, \mathcal{N}_2)}^{n-1}$	Node \mathcal{N}_2 is triggered after an exception with probability $1 - p$.
MRG-1	$l : \overline{\mathcal{N}} \mapsto l' : \overline{M(x, y)}^n \xrightarrow{l} l : \overline{\mathcal{N}} \mapsto l' : \overline{M(x, y)}^n$	MRG-1 puts a token coming from the sub-term \mathcal{N} to one of its inputs flow.
MRG-2	$l : \overline{M(\bar{x}, \bar{y})} \mapsto \mathcal{N}^n \xrightarrow{l} l : \overline{M(x, \bar{y})} \mapsto \mathcal{N}^n$	MRG-2 propagates a token from the input flow x to the outgoing node \mathcal{N} .
JOIN-1	$l : \overline{\mathcal{N}} \mapsto l' : \overline{J(x, y)}^n \xrightarrow{l} l : \overline{\mathcal{N}} \mapsto l' : \overline{J(\bar{x}, \bar{y})}^n$	JOIN-1 activates one of the pins of the join node.
JOIN-2	$l : \overline{J(\bar{x}, \bar{y})} \mapsto \mathcal{N}^n \xrightarrow{l} l : \overline{J(x, y)} \mapsto \mathcal{N}^n$	JOIN-2 move a token in join to the node \mathcal{N} .
SND	$l : \overline{a!v}^n \mapsto \mathcal{N} \xrightarrow{l} l : \overline{a!v}^{n-1} \mapsto \mathcal{N}$	After sending an object v the node \mathcal{N} is activated.
AFIN	$\mathcal{A}[l : \odot] \xrightarrow{l} \mathcal{A} $	This axiom states that if the sub-term $l : \odot$ is reached then all tokens are destroyed.

Table 3: Operational semantic rules of the DAC calculus.

Listing 1: A partial VMU PRISM code

```

1  dtmc
2  // A parametrizable variable P
3  const double P;
4  module satellite
5    initial : bool init true; // initial node
6    .....
7    Torque_Calculation : bool init false;
8    [initial] initial → (Main_Unit_Fork'=true) & (initial'=false);
9    .....
10
11   [Torque_Calculation] Torque_Calculation →
12   // A successful transition with probability P
13   P : (Inverter_Unit_Fork'=true) & (Torque_Calculation'=false)
14   // A failure transition with probability 1-P
15   +1-P : (Error_Detected'=true) & (Torque_Calculation'=false);
16
17   [Error_Detected] Error_Detected →
18   (Torque_Calculation'=true) & (Error_Detected'=false);
19   .....
20 endmodule

```

5.1. PRISM by example

To illustrate the applicability of the PRISM model checker, this subsection portrays an example of reliability model description (Kwiatkowska et al., 2007b). The system comprises an input processor that reads data from three sensors. It is then polled by the main processor, which is, in turn, passes the instructions to an output processor. The output processor takes the instructions it receives and uses them to control two actuators. Also, a bus connects the three processors together. Any of the three sensors can fail with probability P_{sf} ; if more than one processor becomes unavailable, the processor reports this fact to the main processor, and the system is shutdown.

The PRISM model of the system comprises six modules, one for the sensors, one for the actuators, one for each actuator, one for each processor, and one for the connecting bus. Listing 2 shows the section which models the sensors' failure behaviour with an integer value of s representing the number of sensors currently working. The module behaviour is described by three guarded commands representing the failure of a single sensor. The commands with guards " $s=3$ ", " $s=2$ " and " $s=1$ " states this can occur for the third, the second, and the first sensor, respectively. The update decrements the counter of the functioning sensors in each command. For instance, the commands in lines 11-12-13, the sensors stay available with probability P_{sf} and fails with probability $1-P_{sf}$.

Further, Listing 2 shows a second module in lines 16-20, which is the PRISM language description of the input processor. The module has a single variable i evaluable over the set $\{0, 1, 2\}$, which indicates the possible states of the processor: is working, is recovering from a transient fault, or has failed. The two commands correspond, respectively, to the processor is working and may fail with probability ' p ' or suffering a transient fault. The second command indicates that the processors may fail or reboot. The guards of these commands can refer to variables from other modules by the use of $s \geq 2$. It is because the input processor ceases to function once it has detected than less than two processors are operational.

Listing 2: A partial PRISM code for the sensors and processors.

```

1  dtmc
2  //  $P_{sf}$  for sensor failure probability
3  const double  $P_{sf}$ ;
4  //  $P_{pf}$  for processor failure probability
5  const double  $P_{pf}$ ;
6  //  $P_{ptf}$  for processor transient failure probability
7  const double  $P_{ptf}$ ;
8
9  module sensor
10  s : [0..3] init 3; // number of sensors working
11  [] s=3 →  $P_{sf}:(s'=3) + (1-P_{sf}):(s'=3)$ ;
12  [] s=2 →  $P_{sf}:(s'=2) + (1-P_{sf}):(s'=2)$ ;
13  [] s=1 →  $P_{sf}:(s'=1) + (1-P_{sf}):(s'=1)$ ;
14  endmodule
15
16  module inputprocessor
17  i : [0..2] init 2; // 2= working, 1=transient fault, 0=failed
18  [] i =2 & s≥2 →  $P_f:(i'=0) + P_{tf}:(i'=1) + (1-P_{tf}-P_f):(i'=2)$ ;
19  [] i =1 & s≥2 →  $P_f:(i'=0) + (1-P_f):(i'=2)$ ;
20  endmodule

```

The model checking approach for reliability analysis requires both a description of the system in the probabilistic model and properties for performance evaluation using Probabilistic Computation Tree Logic (PCTL) (Kwiatkowska et al., 2002). For the PRISM code presented in Listing 1, the percentage of the reliability is obtained by checking the model against the property: "the probability that the system runs correctly within T time steps which returns 98%" and expressed using PCTL as $P = ?[true \cup^{\leq T} S_{success}]$, $T =$

100 ; T is a non-negative value representing the upper time-bound. The BNF grammar expressing the structure of our temporal logic is presented in the following subsection.

5.2. Property Specification

The syntax of our logic is given by the following grammar:

$$\varphi ::= \text{true} \mid ap \mid \varphi \wedge \varphi \mid \neg \varphi \mid P_{\bowtie p}[\psi],$$

$$\psi ::= \varphi \cup \varphi \mid \varphi \cup^{\leq k} \varphi,$$

Where “ap” is an atomic proposition, P is a probabilistic operator. Operator $P_{\bowtie p}[\psi]$ means that the probability of a path formula ψ being true always satisfies the bound $\bowtie p$, $p \in [0, 1]$. “ \bowtie ” $\in \{<, \leq, >, \geq\}$. “ \wedge ” represents the conjunction operator and “ \neg ” is the negation operator. “ $\cup^{\leq k}$ ” and “ \cup ” are the bounded until and the until temporal logic operators, respectively.

We use $s \models \varphi$ to denote that a state s satisfies the state formula φ , while $\pi \models \psi$ (sequence of states) denotes that π satisfies the path formula ψ .

- $s \models \text{true}$ is always satisfied,
- $s \models ap \iff ap \in L(s)$ and L is a labeling function,
- $s \models \varphi_1 \wedge \varphi_2 \iff s \models \varphi_1 \wedge s \models \varphi_2$,
- $s \models \neg \varphi \iff s \not\models \varphi$,
- $s \models P_{\bowtie p}[\psi] \iff P\{\pi \mid \pi \models \psi\}$ such that the probability of the path $\pi = s_0 \dots s_n$ is given by $P(\pi) = \prod_{i=0}^{n-1} p(s_i, s_{i+1})$,
- $s \models P_{\bowtie p}[\varphi_1 \cup^k \varphi_2] \iff \exists i \leq k : \forall j < i, \pi(j) \models \varphi_1 \wedge \pi(i) \models \varphi_2$, and
- $s \models P_{\bowtie p}[\varphi_1 \cup \varphi_2] \iff \exists k \geq 0 : \pi \models \varphi_1 \cup^k \varphi_2$.

5.3. The PRISM model checker

PRISM model checker (Kwiatkowska et al., 2011) supports modeling probabilistic systems and checking them through the probabilistic formal verification. A system can support deterministic, nondeterministic, and probabilistic decision in addition to timed and priced events such as: Discrete-time Markov chains (DTMC) (Kwiatkowska et al., 2012), Continuous-time Markov chains (CTMC) (Kwiatkowska et al., 2007a), Markov decision process (MDP) (Forejt et al., 2011). We rely on DTMCs since they have been shown as an appropriate semantic model for modeling reliability (Franco et al., 2016) in SysML Activity Diagrams.

A PRISM program (\mathcal{P}) is a composition of one or more modules through synchronization or interleaving where each module describes the behaviour of a part of an entity of the system. The state of each module is defined by a set of fixed ranged variables local V . The global state of the system is determined by evaluating the values of all module variables. The behaviour of a module is described using commands. A command describes the Dirac transition or a distributed probabilistic transitions that is expressed by $[a]\text{guard} \rightarrow p_1 : u_1 \dots + p_n : u_n$ where a labeling the command with respects to the action to be executed. **guard** is a propositional logic formula over the variables of all the modules in the model and it expresses a state of the module. The update u_i present the next values that should be assigned to its variables when executing the action a with a probability p . A given u takes the form $(V'_1 = val_1) \& \dots \& (V'_k = val_k)$ where $V_i \in [\min \dots \max]$ are local variables of the module such that $\min, \max \in \mathbb{N}$ and $\min < \max$, and val_j are values evaluated via expressions denoted by “eval” eval: $V \rightarrow \mathbb{N} \cup \{\text{True}, \text{False}\}$.

Definition 3 stipulates the formal definition of PRISM DTMC denoted by $M_{\mathcal{P}}$. The states of $M_{\mathcal{P}}$ take the form $\langle V_1, \dots, V_k \rangle$. The stepwise behaviour of $M_{\mathcal{P}}$ is described by the operational semantic rules provided in (Baouya et al., 2015).

Definition 3 (PRISM-DTMC). A Discrete-Time Markov Chain (DTMC) of PRISM program \mathcal{P} is the tuple $M_{\mathcal{P}} = \langle s_i, S, P, V \rangle$, where:

- s_i is an initial state,
- S is a finite set of states reachable from s_i ,
- $P: S \times S \rightarrow [0, 1]$ is a probabilistic function where $\sum_{s' \in S} P(s, s') = 1$ for all $s \in S$,
- $V: S \rightarrow 2^{AP}$ is a labeling function that assigns for each state a set of valuated propositions.

6. Reliability-driven deployment analysis

In this section, we highlight the translation rules relative to the mapping from SysML diagrams into PRISM input language. These rules provide a hand up to automatize the deployment checking and analysis. Also, the transformation is proved as sound.

6.1. The transformation rules

The transformation of Activity diagrams \mathcal{A} into a DTMC written in the PRISM input language is managed using a depth-first search algorithm (DFS) developed in (Ouchani et al., 2014b).

The function Γ presented in Listing 3 outputs PRISM commands for each DAC term. The action label of a command is the label of its related term “ n ”. The guard of this command defines how the term is activated. The flag related to DAC terms is initialized to false except the initial node as mentioned rule “INIT-1”. The commands updates deactivate the propositions of the term, activate those related to its successors. For a term $n \in \mathcal{A}$, we define a useful function: $L(n)$ that returns the label of a term n .

For example, the exception “ $l: Ex(p, N_1, N_2)$ ” in line 29 produces command in lines (31-32). The first part of the command in line 31 refers to the correct state transition. Similarly, the second part of the command in line 32 refers to an exception that may occur with probability $1 - p$. This command represents the DAC rule “EXP-1”. The DAC term “ $l: D(g, N_1, N_2)$ ” produces two commands in line 26. The first command is executed when the guard is verified else the second one is executed which follows the DAC rule “DEC-1”. The generated PRISM fragment of each diagram \mathcal{A} is bounded by two PRISM primitives: the module head “module \mathcal{A} ”, and the module termination “endmodule”.

Three examples are portrayed in the Table 4 encompassing the merging, the decision, and the interruptible region with their corresponding PRISM code:

The first example is related to the merging node linked to three action nodes. This structure generates three commands. Following the operational semantic rule **MRG-1** in Table 3, if action nodes “ A_1 ” or “ A_2 ” are activated it will trigger the merging node, then, two PRISM commands are generated by applying the rule in line 9 of the Listing 3 that deactivate the current action node (“ A_1 ” or “ A_2 ”) and activate the merging node “ M ”. The merging node, in turn, will be deactivated, and the next action node “ A_3 ” is activated following the operational semantic rule **ACT-2** in Table 3. Also, by applying the transformation rule in line 46 of the Listing 3 a PRISM command is generated where A_3 will be set to true.

The second example concerns the decision node, the action node “ A_1 ” will trigger the decision node “ D ” following the operational semantic rule **ACT-2** in Table 3. The corresponding PRISM command is generated following the rule in line 46 of Listing 3. Following the the operational semantic rule **DEC-1** in Table 3, the decision node “ D ” is deactivated, and the next nodes will be activated depending on guard values. By applying the rule in line 24 of Listing 3, the first generated PRISM command will activate the action node “ A_2 ” if the guard “ g_1 ” is true. The second command will activate the action node “ A_3 ” if the guard “ g_2 ” is true.

The last example concerns the interruptible region, the action node “ A_1 ” activate the action node “ A_2 ” which belongs to the interruptible region following the operational semantic rule **ACT-2** in Table 3. The corresponding generated PRISM command depends on the rule in line 46 of Listing 3. Then “ A_2 ” will activate the action node “ A_3 ” with probability p , meanwhile, “receive node” is triggered with probability

351 1 – p following the operational semantic rule **EXP-2** in Table 3. The corresponding generated PRISM
 352 commands depend on the rule in line 29 of Listing 3. The transition leaving the interruptible region depends
 353 on the operational semantic rule **ACT-2** in Table 3, and it is translated into a PRISM command following
 354 the rule in line 46 of Listing 3.

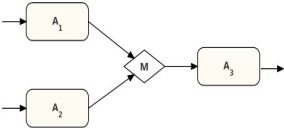
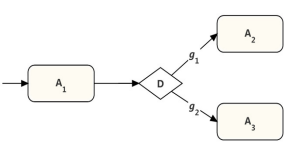
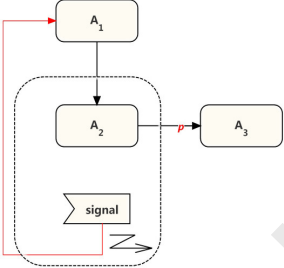
SysML Activity Diagram Structure	PRISM Code Fragment
	<pre> // Activation of merging node and deactivation of Action node (rule in line 9) [A1] A1 → (A1'=false) & (M'=true); [A2] A2 → (A2'=false) & (M'=true); // Activation of the next action node and deactivation of Merging node [M] M → (M'=false) & (A3'=true); </pre>
	<pre> // Activation of Decision node and deactivation of Action node (rule in line 46) [A1] A1 → (A1'=false) & (D'=true); // Activation of Action node and deactivation of Decision node (rule in line 24) [g1] D & g1=true → (D'=false) & (A2'=true); [g2] D & g2=true → (D'=false) & (A3'=true); </pre>
	<pre> // Activation of the next Action node and deactivation of current node (rule in line 46) [A1] A1 → (A1'=false) & (A2'=true); // Activation of Action node with probability p (rule in line 29) [A2] A2 → p:(A2'=false) & (A3'=false) + (1-p):(A2'=false) & (signal'=true); // By applying rule in line 46 [signal] signal → (signal'=false) & (A1'=true); </pre>

Table 4: Mapping examples of SysML activity diagram artifacts.

Listing 3: Generating PRISM commands function.

```

1   $\Gamma : \mathcal{A} \rightarrow \mathcal{P}$ 
2   $\Gamma(\mathcal{A}) = \forall n \in \mathcal{A}, L(n = (i)) = \text{true}, L(n \neq (i)) = \text{false}$ 
3
4   $l : i \rightarrow N \Rightarrow$ 
5  in
6     $\{[l]l \rightarrow (l' = \text{false}) \& (L(N)' = \text{true});\} \cup \Gamma(L(N))$ 
7  end
8
9   $l : M(x, y) \rightarrow N \Rightarrow$ 
10 in
11    $\{[l_x]l_x \rightarrow (l'_x = \text{false}) \& (L(N)' = \text{true});\} \cup \{[l_y]l_y \rightarrow (l'_y = \text{false}) \& (L(N)' = \text{true});\} \cup \Gamma(L(N))$ 
12 end
13
14  $l : J(x, y) \rightarrow N \Rightarrow$ 
15 in
16    $\{[l]l_x \wedge l_y \rightarrow (l'_y = \text{false}) \& (l'_x = \text{false}) \& (L(N)' = \text{true});\} \cup \Gamma(L(N))$ 
17 end
18
19  $l : F(N_1, N_2) \Rightarrow$ 
20 in
21    $\{[l]l \rightarrow (l' = \text{false}) \& (L(N_1)' = \text{true}) \& (L(N_2)' = \text{true});\} \cup \Gamma(L(N_1)) \cup \Gamma(L(N_2))$ 
22 end
23
24  $l : D(g, N_1, N_2) \Rightarrow$ 
25 in
26    $\{[l]l \wedge g \rightarrow (l' = \text{false}) \& (L(N_1)' = \text{true})\} \cup \{[l]l \wedge \neg g \rightarrow (l' = \text{false}) \& (L(N_2)' = \text{true})\}$ 
27 end
28
29  $l : Ex(p, N_1, N_2) \Rightarrow$ 
30 in
31    $\{[l]l \rightarrow p : (l' = \text{false}) \& (L(N_1)' = \text{true}) +$ 
32    $(1 - p) : (l' = \text{false}) \& (L(N_2)' = \text{true})\} \cup \Gamma(L(N_2)) \cup \Gamma(L(N_1))$ 
33 end
34
35
36  $l : \odot \Rightarrow$ 
37 in
38    $[l] \rightarrow \&_{i \in \mathcal{L}} (l' = \text{false});$ 
39 end
40
41  $l : \otimes \Rightarrow$ 
42 in
43    $[l] \rightarrow (l' = \text{false});$ 
44 end
45
46  $l : a \rightarrow N \Rightarrow$ 
47 in
48    $\{[l_a]l \rightarrow (l'_a = \text{false}) \& (L(N)' = \text{true});\} \cup \Gamma(L(N))$ 
49 end
50

```

6.2. The transformation soundness

To prove the soundness of the transformation algorithm Γ , let \mathcal{A} be a DAC term specifying a system modeled in SysML activity diagram and $M_{\mathcal{A}}$ is its semantics corresponding to a DTMC constructed by the DAC operational semantics denoted by \mathcal{S} such that $\mathcal{S}(\mathcal{A}) = M_{\mathcal{A}}$. The generated program \mathcal{P} through the transformation rules, let $M_{\mathcal{P}}$ be its corresponding DTMC constructed by PRISM operational semantics denoted \mathcal{S}' such that $\mathcal{S}'(\mathcal{P}) = M_{\mathcal{P}}$. Proving the soundness of Γ algorithm means that a system property ϕ that should be satisfied on a system modeled in SysML and denoted by \mathcal{A} should be always satisfied after the transformation when producing the program \mathcal{P} and vice versa. Thus, Definition 4 specifies the strong bi-similarity as an adequate relation between both semantics of \mathcal{A} and \mathcal{P} , i.e. $M_{\mathcal{A}} \mathcal{R} M_{\mathcal{P}}$.

Definition 4 (Transformation relation). The relation $M_{\mathcal{A}} \mathcal{R} M_{\mathcal{P}}$ between a DAC term \mathcal{A} and a PRISM term \mathcal{P} such that $\Gamma(\mathcal{A}) = \mathcal{P}$ is a strong bi-simulation relation, where:

- Each initial state of M_1 is related to at least one initial state M_2 .
- For each pair of states $s_1 \mathcal{R} s_2$ and each transition $s_1 \rightarrow_{p_1} s'_1$ of either M_1 or M_2 there exists a transition $s_2 \rightarrow_{p_2} s'_2$ of either M_1 or M_2 , such that $p_1 = p_2$.

Γ is sound means the existence of a strong bi-simulation between $M_{\mathcal{A}}$ and $M_{\mathcal{P}}$, as well as it preserves the satisfiability of \mathcal{A} properties ϕ .

Proposition (PCTL preservation). For both \mathcal{A} and \mathcal{P} semantics, $M_{\mathcal{A}}$ and $M_{\mathcal{P}}$ DTMCs, such that $\Gamma(\mathcal{A}) = \mathcal{P}$ and $M_{\mathcal{A}} \mathcal{R} M_{\mathcal{P}}$, we have: $M_{\mathcal{A}} \models \phi \iff M_{\mathcal{P}} \models \phi$ where ϕ a PCTL property.

Proof. By following a structural induction on DAC terms and PCTL syntax, we prove $M_{\mathcal{A}} \sqsubseteq_{\mathcal{R}} M_{\mathcal{P}}$ and $M_{\mathcal{A}} \models \phi \iff M_{\mathcal{P}} \models \phi$ for a PCTL property ϕ . For two states $s_1, s'_1 \in \mathcal{S}_A$ and $s_2, s'_2 \in \mathcal{S}_P$. By induction on DAC terms, we found the following.

1. $L(s_1) = l : \bar{x} \mapsto \mathcal{N}$ and $x = \{i, a\} \implies \exists s_1 \rightarrow s'_1, L(s'_1) = l : x \mapsto \bar{\mathcal{N}}$. For $L(s_2) = \Gamma(L(s_1))$, we have $L(s_2) = \langle L(x), \neg L(\mathcal{N}) \rangle$ then $\exists s_2 \rightarrow s'_2$ where $L(s'_2) = \langle \neg L(x), L(\mathcal{N}) \rangle$. By taking the case of a state formula $\phi = l : \bar{x} \mapsto \mathcal{N}$, we have: $l : \bar{x} \mapsto \mathcal{N} \subseteq L(s_1) \cap L(s_2)$. Then: $s_1 \models \phi$ and $s_2 \models \phi$.
2. $L(s_1) = l : \bar{x} \mapsto \mathcal{N}$ where $x = \{a!v, a?v\} \implies \exists s_1 \rightarrow s'_1, L(s'_1) = l : x \mapsto \bar{\mathcal{N}}$. For $L(s_2) = \Gamma(L(s_1))$, we have $L(s_2) = \langle L(x), \neg L(\mathcal{N}) \rangle$ then $\exists s_2 \rightarrow s'_2$ and $L(s'_2) = \langle \neg L(x), L(\mathcal{N}) \rangle$. By considering the next operator where $\phi = l : a!v \mapsto \mathcal{N}$, then: $L(s'_1) = L(s'_2) = l : a!v \mapsto \bar{\mathcal{N}} : \bar{x} \mapsto \mathcal{N}$. Then: $s_1 \rightarrow s'_1 \models \phi$ and $s_2 \rightarrow s'_2 \models \phi$.
3. $L(s_1) = l : \overline{Ex(p, \mathcal{N}_1, \mathcal{N}_2)}$ then $\exists s_1 \rightarrow_{p_1} s'_1, L(s'_1) = l : Ex(p, \mathcal{N}_1, \mathcal{N}_2)$. For $L(s_2) = \Gamma(L(s_1))$, we have $L(s_2) = \langle l, \neg l_{\mathcal{N}_1}, \neg l_{\mathcal{N}_2} \rangle$ then $\exists s_2 \rightarrow_{p_2} s'_2$ where $L(s'_2) = \langle \neg l, l_{\mathcal{N}_1}, \neg l_{\mathcal{N}_2} \rangle$. In this case, we have to ensure the behaviour of transition, the evaluation of states in addition to the probabilistic values by considering $\phi = P[l : \overline{Ex(p, \mathcal{N}_1, \mathcal{N}_2)} \cup Ex(p, \mathcal{N}_1, \mathcal{N}_2)]$. Then, we have: $s_1 \models Ex(p, \mathcal{N}_1, \mathcal{N}_2)$, $s'_1 \models Ex(p, \mathcal{N}_1, \mathcal{N}_2)$, $s_2 \models \overline{Ex(p, \mathcal{N}_1, \mathcal{N}_2)}$, $s'_2 \models Ex(p, \mathcal{N}_1, \mathcal{N}_2)$, $P[l : \overline{Ex(p, \mathcal{N}_1, \mathcal{N}_2)} \cup Ex(p, \mathcal{N}_1, \mathcal{N}_2)] = p$ where $p_1 = p - 2$.
4. $L(s_1) = l : \overline{D(g_1, \mathcal{N}_1, \mathcal{N}_2)}^n$ then $\exists s_1 \xrightarrow{g_1}_{\rightarrow_1} s'_1, L(s'_1) = l : \overline{D(g_1, \mathcal{N}_1, \mathcal{N}_2)}^{n-1}$. For $L(s_2) = \Gamma(L(s_1))$, we have $L(s_2) = \langle l, \neg l_{\mathcal{N}_1}, \neg l_{\mathcal{N}_2} \rangle$ then $\exists s_2 \xrightarrow{g_1}_{\rightarrow_1} s'_2$ where $L(s'_2) = \langle \neg l, l_{\mathcal{N}_1}, \neg l_{\mathcal{N}_2} \rangle$. Proving the satisfiability of the decision node is similar to the expansion.
5. $L(s_1) = l : \odot$ then $\exists s_1 \rightarrow_1 s'_1, L(s'_1) = l : \odot$. For $L(s_2) = \Gamma(L(s_1))$, we have $L(s_2) = \langle l \rangle$ then $\exists s_2 \rightarrow_1 s'_2$ where $\forall l_i \in \mathcal{L} : L(s'_2) = \langle \neg l_i \rangle$. The satisfiability of final node is similar to the initial node.
6. $L(s_1) = l : \otimes$ then $\exists s_1 \rightarrow_1 s'_1, L(s'_1) = l : \otimes$. For $L(s_2) = \Gamma(L(s_1))$, we have $L(s_2) = \langle l \rangle$ then $\exists s_2 \rightarrow_1 s'_2$ where $L(s'_2) = \langle \neg l \rangle$. Based on this bi-similarity, it is clear that all PCTL operators satisfied on one DTMC are satisfied on the second one

From the obtained results, we found that $p_1 = p_2 = 1$ in case 1, 2, 4, 5, 6 and $p_1 = p_2 = p, 0 < p < 1$ in case 3 means $s_1 \sqsubseteq_{\mathcal{R}} s_2$. In addition, the unique initial state of $M_{\mathcal{A}}$ is always corresponding to the unique initial state in $M_{\mathcal{D}}$. By fond that $M_{\mathcal{A}} \sqsubseteq_{\mathcal{R}} M_{\mathcal{D}}$ and $M_{\mathcal{A}} \models \phi \iff M_{\mathcal{D}} \models \phi$ for all PCTL operator. Thus, Proposition 6.2 holds and Γ is sound. \square

7. Implementation and experimental results

In order to implement our methodology for deployment-space-exploration, an Eclipse plug-in has been developed. The graphical tool used for creating SysML/MARTE models is Eclipse with Acceleo¹ capabilities. After the generation of the corresponding XML file of the main design, the plug-in parses the document and generates PRISM code for each Activity diagram in the DTMC model in order to check the configuration against the PCTL property. Finally, the plug-in returns a message for the close-candidate that satisfies our reliability requirement. Our approach is illustrated in the case study of an embedded automotive control system designed according to (Meedeniya et al., 2011). The objective of the optimization is to maximize the reliability of the system (Section 3). To assess the performance of model checking process, two software components are deployed on dedicated hardware architecture, namely the Anti-lock Brake System (ABS) and Adaptive Cruise Control (ACC). It is assumed in our analysis that the information exchange among the components does not generate any information anomalies or abnormal signal transmission. However, the main anomalies come from the processor's failures according to our assumptions in the beginning. When the failure occurs, an exception node is activated to restart the block behaviour activity. In our specification, the physical platform is depicted in Figure 3. The parameters related to software components are gathered from papers (Meedeniya et al., 2011, 2012).

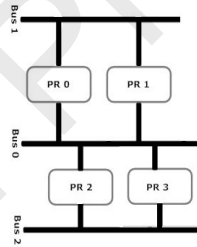


Figure 6: Hardware topology.

Hardware	Processing Speed[MIPS]	Failure Rate
PR 0	20	9.10^{-6}
PR 1	5	2.10^{-6}
PR 2	5	3.10^{-6}
PR 3	10	9.10^{-6}

Table 5: Parameters of processors.

Hardware Architecture: The hardware architecture used for the case study consists of four processors and three buses connecting the processors together as shown previously in Section 4- Figure 3 and then abstracted in Figure 6. Table 5 depicts the parameters related to processors (PR).

¹<https://www.eclipse.org/acceleo/>.

422 **Adaptive Cruise Control (ACC):** The aim of this component is to avoid crashes by reducing speed once
 423 the slower vehicle in front is detected. The main software components used by ACC are depicted in Figure 7
 424 and Figure 8 (Associated Activity diagram). Service initialization is possible at Object Recognition software
 425 that communicates with sensors. Speed Limit, Mode Switch, and Brake Paddle contribute to triggering of
 426 the service. The captured data are processed by the ACC_MAIN_UNIT and Human Machine Interface to
 427 communicate with actuators.

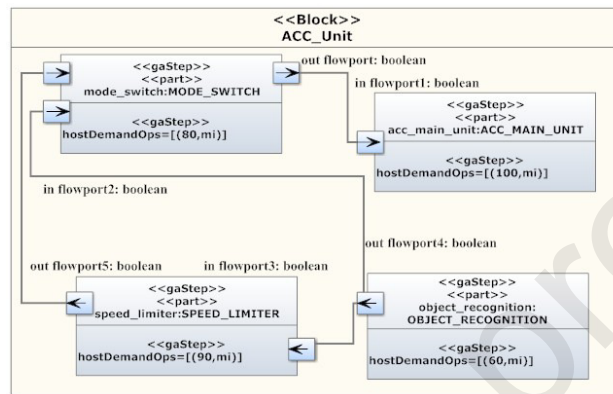


Figure 7: Adaptive cruise control.

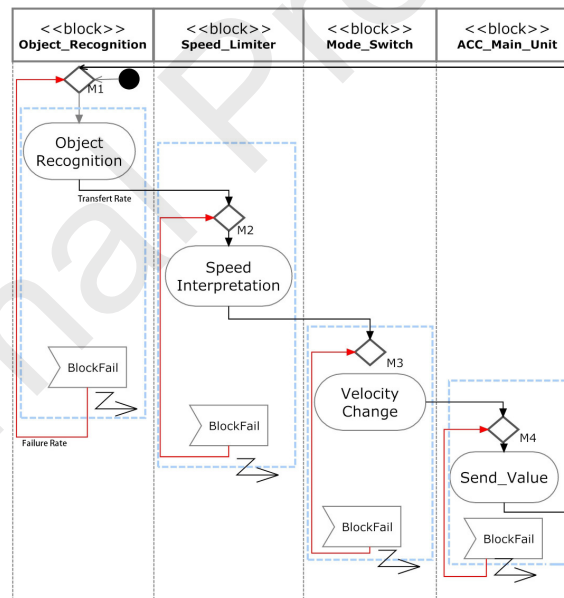


Figure 8: Activity diagram for adaptive cruise control.

428 **Anti-lock Brake System (ABS):** used in modern cars to minimize hazards associated with skidding
 429 and loss of control due to a locked wheel during braking. The software architecture of the ABS is depicted
 430 in Figure 9 and Figure 10 (Associated Activity diagram). The ABS Main unit is the major decision unit
 431 regarding the braking levels for individual wheels, while the Load Compensator unit assists with computing
 432 adjustment factors from wheel load sensor inputs. Component 4-7 (Table 6) represent transceiver software

433 components associated with each wheel, and communicate with sensors and brake actuators. *Brake Paddle*
 434 is the software component that reads from the *Paddle Sensor* and sends the event to the emergency stop
 435 detection module.

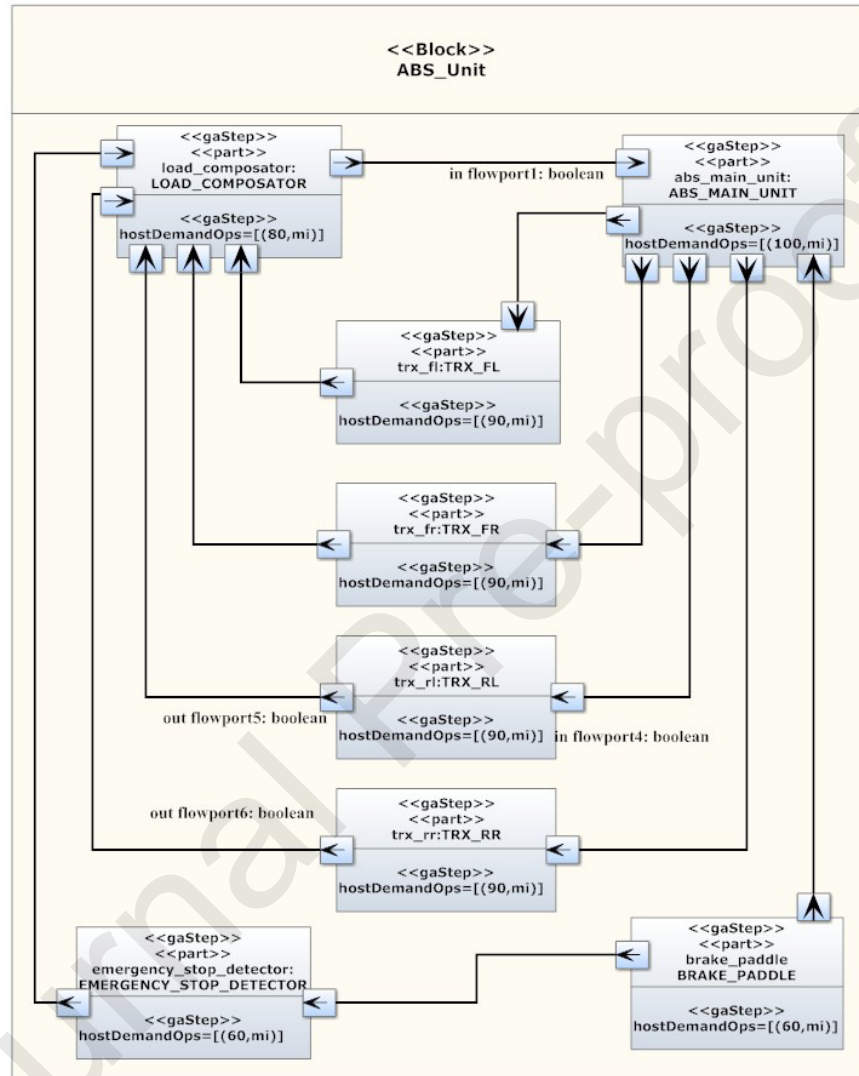


Figure 9: Anti-lock brake system.

Listing 4: ACC PRISM code fragment.

```

1  dtmc
2
3  const double P1;
4  const double P2;
5  const double P3;
6  const double P4;
7  module ACC
8  Initial : bool init true;
9  Object_Recognition: bool init false;
10 Speed_Interpretation: bool init false;
11 Velocity_Change : bool init false;
12 Send_Value : bool init false;
13 Final: bool init false;
14
15 Proc_Fail: bool init false;
16 .....
17
18 D1: bool init false;
19 D2: bool init false;
20 .....
21
22 M11: bool init false;
23 M12: bool init false;
24 M1: bool init false;
25 [Initial] Initial → (Initial' = false) & (M11' = true);
26 [M11] M11 → (M11' = false) & (M1' = true);
27 [M12] M12 → (M12' = false) & (M1' = true);
28 [M1] M1 → (M1' = false) & (Object_Recognition' = true);
29 [Object_Recognition] Object_Recognition → 1.0:(D1' = true) & (Object_Recognition' = false);
30
31 [D1] D1 → P1:(M21' = true) & (D1' = false) + (1-P1):(Block1_Fail' = true) & (D1' = false);
32 [Block1_Fail] Block1_Fail → (Block1_Fail' = false) & (M12' = true);
33 [Speed_Interpretation] Speed_Interpretation → 1.0:(SpeedInterpretation' = false) & (D2' = true);
34 [D2] D2 → P2:(M31' = true) & (D2' = false) + (1-P2):(Block2_Fail' = true) & (D2' = false);
35 [Block2_Fail] Block2_Fail → (Block2_Fail' = false) & (M22' = true);
36 [Velocity_Change] Velocity_Change → 1.0:(Velocity_Change' = false) & (D3' = true);
37 [D3] D3 → P3:(M41' = true) & (D3' = false) + (1-P3):(Block3_Fail' = true) & (D3' = false);
38 [Block3_Fail] Block3_Fail → (Block3_Fail' = false) & (M32' = true);
39 [Send_Value] Send_Value → 1.0:(Send_Value' = false) & (D4' = true);
40 [D4] D4 → P4:(Final' = true) & (D4' = false) + (1-P4):(Block3_Fail' = true) & (D4' = false);
41 [Block4_Fail] Block4_Fail → (Block4_Fail' = false) & (M42' = true);
42 endmodule
43
44 label "Proc_Fail" = Block1_Fail | Block2_Fail | Block3_Fail | Block4_Fail;

```

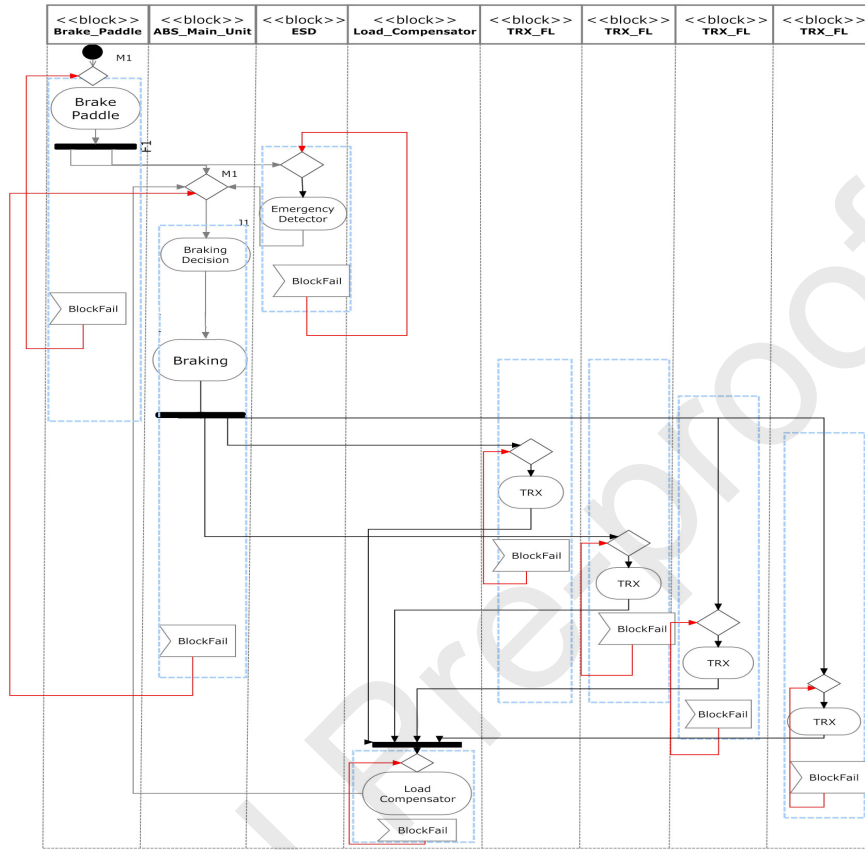


Figure 10: Activity diagram for anti-lock brake system.

i	SW component	$wl(c_i)[KB]$
0	ABS_MAIN_UNIT	60
1	EMERGENCY_STOP_DETECTOR	90
2	BRAKE_PADDLE	150
3	LOAD_COMPENSATOR	100
4	TRX_RR	90
5	TRX_RL	60
6	TRX_FR	90
7	TRX_FL	150
8	ACC_MAIN_UNIT	100
9	MODE_SWITCH	90
10	OBJECT_RECOGNITION	60
11	SPEED_LIMITER	90

Table 6: Software components parameters.

Hardware	Components
PR 0	4,11,9,7
PR 1	2,5,1
PR 2	3,6,0
PR 3	8,10

Table 7: Allocation results.

7.1. Evaluation

Despite the fact that the low number of components and states (i.e. PRISM states) generated after mapping, the deployment-space exploration took 1622 seconds, which is approximately 27 minutes. Our plug-in records each result of the checked model to get the minimum failure probability of each deployment candidate. During this process, 3256 candidates were explored and performed on Intel Core i7-4770@3.40GHz and 8.0GB of RAM. The Listing 4 shows the PRISM code that can be parametrized through the values of the double constants. The constant values are filled with our plug-in during the deployment-space-exploration (i.e., Model checking process).

The reliability property of ACC and ABS that we analyze using PRISM is: “the probability that the ACC and ABS will need to be replaced by a new one after T time steps” and expressed using PCTL property:

$$P = ?[true \cup^{\leq T} (“Proc_Fail”)], T = 200 \quad (5)$$

In the property above *Proc_Fail* asserts that the processor is in failing state when one of the software blocks fail according to the PRISM label :

$$label“Proc_Fail” = Block1_Fail \vee Block2_Fail \vee Block3_Fail \vee Block4_Fail \quad (6)$$

The analysis results of reliability property obtained from PRISM are as follows: The result of the property in case of ACC is shown in Figure 11; the result of property in case of ABS is shown in Figure 12. For better deployment; we choose the minimum probability in case of failure. In Table 7, we show the best software blocks candidate relative to the probabilistic results. In the first figure (ACC), the failure occurs after 200 time steps with probability near to 0.542% and could be deployed on two processors where the second figure (ABS), the failure occurs after 200 time steps with probability near to 0.579% and could be deployed on three processors. The cases where the blocks deployed on one processor *are not considered* in our approach.

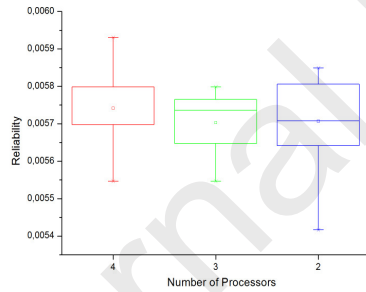


Figure 11: ACC reliability.

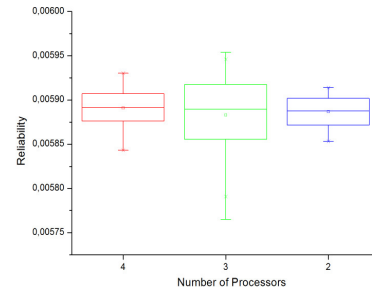


Figure 12: ABS reliability.

7.2. System constraints and optimization

In our experiment, we have accepted a few properties to make our model simple and applicable to the larger system. The software failure is not considered, and they are assumed independent to the deployment. All the deployment parameters could be obtained (e.g. Processor speed) except processors failing parameters. Failure parameters could be estimated by experts using profiling (Houssin and Coulibaly, 2014).

One of the challenges in applying probabilistic model checking is *scalability*: the models need to be constructed and analyzed are time-consuming and affects the design-space exploration. This phenomenon is called *state-explosion problem*.

So, we apply the rules detailed in (Ouchani et al., 2014a) to show the abstraction results on the SysML Activity diagram in Figure 10 (i.e., we refer to the original model as a *concrete model*) with the best deployment candidate parameters values. The abstraction approach produces a reduced activity diagram without

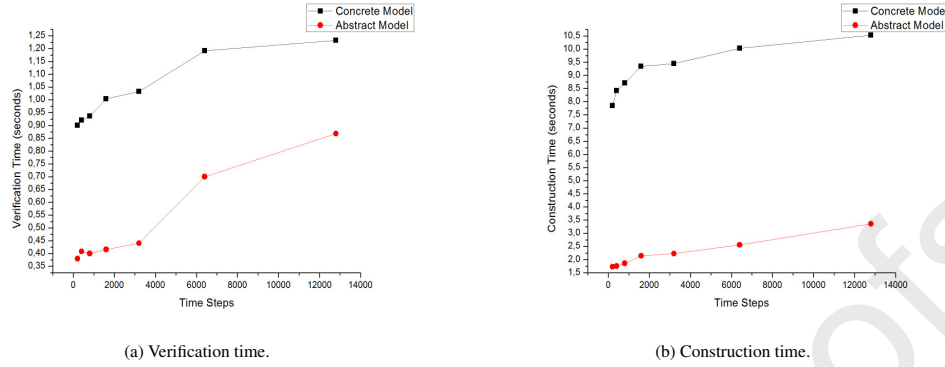


Figure 13: The abstraction effects on ABS diagram.

Time Steps	Tc	Concrete Model Tv	Tc	Abstract Model Tv	Results
200	7.844	0.900	1.728	0.380	0.00579
400	8.720	0.920	1.756	0.408	0.0169
800	8.416	0.936	1.856	0.400	0.0233
1600	9.348	1.004	2.144	0.416	0.0462
3200	9.448	1.032	2.224	0.440	0.0905
6400	10.036	1.192	2.560	0.800	0.1729
12800	10.532	1.232	3.360	1.168	0.3160

Table 8: Abstraction results.

irrelevant actions called *abstracted model* (i.e. unreachable actions) based on the PCTL property. The result is shown in Table 8 presents its different verification results in function of time steps T (i.e. Path length). To compute the verification cost depicted in Figure 13, we measure the time required to construct the model and the time required for model checking, denoted by Tc and Tv, respectively. The number of states for the concrete model and the abstracted model is 16461 and 3025 states, respectively. The number of transitions for the concrete model and the abstracted model is 93412 and 15491 transitions, respectively. Also, We observe that the abstraction rules preserve the results while verification and construction time are optimized.

Also, the PRISM model checker implements a mechanism to handle the state explosion using binary decision diagrams (Kwiatkowska et al., 2012). This engine enables probabilistic verification of models up to 10^{10} states. PRISM includes advanced techniques for model abstraction based on stochastic games (Kwiatkowska et al., 2009a). It also supports an approximate probabilistic model checking using statistical methods. So, with the capability of PRISM engine, it is possible to analyze larger systems using our methodology.

8. Related work

In this section, we highlight the recent works related to deployment, partitioning, and allocation optimization, and then we compare them with our proposed approach.

Meedeniya et al. (2011) present an approach for reliability optimization in case of software deployment. The inputs are a set of hardware and software constraints (i.e. physical failures, software workload, deployment restrictions) that are considered as inputs for the reliability function. The approach uses Genetic Algorithms (GA) (Sivanandam and Deepa, 2010) to optimize the reliability function. This approach is dedicated to being integrated as framework for AADL language (Architecture Analysis and Design Language) (Feiler, 2010) but the authors did not explain how they integrated it in the top of language as an extended property since AADL supports it.

Thiruvady et al. (2014) identify the software-to-hardware assignments that maximize the reliability of the system with respect to constraints. These constraints include memory, physical platform failure and communication between software components. The deployment-space exploration is based on Ant Colony Optimization (ACO) combined with constraint programming (CP). However, a predefined number of solutions are constructed to apply CP which is hard for ample space of deployment candidates

Herrera et al. (2014) present the COMPLEX approach for hardware/software partitioning and allocation to a specific physical platform (i.e. software and hardware). The approach is based on a customized profile. The profile is a UML/MARTE with additional annotation to specify a design-space allocation. However, the authors focus more on the design specification than on partitioning. Also, the authors do not provide any information about the algorithm or applied strategy.

do Nascimento et al. (2012) present a framework for software- hardware mapping and code generation based on Model-Driven Engineering (MDE) proposed by the Object Management Group (OMG). The approach starts from UML class and sequence diagrams annotated using MARTE profile and use the network of timed automata (i.e. behaviour) as input to the UPPAAL (Behrmann et al., 2004) model checking tool for temporal property satisfaction. For design-space exploration, H-SPEX DSE tool (Oliveira et al., 2007) processes the mapped model. The core of the tool is based on Ant Colony Optimization. However, the authors do not clarify the optimization strategy that is abstracted from the user view.

Brosse et al. (2012) present ENOSYS design flow for modelling and synthesis of embedded systems. The approach consists of four consecutive steps, respectively named Modeling, Synthesis, Source Code Optimization and Design Space Exploration. The modelling step is based on UML Composite diagrams (i.e. Components representations) with the core behaviour using a state machine or activity diagrams. The composite diagram is enriched with a set of annotations using MARTE profile to express the software, hardware components and allocations. Automatic partitioning of the system into software and hardware components occurs when software code is generated and executed to obtain performances, area and power figures. These performance characteristics are checked against the required objectives. The process is executed until the required constraints are met. Then, the hardware objects are synthesized directly to VHDL/Verilog, whereas the software aspects are compiled for the multicore CPU platform (i.e. C/C++) which forms the programmable domain of the system implementation. The disadvantage of the approach is that the process of exploration occurs at the code level, which is time-consuming.

Martínez-Álvarez et al. (2013) propose a multi-objective optimization tool with the Software Hardening Environment for the fault-tolerant embedded systems design. The tool automatically transforms a source code (application) into a selective hardened version of the program and return a set of valuable information about the code and execution time overheads, and the fault coverage provided. The optimization tool applies a Genetic Algorithm (GA) to fulfil a set of given criteria of interest, taking into account fault tolerance parameters.

Besnard et al. (2015) present a verification and validation framework of embedded software using AADL and its behavioural annex. The specification is based on AADL language (Architecture Analysis and Design Language) (Feiler, 2010) (Baouya et al., 2019) and Simulink (Simulink Link) for functional behaviour. The authors implement a systematic translation of the AADL standard and of Simulink diagrams in the multi-clocked synchronous semantics of the Signal data-flow language (Ma et al., 2013) that enables timing analysis, formal verification and simulation. Multiprocessor partitioning occurs when the allocation of threads to processors has been achieved with respect to timing constraints that result from scheduling analysis.

Nath and Datta (2014) address a partitioning of JPEG encoder (i.e. applied for obtaining high quality output from continuous-tone images) into software and hardware components. The approach is based on multi-objective optimization, taking into account mainly the execution time, the memory requirement, the power consumption. The process applied the genetic algorithm on the control data flow graph (CDFG). However, the specification is based on the control data flow graph (CDFG) of 22 components of JPEG encoder.

Posadas et al. (2014) present a methodology based on MDE approach to automatically synthesizing the software code of complex embedded systems specified using UML/MARTE model for components diagrams and C code for functional behaviour. The synthesis process enables the exploration of different

allocations of software components in real physical platforms, including processors and memories. By identifying the memory spaces of the components, their interfaces and allocations, it is possible to generate binary files for different allocations from the same inputs and chose which has a positive impact on the total performance of the system. The approach is not interesting since the decision is made at the synthesis level.

Jia et al. (2014) propose a two-phase approach to optimal mapping of real-time application on Multiprocessor System-On-chip (MPSoC) platform while considering multiple design constraints. The first phase is heuristic-based for a rapid pruning of the large design space based on partitioning, scheduling and assignment. The reduced number of potential solutions are simulated. Three objectives are optimized: Maximize efficiency in the utilization of processing elements, Minimize the load unbalancing in processing elements and Minimize the communication traffic. The input of the optimization is a set of software and hardware parameters such as real-time constraints and communication delays except for the failures of the processing units that are not considered.

Qadri et al. (2016) have applied Fuzzy logic to derive a multicore architecture based on workload requirements and an optimum balance between throughput and energy of the System-on-Chips. In order to achieve this, control parameters are modified dynamically in a reconfigurable SoC, i.e. Number of cores, Operating frequency and Cache size. Consequently, this approach requires the availability of an appropriate mathematical model for energy estimation in multicore architectures.

8.1. Comparison

As a summary, in Table 9 we compare our framework to the existing works by taking into consideration six criteria: specification language, the addressed problem, applied algorithm, formalization, soundness, and automation. The Specification criteria show if the design is based on a clear user view (i.e. Languages and Models). The second criteria focus on the decision problem (i.e. deployment and partitioning) that the authors want to solve. The applied algorithm indicates the procedure used to solve the problem. Formalization criteria check if the approach provides semantics and formalizes the studied diagrams. Soundness feature checks if the mapping is sound. Automation criteria show if the approach is implemented. According to the studied papers, a few works formalize the specification diagrams, including the model checking as a decision tool for deployment case. Our work has for objective to support: component-based specification using SysML/MARTE and formalizing the SysML activity diagram. Also, we implement a tool that allows the automatic deployment decision.

9. Discussion and threats to validity

9.1. Use of SysML

Through the application of our methodology, we found that SysML is an acceptable language in industrial contexts since it is a good fit for capturing behavioural and structural aspects of the system in our study. Nevertheless, using SysML for complex design with accurate structural properties is not feasible. So, we found the extendability aspect of SysML language to be advantageous for system engineering. The designer can develop its proper profile to support specific properties. In the case of our methodology, we introduce the MARTE profile, as shown in Figure 3. We found that profile very useful for specifying hardware and software properties to perform our analysis.

9.2. Architecture-level alternatives

The main observation from the experimental results confirms our proposed idea based on the parameterizable Markov model (Filieri et al., 2016) has an impact on the system reliability. The proposed methodology explores all the architectural alternatives to select a suitable one that fulfils the reliability requirement expressed in temporal logic. The inputs parameters of each alternative are computed from the properties of our automotive electronic system. The overhead of such computation is negligible in our experiments since it occurred before the design space exploration.

Approach	Specification	Problem	Algorithm	Formalization	Soundness	Automation
Herrera et al. (2014)	UML/MARTE	Partitioning				✓
Thiruvady et al. (2014)		Deployment	Ant Colony Optimisation			✓
Meedeniya et al. (2011)		Deployment	Genetic Algorithm			✓
do Nascimento et al. (2012)	UML/MARTE	Deployment	Ant Colony Optimisation	✓		✓
Brosse et al. (2012)	UML/MARTE	Partitioning				✓
Martínez-Álvarez et al. (2013)		Requirement Optimization	Genetic Algorithm			
Besnard et al. (2015)	AADL/Simulink	Allocation	Scheduling Algorithm			✓
Nath and Datta (2014)		Partitioning	Genetic Algorithm			✓
Posadas et al. (2014)	UML/MARTE	Deployment				✓
Qadri et al. (2016)		Deployment	Fuzzy Logic			✓
Jia et al. (2014)	SysML/MARTE	Partitioning				✓
Our		Deployment	Model Checking	✓	✓	✓

Table 9: Comparison with the existing approaches for design-space exploration.

9.3. Threats to validity

The approach discussed in this paper focuses on the reliability that is hardware-dependent. The utility of the approach is to derive the optimal deployment candidate based on a few relevant metrics discussed in the published paper (Meedeniya et al., 2011). To handle the automotive system failures, our approach expresses the system-services flow through the activity diagram partitioned on software components. We assume that the external environment factors do not influence the system reliability. Also, the problem of power and energy consumption still exists, which limits the performance and reliability of our system. These issues deserve attention in our future work.

10. Conclusion

In this paper, we presented a deployment-exploration approach of embedded software modelled by using the SysML internal blocks diagram. The first objective is to validate a deployment in case of hardware failures that generally emerge in system design-flow and secondly to reduce the cost of maintenance and repairing. The proposed approach uses the SysML blocks diagram with additional real-time system annotations using the MARTE profile where behaviour is expressed using the SysML Activity diagram. Compared to (Meedeniya et al., 2011) and (Thiruvady et al., 2014), which use genetic algorithms, our approach leverages probabilistic modeling, thus allowing the assessment of properties formally expressed in probabilistic temporal logic. With respect to (Besnard et al., 2015), which uses a scheduling algorithm for allocation in AADL, leads to the difficulty to obtain right partitions due to the constraints limited to time, we employ a probabilistic formula based on reliability-relevant attributes. Moreover, in contrast to (Brosse et al., 2012), which is based on code generated for performance estimation area and early power figures, our approach extracts the relevant information directly from the specification and maps the behaviour to PRISM for model checking.

The contribution of the presented work focuses on capturing the system behaviour as Discrete-Time Markov chains (DTMC) supported by the PRISM language. We proposed a calculus dedicated to flow observation on blocks behaviour that captures precisely the underlying semantics. So, the PRISM language is formalized and its semantics highlighted. Then, we proved that the mapping approach is sound by defining the relation between the semantics of diagrams and the resulting PRISM models. The relation also preserves the satisfiability of PCTL properties. We have shown the effectiveness of our approach in a realistic case study: Embedded automotive control system.

The practical advantage of the proposed approach in the context of deployment is to provide a crucial decision for the acceptable candidate via probabilistic model checking. In addition, the results of our approach using PRISM could be plotted as graphs to be inspected for interpretations and anomalies.

Future works can take two directions. First, we intend to extend our approach to support more constraints that affect the entire system behaviour such as energy, busloads, and memory restrictions. Finally, we planned to translate the system into a variety of programming/description languages such as VHDL/C/C++/NesC for simulation.

References

- OMG Systems Modeling Language (Object Management Group SysML). O. M. Group (Ed.), 2012.
- Jean-Paul Arcangeli, Raja Boujbel, and Sébastien Leriche. Automatic deployment of distributed software systems: Definitions and state of the art. *Journal of Systems and Software*, 103(0):198 – 218, 2015. ISSN 0164-1212.
- Abdelhakim Baouya, Djamel Bennouar, Otmame Ait Mohamed, and Samir Ouchani. A quantitative verification framework of sysml activity diagrams under time constraints. *Expert Systems with Applications*, 42(21):7493 – 7510, 2015. ISSN 0957-4174.
- Abdelhakim Baouya, Djamel Bennouar, Otmame Ait Mohamed, and Samir Ouchani. *A Formal Approach for Maintainability and Availability Assessment Using Probabilistic Model Checking*, pages 295–309. Springer International Publishing, Cham, 2016. ISBN 978-3-319-33410-3. doi: 10.1007/978-3-319-33410-3_21.
- Abdelhakim Baouya, Otmame Ait Mohamed, Djamel Bennouar, and Samir Ouchani. Safety analysis of train control system based on model-driven design methodology. *Computers in Industry*, 105:1 – 16, 2019. ISSN 0166-3615.
- Abdelhakim Baouya, Chehida Salim, Bensalem Saddek, and Bozga Marius. *Formal Modeling and Verification of Blockchain Consensus Protocol for IoT Systems*, pages 330–342. IOS Press, 2020.

- 637 Gerd Behrmann, Alexandre David, and Kim G. Larsen. A tutorial on uppaal. In Marco Bernardo and Flavio Corradini, editors, *Formal*
638 *Methods for the Design of Real-Time Systems*, volume 3185 of *Lecture Notes in Computer Science*, pages 200–236. Springer Berlin
639 Heidelberg, 2004. ISBN 978-3-540-23068-7.
- 640 Loïc Besnard, Adnan Bouakaz, Thierry Gautier, Paul Le Guernic, Yue Ma, Jean-Pierre Talpin, and Huafeng Yu. Timed behavioural
641 modelling and affine scheduling of embedded software architectures in the {AADL} using polychrony. *Science of Computer*
642 *Programming*, 106:54 – 77, 2015. ISSN 0167-6423. Special Issue: Architecture-Driven Semantic Analysis of Embedded Systems.
- 643 Etienne Brosse, Imran Quadri, Andrey Sadovykh, Frank Ieromnimon, Dimitrios Kritharidis, Rafael Catrou, and Michel Sarlotte.
644 Enosys fp7 eu project: An integrated modeling and synthesis flow for embedded systems design. In *Reconfigurable Communication-*
645 *centric Systems-on-Chip (ReCoSoC), 2012 7th International Workshop on*, pages 1–5, July 2012.
- 646 Jan Carlson, John Håkansson, and Paul Pettersson. Saveccm: An analysable component model for real-time systems. *Electronic*
647 *Notes in Theoretical Computer Science*, 160:127 – 140, 2006. ISSN 1571-0661. Proceedings of the International Workshop on
648 Formal Aspects of Component Software (FACS 2005) Proceedings of the International Workshop on Formal Aspects of Component
649 Software (FACS 2005).
- 650 Abraham Cherfi, Michel Leeman, Florent Meurville, and Antoine Rauzy. Modeling automotive safety mechanisms: A markovian
651 approach. *Reliability Engineering & System Safety*, 130:42 – 49, 2014. ISSN 0951-8320. doi: [http://dx.doi.org/10.1016/j.ress.](http://dx.doi.org/10.1016/j.ress.2014.04.013)
652 [2014.04.013](http://www.sciencedirect.com/science/article/pii/S0951832014000817). URL <http://www.sciencedirect.com/science/article/pii/S0951832014000817>.
- 653 Lenny Delligatti. *SysML Distilled: A Brief Guide to the Systems Modeling Language*. Addison-Wesley Professional, 1st edition, 2013.
654 ISBN 0321927869, 9780321927866.
- 655 Francisco Assis Moreira do Nascimento, Marcio F.S. Oliveira, and Flávio Rech Wagner. A model-driven engineering framework for
656 embedded systems design. *Innovations in Systems and Software Engineering*, 8(1):19–33, 2012. ISSN 1614-5046.
- 657 Peter Feiler. Model-based validation of safety-critical embedded systems. In *Aerospace Conference, 2010 IEEE*, pages 1–10, March
658 2010.
- 659 Antonio Filieri, Giordano Tamburrelli, and Carlo Ghezzi. Supporting self-adaptation via quantitative verification and sensitivity anal-
660 ysis at run time. *IEEE Transactions on Software Engineering*, 42(1):75–99, 2016.
- 661 Areski Flissi, Jérémy Dubus, Nicolas Dolet, and Philippe Merle. Deploying on the grid with deployware. In *Eighth IEEE International*
662 *Symposium on Cluster Computing and the Grid*, pages 177–184, France, 2008.
- 663 Vojtěch Forejt, Marta Kwiatkowska, Gethin Norman, and David Parker. *Formal Methods for Eternal Networked Software Systems:*
664 *11th International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM 2011,*
665 *Bertinoro, Italy, June 13-18, 2011. Advanced Lectures*, chapter Automated Verification Techniques for Probabilistic Systems, pages
666 53–113. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. ISBN 978-3-642-21455-4.
- 667 Joao M. Franco, Francisco Correia, Raul Barbosa, Mario Zenha-Rela, Bradley Schmerl, and David Garlan. Improving self-adaptation
668 planning through software architecture-based stochastic modeling. *Journal of Systems and Software*, 115:42 – 60, 2016. ISSN
669 0164-1212.
- 670 Sanford Friedenthal, Alan Moore, and Rick Steiner. *A Practical Guide to SysML: Systems Modeling Language*. Morgan Kaufmann
671 Publishers Inc., San Francisco, CA, USA, 2008. ISBN 0123743796, 9780080558363, 9780123743794.
- 672 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman
673 & Co., New York, NY, USA, 1979. ISBN 0716710447.
- 674 Günter Heiner and Thomas Thurner. Time-triggered architecture for safety-related distributed real-time systems in transportation
675 systems. In *Digest of Papers: FTCS-28, The Twenty-Eighth Annual International Symposium on Fault-Tolerant Computing, Munich,*
676 *Germany, June 23-25, 1998*, pages 402–407, 1998. doi: 10.1109/FTCS.1998.689491.
- 677 Fernando Herrera, Héctor Posadas, Pablo Peñil, Eugenio Villar, Francisco Ferrero, Raúl Valencia, and Gianluca Palermo. The {COM-
678 PLEX} methodology for uml/marte modeling and design space exploration of embedded systems. *Journal of Systems Architecture*,
679 60(1):55 – 78, 2014. ISSN 1383-7621.
- 680 Khaza Anuarul Hoque, Othmane Ait Mohamed, Yvon Savaria, and Claude Thibeault. Early analysis of soft error effects for aerospace
681 applications using probabilistic model checking. In Cyrille Artho and Peter Csaba Á-Íveczky, editors, *Formal Techniques for*
682 *Safety-Critical Systems*, volume 419 of *Communications in Computer and Information Science*, pages 54–70. Springer International
683 Publishing, 2014. ISBN 978-3-319-05415-5.
- 684 Remy Houssin and Amadou Coulibaly. Safety-based availability assessment at design stage. *Computers & Industrial Engineering*, 70:
685 107 – 115, 2014. ISSN 0360-8352. doi: <http://dx.doi.org/10.1016/j.cie.2014.01.005>.
- 686 Zai J. Jia, Antonio Núñez, Tomás Bautista, and Andy D. Pimentel. A two-phase design space exploration strategy for system-level
687 real-time application mapping onto mp soc. *Microprocessors and Microsystems*, 38(1):9–21, 2014.
- 688 Gideon Juve and Ewa Deelman. Automating application deployment in infrastructure clouds. In *Cloud Computing Technology and*
689 *Science (CloudCom), 2011 IEEE Third International Conference on*, pages 658–665, 2011.
- 690 Marta Kwiatkowska, Gethin Norman, and António Pacheco. *Process Algebra and Probabilistic Methods: Performance Modeling and*
691 *Verification: Second Joint International Workshop PAPM-PROBMIV 2002 Copenhagen, Denmark, July 25–26, 2002 Proceedings,*
692 *chapter Model Checking CSL until Formulae with Random Time Bounds*, pages 152–168. Springer Berlin Heidelberg, Berlin,
693 Heidelberg, 2002. ISBN 978-3-540-45605-6.
- 694 Marta Kwiatkowska, , Gethin Norman, and David Parker. Stochastic model checking. In M. Bernardo and J. Hillston, editors, *Formal*
695 *Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM’07)*, volume 4486 of
696 *LNCS (Tutorial Volume)*, pages 220–270. Springer, 2007a.
- 697 Marta Kwiatkowska, Gethin Norman, and David Parker. Controller dependability analysis by probabilistic model checking. *Control*
698 *Engineering Practice*, 15(11):1427 – 1434, 2007b. ISSN 0967-0661. Special Issue on Manufacturing Plant Control: Challenges
699 and Issues.
- 700 Marta Kwiatkowska, Gethin Norman, and David Parker. Stochastic games for verification of probabilistic timed automata. In J. Ouak-
701 nine and F. Vaandrager, editors, *Formal Modeling and Analysis of Timed Systems*, volume 5813 of *Lecture Notes in Computer*

- Science, pages 212–227. Springer Berlin Heidelberg, 2009a. ISBN 978-3-642-04367-3.
- Marta Kwiatkowska, Gethin Norman, and David Parker. Prism: Probabilistic model checking for performance and reliability analysis. *SIGMETRICS Perform. Eval. Rev.*, 36(4):40–45, March 2009b. ISSN 0163-5999.
- Marta Kwiatkowska, Gethin Norman, and David Parker. Prism 4.0: Verification of probabilistic real-time systems. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Computer Aided Verification*, volume 6806 of *Lecture Notes in Computer Science*, pages 585–591. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-22109-5.
- Marta Kwiatkowska, , and David Parker. Advances in probabilistic model checking. In T. Nipkow, O. Grumberg, and B. Hauptmann, editors, *Software Safety and Security - Tools for Analysis and Verification*, volume 33 of *NATO Science for Peace and Security Series - D: Information and Communication Security*, pages 126–151. IOS Press, 2012.
- Yu Lu, Zhaoguang Peng, Alice A. Miller, Tingdi Zhao, and Christopher W. Johnson. How reliable is satellite navigation for aviation? checking availability properties with probabilistic verification. *Reliability Engineering & System Safety*, 144:95 – 116, 2015. ISSN 0951-8320.
- Yue Ma, Huafeng Yu, Thierry Gautier, Paul Le Guernic, Jean-Pierre Talpin, Loïc Besnard, and Maurice Heitz. Toward polychronous analysis and validation for timed software architectures in aadl. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '13, pages 1173–1178, San Jose, CA, USA, 2013. EDA Consortium. ISBN 978-1-4503-2153-2.
- Frédéric Mallet and Robert de Simone. Marte: A profile for rt/e systems modeling, analysis—and simulation? In *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, Simutools '08, pages 43:1–43:8, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). ISBN 978-963-9799-20-2.
- Antonio Martínez-Álvarez, Felipe Restrepo-Calle, Luis Alberto Vivas Tejuelo, and Sergio Cuenca-Asensi. Fault tolerant embedded systems design by multi-objective optimization. *Expert Systems with Applications*, 40(17):6813 – 6822, 2013. ISSN 0957-4174.
- Bernhard Mattes. Occupant protection systems. In Konrad Reif, editor, *Brakes, Brake Control and Driver Assistance Systems*, Bosch Professional Automotive Information, pages 162–179. Springer Fachmedien Wiesbaden, 2014. ISBN 978-3-658-03977-6.
- Indika Meedeniya, Barbora Buhnova, Aldeida Aleti, and Lars Grunske. Reliability-driven deployment optimization for embedded systems. *Journal of Systems and Software*, 84(5):835 – 846, 2011. ISSN 0164-1212.
- Indika Meedeniya, Aldeida Aleti, and Lars Grunske. Architecture-driven reliability optimization with uncertain model parameters. *Journal of Systems and Software*, 85(10):2340 – 2355, 2012. ISSN 0164-1212.
- Robin Milner. *Communicating and Mobile Systems: The π -calculus*. Cambridge University Press, New York, NY, USA, 1999. ISBN 0-521-65869-1.
- Pankaj Kumar Nath and Dilip Datta. Multi-objective hardware–software partitioning of embedded systems: A case study of {JPEG} encoder. *Applied Soft Computing*, 15:30 – 41, 2014. ISSN 1568-4946.
- Shiva Nejati, Mehrdad Sabetzadeh, Davide Falessi, Lionel Briand, and Thierry Coq. A sysml-based approach to traceability management and design slicing in support of safety certification: Framework, tool support, and case studies. *Information and Software Technology*, 54(6):569–590, 2012.
- Marcio F. S. Oliveira, Eduardo W. Brião, Francisco A. Nascimento, and Flávio R. Wagner. Model driven engineering for mp soc design space exploration. In *Proceedings of the 20th Annual Conference on Integrated Circuits and Systems Design*, SBCCI '07, pages 81–86, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-816-9.
- Samir Ouchani, Othmane Ait Mohamed, and Mourad Debbabi. A property-based abstraction framework for sysml activity diagrams. *Knowledge-Based Systems*, 56(0):328–343, 2014a.
- Samir Ouchani, Othmane Ait Mohamed, and Mourad Debbabi. A formal verification framework for sysml activity diagrams. *Expert Systems with Applications*, 41(6):2713 – 2728, 2014b. ISSN 0957-4174.
- Zhaoguang Peng, Yu Lu, Alice Miller, Tingdi Zhao, and Chris Johnson. Formal specification and quantitative analysis of a constellation of navigation satellites. *Quality and Reliability Engineering International*, 32(2):345–361, 2014.
- Héctor Posadas, Pablo Peñil, Alejandro Nicolás, and Eugenio Villar. Automatic synthesis of embedded {SW} for evaluating physical implementation alternatives from uml/marte models supporting memory space separation. *Microelectronics Journal*, 45(10):1281 – 1291, 2014. ISSN 0026-2692. DCIS'12 Special Issue.
- Muhammad Yasir Qadri, Nadia N. Qadri, and Klaus D. McDonald-Maier. Fuzzy logic based energy and throughput aware design space exploration for mp socs. *Microprocessors and Microsystems*, 40:113–123, 2016.
- Layali Rashid, Karthik Pattabiraman, and Sathish Gopalakrishnan. Characterizing the impact of intermittent hardware faults on programs. *Reliability, IEEE Transactions on*, 64(1):297–310, March 2015. ISSN 0018-9529.
- Tripti Saxena and Gabor Karsai. A meta-framework for design space exploration. In *Engineering of Computer Based Systems (ECBS), 2011 18th IEEE International Conference and Workshops on*, pages 71–80, April 2011.
- Simulink Link. <http://www.mathworks.com/simulink>.
- S. N. Sivanandam and S. N. Deepa. *Introduction to Genetic Algorithms*. Springer Publishing Company, Incorporated, 1st edition, 2010. ISBN 3642092241, 9783642092244.
- Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2002. ISBN 0201745720.
- Dhananjay Thiruvady, I. Moser, Aldeida Aleti, and Asef Nazari. Constraint programming and ant colony system for the component deployment problem. *Procedia Computer Science*, 29(0):1937 – 1947, 2014. ISSN 1877-0509. 2014 International Conference on Computational Science.
- Fengling Zhang, Lei Bu, Linzhang Wang, Jianhua Zhao, Xin Chen, Tian Zhang, and Xuandong Li. Modeling and evaluation of wireless sensor network protocols by stochastic timed automata. *Electronic Notes in Theoretical Computer Science*, 296(0):261 – 277, 2013. ISSN 1571-0661. Proceedings the Sixth International Workshop on the Practical Application of Stochastic Modelling (PASM) and the Eleventh International Workshop on Parallel and Distributed Methods in Verification (PDMC).