Available online at www.sciencedirect.com

ScienceDirect

Information & Software Technology 00 (2017) 1–27

Information & Software Technology

www.elsevier.com/locate/procedia

Reliability analysis based on probabilistic model checking for software deployment in embedded systems

Abdelhakim Baouya[a,*], Djamal Bennouar[b], Otmane Ait Mohamed[c], Samir Ouchani[d]

[a] *University of BLIDA 1 , Blida , Algeria, LIMPAF & LRDSI Lab, CS Department*
[b] *University of Bouira, Algeria, LIMPAF Lab, CS Department*
[c] *Concordia University, Montréal, Canada, ECE Department*
[d] *University of Luxembourg, Luxembourg, SnT Center*

Abstract

Context: Todays, embedded systems span a wide range from a small platform of sensors and actuators to a distributed systems combining a huge number of interacting nodes. Designing such systems includes hardware parts and software parts. Software parts gain more and more importance since it handles the complex embedded system functionality. The deployment of software components to hardware nodes is very hard and time-consuming to determine whether its design fulfills the reliability requirements.
Objective: To efficiently exploit the physical platform in the software development process, we introduce a novel approach in deployment-decision making based on PRISM probabilistic model checker that takes the software components and the physical platform to produce a set of deployment candidates.
Method: We construct our framework on System Modeling Language (SysML). The framework includes mechanisms to extract hardware and software properties and to produce the deployment candidates. To check the best one, each candidate should satisfy the reliability property that is expressed in Probabilistic Computation Tree Logic.
Results: We capture the underlying semantics of the software blocks behavior expressed as an activity diagram and their generated PRISM code to prove the soundness of the approach. We found that the probabilistic equivalence relation between both semantics preserves the satisfaction of the system requirements. We present the automotive control system as case study to illustrate the applicability of the proposed approach.
Conclusion: Our methodology provide designers with optimal deployment candidate based on the reliability attributes. This solution based on SysML and probabilistic model checking is implemented as a software tool.

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

1. Introduction

Many cities are more and more overcrowded and lead to the growing accidents and unpredicted emergencies. In response to that, the engineers have to open a way to improve the safety and efficiency. Several innovative and cost-effective solutions are emerging to simplify our daily-life so-called Automotive Control

*Corresponding author at : CS Department, University of BLIDA 1, Blida, Algeria. Tel.:+213 794 005 852
Email addresses: baouya.abdelhakim@gmail.com (Abdelhakim Baouya), dbennouar@gmail.com (Djamal Bennouar), otmane.aitmohamed@concordia.ca (Otmane Ait Mohamed), samir_ouchani@yahoo.com (Samir Ouchani)

Systems (ACS) (Mattes, 2014). The evolution of *Automotive Control Systems* 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. In one side these systems are increasingly complex, in the other side the hardware platform resources are often limited in memory, number of hardware interfaces, communication bandwidth, and so on, which make the task of its design hard. One major challenge in this process is to enhance the system reliability by taking in account the restricted physical resources as well as their failure. (Rashid et al., 2015) assert that the hardware faults have a negative impacts on the programs (i.e. software). So, considering these assertions, the high-level description will provide sufficient information to predict system reliability in our deployment plan.

Software deployment is a complex procedure known to be NP-hard (Garey and Johnson, 1979), the process consists in the distribution of a software components on different physical locations with respect to the requirements. The term *component* refers to an operational unit or modules consisting in set of operations where the assembled components represent a system. Note that the component-based development approach is a proven concept for managing complexity (Carlson et al., 2006), and it has been used also in embedded system design for a while (Herrera et al., 2014). In case of software-hardware deployment, we have to distinguish two set of components; hardware and software communicating using a proper interface. (Arcangeli et al., 2015) and (Saxena and Karsai, 2011) give a 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 with minimum failures. This issue is addressed by (Baouya et al., 2016) in order to increase reliability and to plan maintenance strategies of multi-processing systems. In our approach, we want to optimize the deployment with respect to system reliability computed from a set of quality attributes. 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 approach 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).

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

The basic idea is to first build a parameterizable Markov model (Filieri et al., 2016) that captures the software behavior 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 properties of our electronic automotive system and the results of the software-hardware allocations. In order to allow for automatic verification using PRISM (Kwiatkowska et al., 2011), the underlying *Discrete-time Markov Chains* (DTMC) semantic model of system behavior is constructed using the semantic rules. DTMC is used under the assumption that the system's behavior meets with some tolerable approximation of the Markov property, i.e. the probability of moving to the next state depends only on the current state, not on the history that lead to that state (Kwiatkowska et al., 2012). Fig.1 depicts all the steps for the deployment-space exploration (DSE). The specification consists on the SysML Internal Block Diagram (IBD) of both software and hardware components with its behavior 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 sub parts of automotive control system.

The remainder of this paper is structured as follows: Section 2 discusses briefly about Automotive Control Systems. Section 3 shows how the deployment is resolved according to the designer parameters. Section 4 reviews the related work. Section 5 describes the SysML blocks diagrams and its associated behavior. The PRISM model checker is presented in section 6. The internal blocks behavior expressed in activity diagram is formalized in Section 7. The semantics of PRISM models is presented in section 8. Section 9 provides a mapping algorithm of SysML activity Diagrams into PRISM code and its soundness

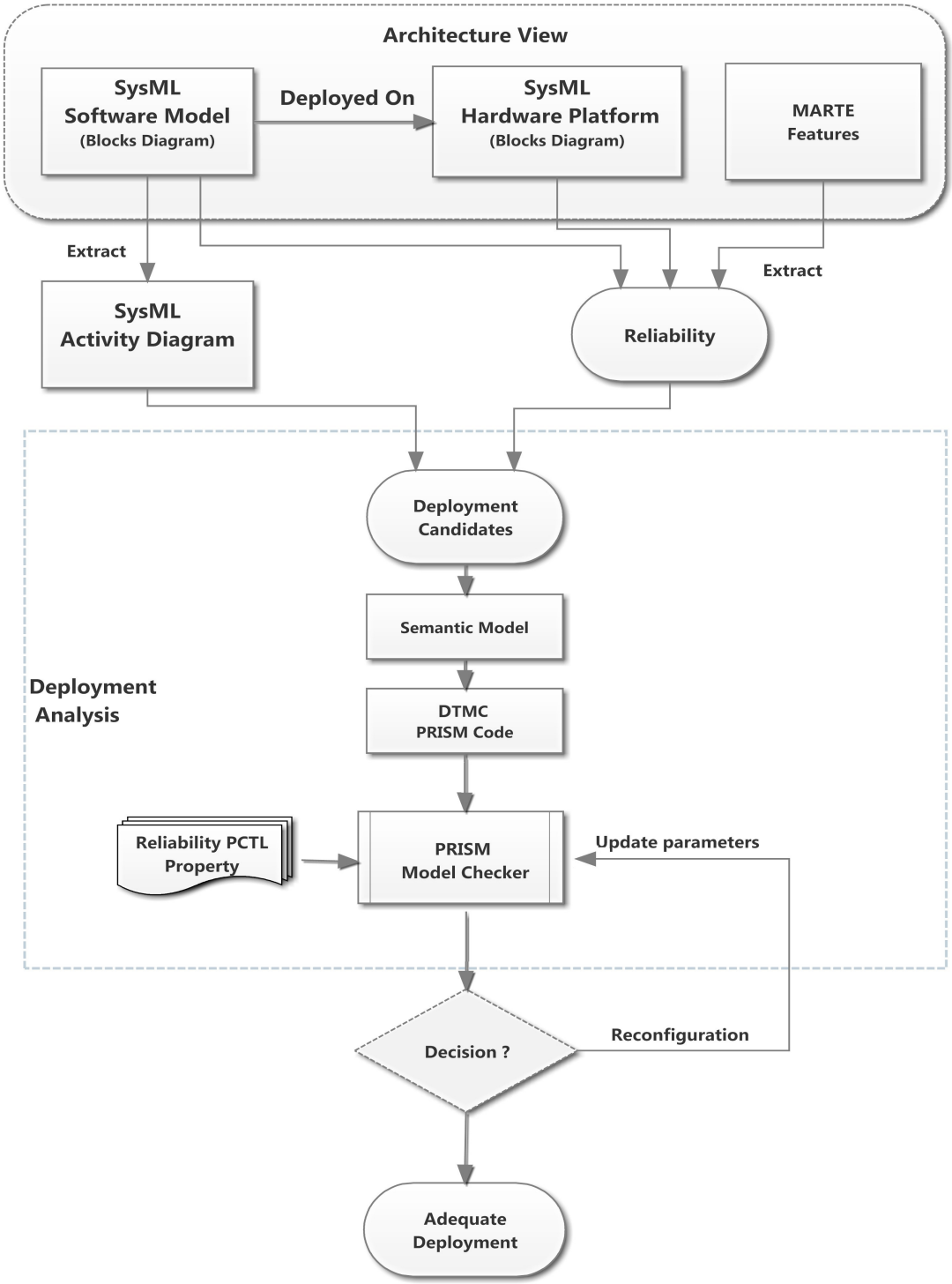


Figure 1: Reliability-driven Deployment Appoach

is proved in section10. Section 11 illustrates the application of our framework on a case study. In section 12 our main findings and threats to validity are discussed. Section 13 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 main function of ACC is keeping the speed constant at the setting selected by the driver (i.e. Absence of vehicle in front). In Fig.2, if the vehicle in front is traveling 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.

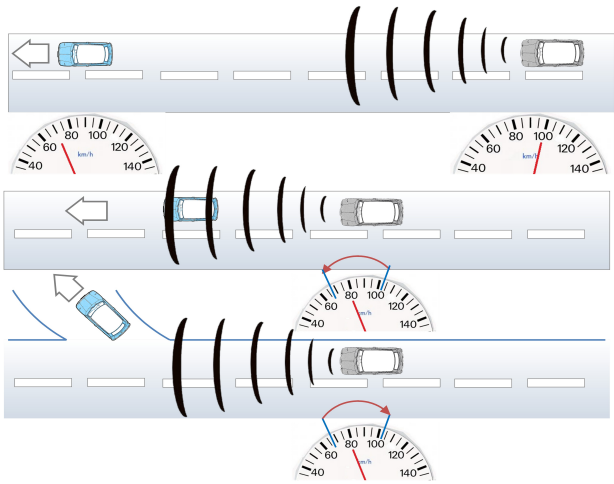


Figure 2: Adaptive Cruise Control System

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 11).

3. Solving the deployment problem

According to (Arcangeli et al., 2015), the deployment is a post-production process that consists in making software available for use and keep it up-to-date and operational. Nevertheless, current software are organized in as a set of components assembled as a system and operating together (Carlson et al., 2006). Component-based software must take into account dependencies between components themselves (i.e. Scheduling and communication) and between software components and its environment (e.g. Execution platform, smartphone). Thus, deployment plan must satisfy a set of constraints to build the correct

system. The reference papers on 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 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 on 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 software level. We assume that software failures due to programing imperfections are unlikely influence the deployment process, then we may abstract away from the hardware-independent software reliability. In addition, the software and hardware architecture is constant during the deployment.

Our approach assumes that processors are fail silent (Meedeniya et al., 2011), (Heiner and Thurner, 1998). This means that the processor detects the all errors and switch immediately to passive state which leads to an exception at software level. When failures occur during the execution of a software blocks, it impacts the reliability of our system. With a fixed and deterministic scheduling strategy, any failure happens on 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 SysML specification. For instance, the processor capacity corresponds to the << *HwProcessor* >> featured with MIPS as MARTE annotation (see Section 5).

3.1. Hardware Architecture

The set of execution components represented by processor are denoted by U =$\{ u_1, u_2, ..., u_m \}$, 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 =$\{ c_1, c_2, ..., c_n \}$, 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| d:C → U } denotes the set of all functions assigning software components to hardware processors, and we write a single deployment alternative d_j =$\{ (c_1, u_{j1}), (c_2, u_{j2}), ..., (c_n, u_{jm}) \} \subseteq D$ as d_j =$\{ u_{j1}, u_{j2}, ..., u_{jm} \}$.

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 failures of execution elements (Processors). The reliability of single element i (Meedeniya et al., 2011) can be modeled with exponential distribution as:

$$R_i = e^{-\lambda.T} \tag{1}$$

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 time taken for the execution is defined as function of the software-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 is arrived. Based on the software workload and processing speed of the processors, the scheduling length can be calculated as follows:

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

The reliability of individual software block are computed as :

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

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 a parameters for PRISM model checker. We note that the probabilities on activity diagram are unset (i.e. variables). When the reliability of individual software blocks are known, the full reliability is computed using 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 :

$$\mathrm{R} = Max\{R_{d_0}, ..., R_{d_m}\}, \quad m \in \mathbb{N} \tag{4}$$

4. Related work

In this section, we depict the recent works related to the deployment, partitioning and allocation optimization 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 be 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 properties 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 large space of deployment candidates

Herrera et al. (2014) present the COMPLEX approach for hardware/software partitioning and allocation to 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 algorithm or strategy applied.

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. Behavior) as input to the UPPAAL (Behrmann et al., 2004) model checking tool for temporal property satisfaction. For design-space exploration. The mapped model is processed by H-SPEX DSE tool (Oliveira et al., 2007). The core of the tool is based on Ant Colony Optimization. However, the authors do not clarify the optimization strategy that is hidden from the user.

Brosse et al. (2012) present ENOSYS design flow for the Modeling and Synthesis of Embedded Systems. The approach consists of four consecutive steps respectively named Modeling, Synthesis, Source Code Optimization and Design Space Exploration. The modeling step is based on UML Composite diagrams (i.e. Components representations) with the core behavior using state machine or activity diagrams. The composite diagram is enriched with 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 early 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 multi-core CPU platform (i.e. C/C++) which forms the programmable domain of the system implementation. The disadvantages of the approach is that the process of exploration occurs at code level instead on design level which is time consuming.

Martínez-Álvarez et al. (2013) propose a multi-objectives optimization tool with the Software Hardening Environment for the fault tolerant embedded systems design. The tool automatically transform an original 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 apply Genetic Algorithm (GA) to fulfill 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 behavioral annex. The specification is based on AADL language(Architecture Analysis and design Language) (Feiler, 2010) and Simulink (Simulink Link) for functional behavior. 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 results 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-objectives optimization taking into account mainly the execution time, the memory requirement, the power consumption. The process applied the genetic algorithm on control data flow graph (CDFG). However, the specification is based on 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 behavior. 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 really interesting since the decision is made at synthesis level.

Jia et al. (2014) propose 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 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, a 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.

4.1. Comparison

As a summary, in Table.1 we compare our framework to the existing works by taking consideration six criteria: specification language, addressed problem, applied algorithm, formalization, soundness and automation. The Specification criteria shows if the design is based on 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 confirms if the approach presents a semantics and formalizes the studied diagrams. Soundness feature shows if the mapping of the studied approach is proved. Automation criteria checks if the presented approach is automatized. From the comparison, we observe that only few works formalize the specification diagrams, including the model checking as decision tool for deployment case. Our work has for objective to support: component-based specification using SysML/MARTE and formalizing the SysML activity diagram. In addition, we implement the tool that allows the automatic deployment decision.

5. Background on SysML diagrams

The system specification described in this paper is characterized by following a component-oriented approach (Szyperski, 2002) for the deployment of software components. In component-based approach, the system is built by the composition of multiples blocks (i.e. component) interacting with each others through a well defined interfaces. In addition, the specification of behavior is based on activity diagrams.

5.1. SysML internal blocks diagram

The **block** (Friedenthal et al., 2008) is the fundamental modular unit for describing system structure in SysML. 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 more information about the software and hardware blocks to apply analysis such as the processor speed. To provide capabilities that are missing in SysML, 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 Fig.3) and some attributes can be added like *mips* that characterize the instruction-processing capacity. To annotate each processor and bus components with probability of success and failure of data transmission, we use "GaStep" annotation from Generic Quantitative Analysis Modeling package (GQAM) with attribute *prob*. Fig.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.2, 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 behavior. There are three main behavioral formalisms in SysML: activities, state machines and interactions diagrams.

5.2. SysML activity diagram

SysML Activity diagram (OMG, 2012) is a graph-based diagram where activity nodes are connected by activity edges. Fig 4 shows the set of interesting artifacts used for verification in our framework. The artifacts consist of activity nodes, activity edges, activity control and activity partitions. Activity nodes have three type: Activity invocation, objects and control nodes. Activity invocation includes action, call behavior, send/receive signals (objects). The Activity control includes: initial node, flow final, activity final, fork, merge and join node. Activity edge includes: control flow and object flow. Object flow connects the output pin of one action to the input pin of next action to enable the passage of data. Control flow provides additional constraints on when and in which order the action within an activity will be executed. A token on an incoming control flow enables the execution of an action and offers a control token on outgoing control flow when action completes its execution. Control nodes such as join, fork, merge, decision, initial and final are used to control the routing of control token over edges and specify the sequence of actions (concurrency,

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)		Partitioning				√
Our	SysML/MARTE	Deployment	Model Checking	√	√	√

Table 1: Comparison with the existing approaches for design-space exploration

Deployment features	MARTE annotations
Processor speed	*mips*
Processor failure	*Prob*
Component workload	*hostDemandOps*

Table 2: Software/Hardware Deployment features with their corresponding MARTE annotations

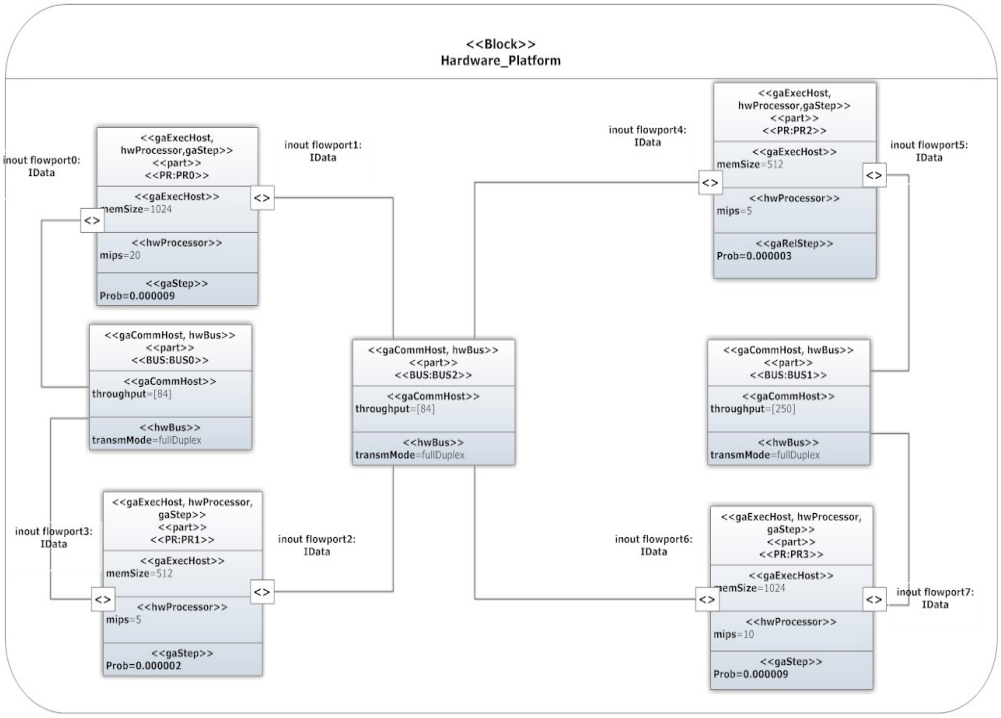


Figure 3: Example of Hardware platform

synchronization). A set of activity diagram artifacts can be grouped into an activity partition (also known as a *swimlane*). A typical case is when an activity partition represents a block or a part and indicates that any behaviors invoked in that partition are the responsibility of the block or the part. In our paper, we link each sub set of actions to each block of the IBD. Fig.5 presents an example of the electric motor inverter (Cherfi et al., 2014) where different activity artifacts 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.

6. Probabilistic model checking

Probabilistic model checking is based on the construction and the analysis of a probabilistic model of 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 as set of states S and a transition probability matrix **P**: $S \times S \rightarrow [0,1]$. The formal definition is:

Definition 1. Discrete-Time Markov chains (DTMC). DTMC is a tuple M = <$\overline{s}$, S, P, L >, where:

- $\overline{s}$ is an initial state, such that $\overline{s} \in$ S ,

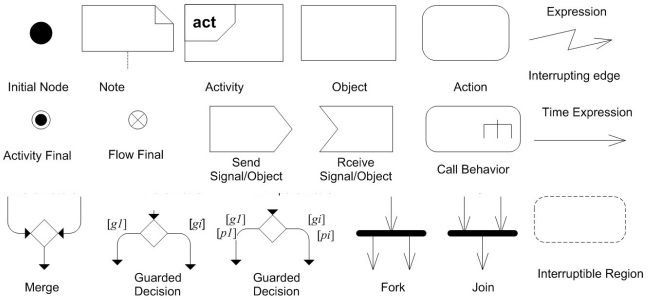


Figure 4: A sub set of SysML activity diagram artifacts

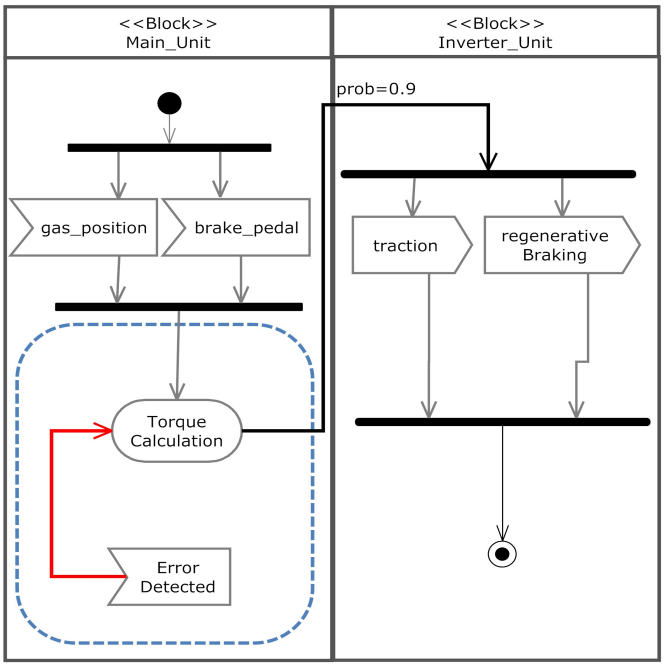


Figure 5: An example of activity allocated to the blocks

- S is a finite set of states,
- P: S $\times S \rightarrow [0, 1]$ is a rate function assigning for each transition from state s *to* s', a positive real value $P(s, s') \in [0, 1]$ where $\sum_{s' \in S} P(s, s') = 1$ for all $s' \in S$.
- L : S $\rightarrow 2^{AP}$ is a labeling function which assigns to each state $s \in S$ the set L(s) of atomic propositions that are valid in s.

Lets 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 certain 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 Fig.5. In order to prevent an 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 Markov model of such a system as shown in Fig.6, is built with two kind of states (S: Operational, S_f: Fault detected) representing the block status. P and $1 - P$ are the reliability of correct and failure signal transmission, respectively. This system is described using PRISM modeling language as shown in Listing.1.

```
 const double P;
module satellite
 sc1 :bool init true; // initial node
 ................;
 sf : bool init false;  // final node
[] sc1 -> (Sfj1'=true)&(sc1'=false);
 ................
[] sf    -> (sc1'=true);
[] sc2 -> (sc3'=true) & (sc2'=false);

[] S -> P:(Ssuccess'=true)&(S'=false)
               +1-P:(Sf'=true)&(S'=false);

[] Sf ->(S'=true)&(Sf'=false) ;
[] Ssuccess ->(Sfj2'=true)&(Ssuccess'=false) ;
 ................
endmodule
```

Listing 1: Satellite PRISM code

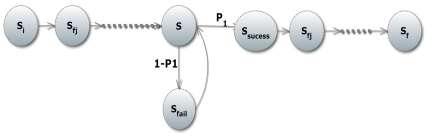


Figure 6: A simple Markov chain

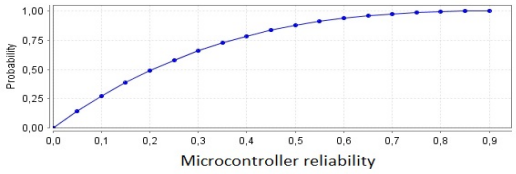


Figure 7: Total system reliability

The model checking approach for reliability analysis requires both a description of system in probabilistic model and properties for performance evaluation using Probabilistic Computation Tree Logic (PCTL) (Kwiatkowska et al., 2002). For the Markov chain in Fig. 6, if P is parameterizable taking values from 0 to 0.95, then the results could be plotted as graph in Fig.7. The reliability is obtained by checking the model against the property : "the probability that the system runs correctly within T time steps" and expressed using PCTL as $P =?[true \cup^{\leq T} S_{success}], T = 100$; T is non-negative value representing the upper time bound. The structure of our temporal logic is expressed by the following BNF grammar:

6.1. Property Specification

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

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

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

Where "ap" is an atomic proposition, P is a probabilistic operator. Operator $P_{\bowtie p}[\psi]$ means that the probability of 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 $\vDash \psi$ to denote that s satisfies the state formula φ, while $\pi \vDash \psi$ (sequence of states) denotes that π satisfies the path formula ψ.

- $s \vDash$ true is always satisfied,

- $s \vDash ap \Longleftrightarrow ap \in L(s)$ and L is a labeling function,
- $s \vDash \varphi_1 \wedge \varphi_2 \Longleftrightarrow s \vDash \varphi_1 \wedge s \vDash \varphi_2$,
- $s \vDash \neg\varphi \Longleftrightarrow s \nvDash \varphi$,
- $s \vDash P_{\bowtie p}[\psi] \Longleftrightarrow P\{\pi | \pi \vDash \psi\}$ such that the probability of the path $\pi = s_0 ... s_n$ is given by $\mathrm{P}(\pi) = \prod_{i=0}^{n-1} p(s_i, s_{i+1})$,
- $s \vDash P_{\bowtie p}[\varphi_1 \cup^k \varphi_2] \Longleftrightarrow \exists i \leq k : \forall j < i, \pi(j) \vDash \varphi_1 \wedge \pi(i) \vDash \varphi_2$
- $s \vDash P_{\bowtie p}[\varphi_1 \cup \varphi_2] \Longleftrightarrow \exists k \geq 0 : \pi \vDash \varphi_1 \cup^k \varphi_2$

7. 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) and (Baouya et al., 2015), we formalize the deployment by developing its calculus : *Deployment Activity Calculus* (DAC). In Table.3, 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 probability $1 - p$. In this calculus we distinguish between two syntactic concepts: Marked and Unmarked terms. A marked terms are active component with tokens. The unmarked terms correspond to the static structure of the service workflow. We use $Ex(p, \mathcal{N}_1, \mathcal{N}_2)$ to express an exception that could happen with probability $1 - p$. To describe better our approach, in Table.3 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) to formally describe how the computation steps of DAC atomic terms take place. We describe the operational semantic rules of the DAC calculus in Table.4. The semantics of parameterizable activity diagram can be described in terms of DTMC as stipulated by Definition 2.

Definition 2 (DAC-DTMC)**.** A discrete-time Markov Chain of DAC term $\mathscr{A}$ is the tuple $M_{\mathscr{A}} =< \overline{s}, S, P, L >$, where:

- $\overline{s}$ is an initial state, such that $L(\overline{s}) = \left\{ l : \overline{i} \rightarrowtail \mathcal{N} \right\}$,
- S is a finite set of states reachable from $\overline{s}$, such that, $S = \left\{ s_{i:0 \leq i \leq n} | L(s_i) = \left\{ \overline{\mathcal{N}} \right\} \right\}$,
- P: $\mathrm{S} \times \mathrm{S}' \rightarrow [0, 1]$ 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'$=\left\{ s_i : 0 \leq i \leq n : s \rightarrow_{p_i} s_i \right\}$, 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: $[[\mathcal{L}]] : \mathcal{L} \rightarrow \{true, false\}$.

8. The PRISM model checker

In this paper, we use PRISM probabilistic model checker (Kwiatkowska et al., 2011). It supports the analysis type of probabilistic models: Discrete-time Markov chains (DTMC) (Kwiatkowska et al., 2012), Continuous-time Markov chains (CTMC) (Kwiatkowska et al., 2007), Markov decision process (MDP) (Forejt et al., 2011). Since Discrete-timed Markov chains (DTMCs) is considered as appropriate semantic model for modeling reliability (Franco et al., 2016) in SysML Activity Diagram, we briefly review the concepts of DTMC.

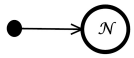
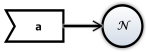
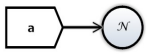
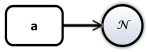
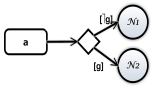
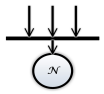
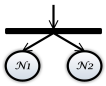
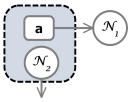
Artifacts	DAC terms	Description
	$l : i \rightarrowtail \mathcal{N}$	Initial node is activated when a diagram is invoked.
	$l : \odot$	Activity final node stops the diagram' execution.
	$l : \otimes$	Flow final node kills its related path' execution.
	$l : a?v \rightarrowtail \mathcal{N}$	Receive node is used to receive a signal/object.
	$l : a!v \rightarrowtail \mathcal{N}$	Send node is used to send a signal/object.
	$l : a \rightarrowtail \mathcal{N}$	Action node defines an atomic action.
	$l : D(g, \mathcal{N}_1, \mathcal{N}_2)$	Decision node with guarded edges $\{g, \neg g\}$.
	$l : M(x) \rightarrowtail \mathcal{N}$	Merge node specifies the continuation and $x = \{x_1, x_2\}$ is a set of input flows.
	$l : J(x) \rightarrowtail \mathcal{N}$	Join node presents the synchronization where $x = \{x_1, x_2\}$ is a set of input pins.
	$l : F(\mathcal{N}_1, \mathcal{N}_2)$	Fork node models the concurrency that begins multiple parallel control threads.
	$l : Ex(p_1, \mathcal{N}_1, \mathcal{N}_2)$	Interruptible region models the interruption when errors occurs with $1 - p_1$

Table 3: DAC Terms of SysML Activity Diagram Artifacts.

A PRISM program ($\mathscr{P}$) is a composition of one or more modules. The state of each module is defined by a set of fined ranged variables V. The global state of the system is determined by evaluating the values of those module variables. The behavior of each module is described using commands which include a **guard** and one or more updates of the form:

$$[]\text{guard} \rightarrow p_1 : u_1 .. + p_n : u_n$$

The **guard** is a predicate over the variables of all the modules in the model and p is a probability value. Each update $\boldsymbol{u}$ describes a transition which the module can make if the guard is true as assignments form $(V'_1 = val_1)\&..\&(V'_k = val_k)$ where $V_i \in [min...max]$ such that $min, max \in \mathbb{N}$ and min < max, and val_j are values evaluated via expressions denoted by "eval" eval: $\mathrm{V} \rightarrow \mathbb{N} \cup \{True, False\}$.

Definition 3 stipulates the formal definition of PRISM probabilistic timed automata denoted by $\mathrm{M}_{\mathscr{P}}$. The states of $\mathrm{M}_{\mathscr{P}}$ take the form $\langle V_1, ..., V_k, eval \rangle$. The stepwise behavior of $\mathrm{M}_{\mathscr{P}}$ 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 $\mathscr{P}$ is the tuple $\mathrm{M}_{\mathscr{P}} =< \mathrm{s_i}, S, \mathrm{P}, \mathrm{L} >$, where:

INT-1 $l : i \rightarrowtail \mathcal{N} \xrightarrow{l}_1 l : \bar{i} \rightarrowtail \mathcal{N}$ This axiom introduces the execution of $\mathcal{A}$ by putting a token on i .

ACT-1 $\forall n > 0, m \geq 0\ l : \overline{\overline{a^m} \rightarrowtail \mathcal{N}}^n \xrightarrow{l}_1 l : \overline{\overline{a^{m+1}} \rightarrowtail \mathcal{N}}^{n-1}$ This axiom propagates the token from the global marked term to a.

ACT-4 $\forall n > 0, m \geq 0\ l : \overline{\overline{a^m} \rightarrowtail \mathcal{N}}^n \xrightarrow{l}_1 l : \overline{\overline{a^{m+1}} \rightarrowtail \mathcal{N}}^{n-1}$ This axiom propagates the token from the global marked term to a with probability p (i.e. failure rate).

DEC-1 $\forall n > 0\ l : \overline{D(g, \mathcal{N}_1, \mathcal{N}_2)}^n \xrightarrow{g} l : \overline{D(g, \overline{\mathcal{N}_1}, \mathcal{N}_2)}^{n-1}$ The axiom DEC-1 describes a non-probabilistic decision where a token flows through the edge satisfying its guard.

EXP-2 $l : \overline{Ex(p, \mathcal{N}_1, \mathcal{N}_2)}^n \xrightarrow{l}_{1-p} l : \overline{Ex(p, \mathcal{N}_1 \overline{\mathcal{N}_2})}^{n-1}$ Node $\mathcal{N}_2$ is triggered after an exception with probability $1 - p$.

MRG-2 $l : \overline{M(\bar{x}, \bar{y}) \rightarrowtail \mathcal{N}}^n \xrightarrow{l} l : \overline{M(x, \bar{y}) \rightarrowtail \overline{\mathcal{N}}}^n$ MRG-2 is a transition with rate value equal 1 to present a token flowing from merge labeled by l to the sub-term $\mathcal{N}$.

JOIN-2 $l : \overline{J(\bar{x}, \bar{y}) \rightarrowtail \mathcal{N}}^n \xrightarrow{l} l : \overline{J(x, y) \rightarrowtail \overline{\mathcal{N}}}^n$ JOIN-2 represents a transition with rate value equal 1 to move a token in join to the sub-term $\mathcal{N}$

AFIN $\mathcal{A}[l : \overline{\odot}] \xrightarrow{l} |\mathcal{A}|$ This axiom states that if the sub-term $l : \odot$ is reached then no action is taken later by destroying all tokens.

INT-2 $l : \bar{i} \rightarrowtail \mathcal{N} \xrightarrow{l}_1 l : i \rightarrowtail \overline{\mathcal{N}}$ This axiom propagates the token in the marked term i into its outgoing $\mathcal{N}$.

ACT-2 $l : \overline{\overline{a}^{m+1} \rightarrowtail \mathcal{N}}^n \xrightarrow{l}_1 l : \overline{\overline{a}^m \rightarrowtail \overline{\mathcal{N}}}^n$ This axiom propagates the token from the marked term a to $\mathcal{N}$.

FRK-1 $\forall n > 0\ l : \overline{F(\mathcal{N}_1, \mathcal{N}_2)}^n \xrightarrow{l}_1 l : \overline{F(\overline{\mathcal{N}_1}, \overline{\mathcal{N}_2})}^{n-1}$ The FRK-1 axiom shows the multiplicity of the arriving tokens according to the outgoing sub-terms

EXP-1 $l : \overline{Ex(\mathcal{N}_1, \mathcal{N}_2)}^n \xrightarrow{l} l : \overline{Ex(\mathcal{N}_1, \overline{\mathcal{N}_2})}^{n-1}$ Node $\mathcal{N}_2$ is triggered after an exception

MRG-1 $l : \overline{\overline{\mathcal{N}} \rightarrowtail l' : M(x, y)}^n \xrightarrow{l} l : \overline{\mathcal{N} \rightarrowtail l' : M(\bar{x}, y)}^n$ MRG-1 is a transition with rate value equal 1 to put a token coming from the sub-term $\mathcal{N}$ on merge labeled by l.

JOIN-1 $l : \overline{\overline{\mathcal{N}} \rightarrowtail l' : J(x, y)}^n \xrightarrow{l} l : \overline{\mathcal{N} \rightarrowtail l' : J(\bar{x}, y)}^n$ JOIN-1 represents a transition with rate value equal 1 to activate the pin x in a join labeled by l

SND $l : \overline{a!v}^n \rightarrowtail \mathcal{N} \xrightarrow{l} l : \overline{a!v}^{n-1} \rightarrowtail \overline{\mathcal{N}}$ SND describes the evolution of the token after sending an object v.

Table 4: Operational semantic rules of the DAC calculus

- s_i is an initial state,
- S is a finite set of states reachable from s_i, such that, $S = \{s_{i:0 \leq i \leq n} | L(s_i) \in \{AP\}\}$,
- P: S × S→ [0, 1] is a probabilistic function where $\sum_{s' \in S}$ P(s, s′) = 1 for all $s \in$ S,
- L : S → 2^{AP} is a labeling function that assigns for each state a set of valuated propositions.

9. Translation rules

This section describes the transformation of SysML Activity diagrams $\mathscr{A}$ into a DTMC written in PRISM input language. We apply an Algorithm that takes $\mathscr{A}$ as input and returns its PRISM code. For further details about this algorithm, consult (Ouchani et al., 2014b). The diagram $\mathscr{A}$ is visited using a depth-first search procedure (DFS) without backing to the visited node and the algorithm's output produces PRISM modules.

The function Γ presented in Listing.2 produces the appropriate 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 depends on how the term is activated. The flag related to the term is its label l that is initialized to false except for the initial node it is true which conforms to the premise of the DAC rule "INIT-1". The updates of the command deactivate the propositions of the term, activate that ones related to its successors. For a term $n \in \mathscr{A}$, we define a useful function: $L(n)$ that returns the label of a term n.

For example, the exception "$l : Ex(p, N_1, N_2)$" (line 29) produces two commands (line 31-32). The first one in (line 31) produces a PRISM command that refers to the correct state transition. Similarly, the second 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 "$l : D(g, N_1, N_2)$" produce 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 $\mathscr{A}$ is bounded by two PRISM primitives: the module head "module $\mathscr{A}$", and the module termination "endmodule".

$\mathbf{\Gamma} : \mathscr{A} \longrightarrow \mathscr{P}$

$\Gamma(\mathscr{A}) = \forall n \in \mathscr{A}\ ,\ L(n = (i)) = true,\ L(n \neq (i)) = false$

$l : i \rightarrowtail \mathcal{N} \Longrightarrow$

in

$\{[l]l \longrightarrow (l' = false)\&(L(\mathcal{N})' = true);\} \cup \Gamma(L(\mathcal{N}))$

end

$l : M(x, y) \rightarrowtail \mathcal{N} \Longrightarrow$

in

$\{[l_x]l_x \longrightarrow (l'_x = false)\&(L(\mathcal{N})' = true);\} \cup \left\{[l_y]l_y \longrightarrow (l'_y = false)\&(L(\mathcal{N})' = true);\right\} \cup \Gamma(L(\mathcal{N}))$

end

$l : J(x, y) \rightarrowtail \mathcal{N} \Longrightarrow$

in

$\left\{[l]l_x \wedge l_y \longrightarrow (l'_y = false)\&(l'_x = false)\&(L(\mathcal{N})' = true);\right\} \cup \Gamma(L(\mathcal{N}))$

end

$l : F(\mathcal{N}_1, \mathcal{N}_2) \Longrightarrow$

in

$\{[l]l \longrightarrow (l' = false)\&(L(\mathcal{N}_1)' = true)\&(L(\mathcal{N}_2)' = true);\} \cup \Gamma(L(\mathcal{N}_1)) \cup \Gamma(L(\mathcal{N}_2))$

end

$l : D(g, \mathcal{N}_1, \mathcal{N}_2) \Longrightarrow$

in

$\{[l]l \wedge g \rightarrow (l' = false)\&(L(\mathcal{N}_1)' = true)\} \cup \{[l]l \wedge \neg g \rightarrow (l' = false)\&(L(\mathcal{N}_2)' = true)\}$

end

$l : Ex(p, \mathcal{N}_1, \mathcal{N}_2) \Longrightarrow$

in

$\{[l]l \rightarrow p : (l' = false)\&(L(\mathcal{N}_1)' = true);\}$

$\{[l_{error}]l \rightarrow (1 - p) : (l' = false)\&(L(\mathcal{N}_2)' = true)\} \cup \Gamma(L(\mathcal{N}_2)) \cup \Gamma(L(\mathcal{N}_1))$

end

$l : \odot \Longrightarrow$

in

$[l] \rightarrow \&_{l \in \mathcal{L}}(l' = false);$

end

$l : \otimes \Longrightarrow$

in

$[l] \rightarrow (l' = false);$

end

$l : a \rightarrowtail \mathcal{N} \Longrightarrow$

in

$\{[l_a]l \longrightarrow (l'_a = false)\&(L(\mathcal{N})' = true);\} \cup \Gamma(L(\mathcal{N}))$

end

Listing 2: Generating PRISM Commands Function

10. The transformation soundness

Our aim is to prove the soundness of the transformation algorithm Γ by showing that the proposed algorithm preserves the satisfiability of DTMC properties. Let $\mathscr{A}$ be a DAC term and $M_{\mathscr{A}}$ is its corresponding DTMC constructed by the DAC operational semantics denoted by $\mathscr{S}$ such that $\mathscr{S}(\mathscr{A}) = M_{\mathscr{A}}$. For the program $\mathscr{P}$ resulting after transformation rules, Let $M_{\mathscr{P}}$ its corresponding DTMC constructed by PRISM operational semantics denoted $\mathscr{S}'$ such that $\mathscr{S}'(\mathscr{P}) = M_{\mathscr{P}}$. As illustrated in Fig.8, proving the soundness of Γ algorithm is to find the adequate relation $\mathscr{R}$ between $M_{\mathscr{A}}$ and $M_{\mathscr{P}}$. To define the relation $M_{\mathscr{A}}\mathscr{R}M_{\mathscr{P}}$, we have to establish a step-by-step correspondence between $M_{\mathscr{A}}$ and $M_{\mathscr{P}}$. First, we introduce the notion of the strong bi-simulation relation for DTMC (Baier et al., 2005) in definition 4.

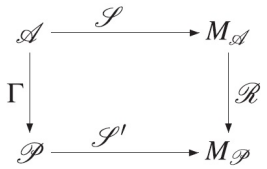


Figure 8: The Transformation Soundness.

Definition 4 (Strong bi-simulation Relation)**.** A strong probabilistic bi-simulation between two DTMCs M_1 and M_2 is a binary relation $\mathscr{R}$ iff whenever $s_1\mathscr{R}s_2$:

- Each initial state of M_1 is related to at least one initial state M_2.
- For each pair of states $s_1\mathscr{R}s_2$ 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 $p_1 = p_2$.

For our proof, we stipulate herein the mapping relation $\mathscr{R}$ denoted by $M_{\mathscr{A}}\mathscr{R}M_{\mathscr{P}}$ between a DAC term $\mathscr{A}$ and its corresponding PRISM term $\mathscr{P}$.

Definition 5 (Mapping relation)**.** The relation $M_{\mathscr{A}}\mathscr{R}M_{\mathscr{P}}$ between a DAC term $\mathscr{A}$ and a PRISM term $\mathscr{P}$ such that $\Gamma(\mathscr{A}) = \mathscr{P}$ is a strong bi-simulation relation.

Finally, proving that Γ is sound means showing the existence of a strong bi-simulation between $M_{\mathscr{A}}$ and $M_{\mathscr{P}}$.

Lemma (Soundness)**.** The mapping algorithm Γ is sound, *i.e.* $M_{\mathscr{A}} \sim M_{\mathscr{P}}$.

Proof. We prove $M_{\mathscr{A}} \sim M_{\mathscr{P}}$ by following a structural induction on DAC terms and their related PRISM terms. For that, let $s_1, s_1' \in \mathscr{S}_A$ and $s_2, s_2' \in \mathscr{S}_P$. We distinguish the following cases where L(s) takes different values:

1. $L(s_1) = l : \overline{x} \rightarrowtail \mathcal{N}$ such as $x = \{i, a\} \Longrightarrow \exists s_1 \rightarrow s_1'$, L($s_1$')$= l : x \rightarrowtail \overline{\mathcal{N}}$. For L($s_2$) = Γ(L($s_1$)), we have L($s_2$)=$\langle L(x), \neg L(\mathcal{N})\rangle$ then $\exists s_2 \rightarrow s_2'$ where $L(s_2') = \langle \neg L(x), L(\mathcal{N})\rangle$.
2. $L(s_1) = l : \overline{x} \rightarrowtail \mathcal{N}$ such as $x = \{a!v, a?v\} \Longrightarrow \exists s_1 \rightarrow s_1'$, L($s_1$')$= l : x \rightarrowtail \overline{\mathcal{N}}$. For L($s_2$) = Γ(L($s_1$)), we have L($s_2$)=$\langle L(x), \neg L(\mathcal{N})\rangle$ then $\exists s_2 \rightarrow s_2'$ where $L(s_2') = \langle \neg L(x), L(\mathcal{N})\rangle$.
3. $L(s_1) = l : \overline{Ex(p, \mathcal{N}_1, \mathcal{N}_2)}$ then $\exists s_1 \rightarrow_{p_1} s_1'$, L($s_1$')$= l : Ex(p, \mathcal{N}_1, \overline{\mathcal{N}_2})$. For L($s_2$) = Γ(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 $L(s_2') = \langle \neg l, l_{\mathcal{N}_1}, \neg l_{\mathcal{N}_2}\rangle$.
4. $L(s_1) = l : \overline{D(g_1, \mathcal{N}_1, \mathcal{N}_2)}^{n}$ then $\exists s_1 \xrightarrow{g_1}_1 s_1'$, L($s_1$')$= l : \overline{D(g_1, \overline{\mathcal{N}_1}, \mathcal{N}_2)}^{n-1}$. For L($s_2$) = Γ(L($s_1$)), we have L($s_2$)=$\langle l, \neg l_{\mathcal{N}_1}, \neg l_{\mathcal{N}_2}\rangle$ then $\exists s_2 \xrightarrow{g_1}_1 s_2'$ where $L(s_2') = \langle \neg l, l_{\mathcal{N}_1}, \neg l_{\mathcal{N}_2}\rangle$.

5. $L(s_1) = l : \overline{\odot}$ 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 $\forall l_i \in \mathcal{L} : L(s'_2) = \langle \neg l_i \rangle$.
6. $L(s_1) = l : \overline{\otimes}$ 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 $L(s'_2) = \langle \neg l \rangle$.

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 $s_1 \sim s_2$. In addition, the unique initial state of $M_{\mathscr{A}}$ is always corresponding to the unique initial state in $M_{\mathscr{P}}$. By studying all DAC terms, we find that $M_{\mathscr{A}} \sim M_{\mathscr{P}}$, which confirms that Lemma holds.

□

In the following, we show that the mapping relation preserves the satisfiability of PCTL properties. This means, if a PCTL property is satisfied in the resulting model by a mapped function Γ then it is satisfied by the original one.

Proposition (PCTL preservation)**.** For two DTMCs $M_{\mathscr{A}}$ and $M_{\mathscr{P}}$ such that $\Gamma(\mathscr{A}) = \mathscr{P}$ where $M_{\mathscr{A}} \sim M_{\mathscr{P}}$. For a PCTL property ϕ, then: $(M_{\mathscr{A}} \vDash \phi) \Longleftrightarrow (M_{\mathscr{P}} \vDash \phi)$.

Proof. The preservation of PCTL properties is proved by induction on the PCTL structure and its semantics. Since $M_{\mathscr{A}} \sim M_{\mathscr{P}}$ and by relying to the semantics of each PCTL operator $\zeta \in \{\cup, \cup^{\leq k}\}$, we find that $(M_{\mathscr{A}} \vDash \zeta) \Longleftrightarrow (M_{\mathscr{P}} \vDash \zeta)$ which means: $(M_{\mathscr{A}} \vDash \phi) \Longleftrightarrow (M_{\mathscr{P}} \vDash \phi)$ □

11. 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 TopCased [1]. 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 DTMC model in order to check the configuration against the PCTL property. Finally, the plug-in returns a message for the close-candidate that satisfy our reliability requirement. Our approach is illustrated on 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 system (Section 3). To assess the performance of model checking on automotive systems, 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 processors failures according to our assumptions in the beginning. When the failure occurs, an exception node is activated to restart the block behavior activity. In our specification the physical platform is depicted in Fig.3. Table.6 refers to software components parameters.

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 5- Fig.3 abstracted in Fig.11. Table.5 contains the parameters of processors (PR).

Anti-lock Brake System (ABS): used in modern cars to minimize hazards associated with skidding and loss of control due to locked wheel during braking. The software architecture of the ABS is depicted in Fig.10 and Fig.13(Associated Activity diagram) (components 0-7 in Table.6). The ABS Main unit is the major decision unit regarding the braking levels for individual wheels, while Load Compensator unit assists with computing adjustment factors from wheel load sensor inputs. Component 4-7 (Table.6) represent transceiver software components associated with each wheel, and communicate with sensors and brake actuators. *Brake Paddle* is the software component that reads from the *Paddle Sensor* and sends event to the emergency stop detection software module.

[1] http://www.topcased.org Toolkit in OPen source for Critical Applications & SystEms Development.

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

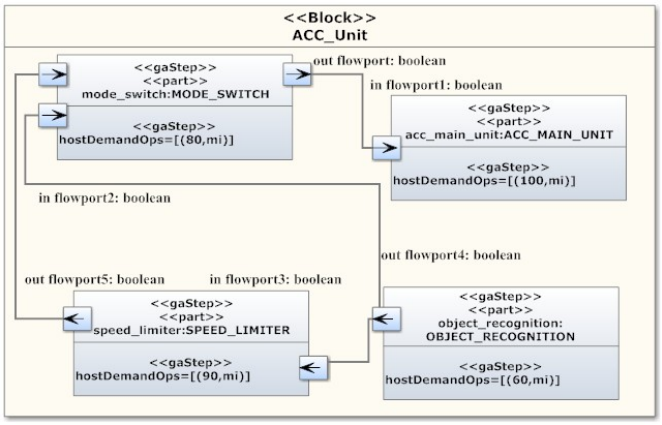


Figure 9: Adaptive Cruise Control

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

Adaptive Cruise Control (ACC): The aim of this component is to avoid crashes by reducing speed once the slower vehicle in front is detected. The main software components used by ACC are depicted in Fig.9 and Fig.12(Associated Activity diagram). Service initialization is possible at Object Recognition software that communicate with sensors. Speed Limit, Mode Switch and Brake Paddle contribute to triggering of the service. The captured data are processed by the ACC_MAIN_UNIT and Human Machine Interface to communicate with actuators.

11.1. Evaluation

Despite 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 checked model to get the minimum failure probability of each deployment candidate. During

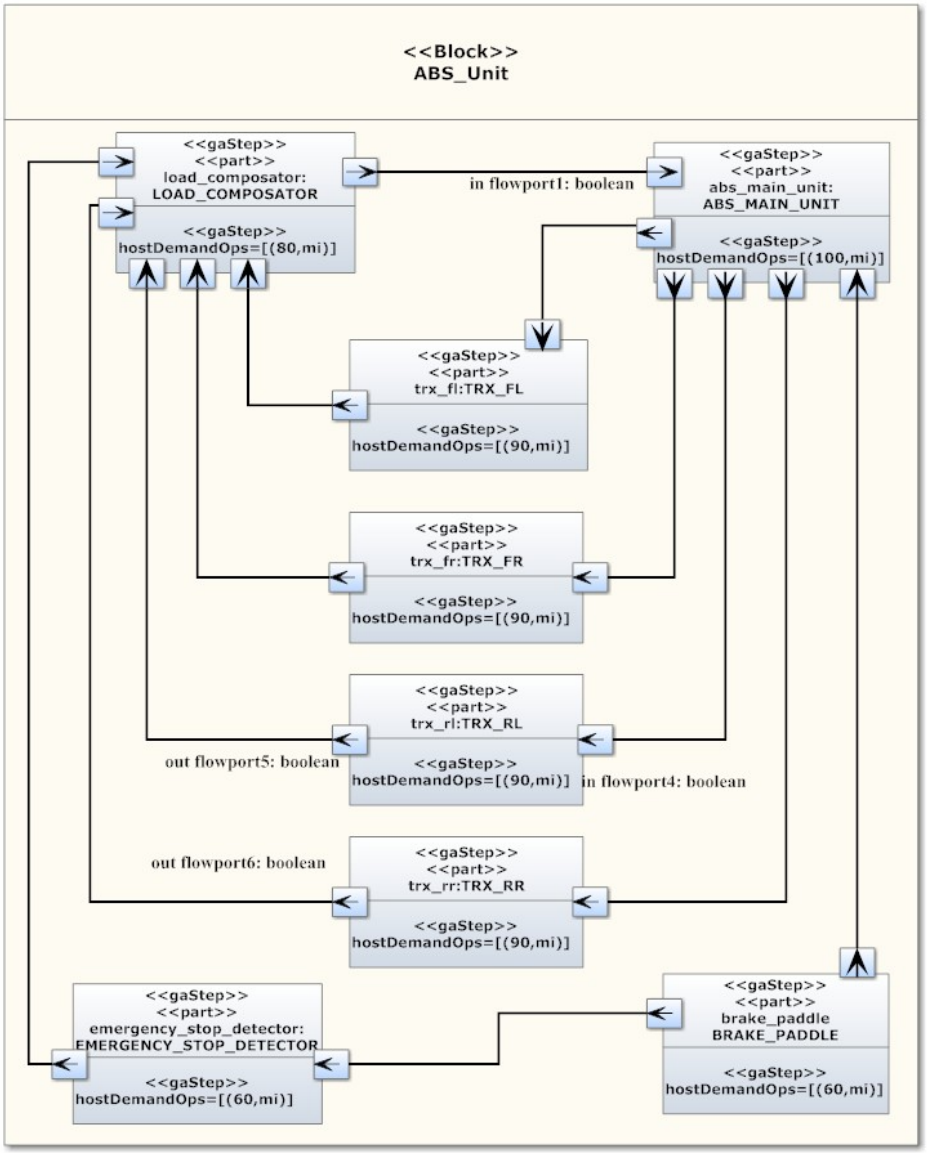


Figure 10: Anti-lock Brake System

this process, 3256 candidates was explored and performed on i7 CPU 2.67GHz with 8.0GB of RAM. The listing.3 shows the PRISM code that can be parametrized through the double constants values. The constant values are filled with our plug-in during the deployment-space-exploration (i.e. Model checking process).

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

$$P =?[true \cup^{\leq T} (\text{“}Proc\_Fail\text{”})], T = 200 \tag{5}$$

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* :

$$label\text{“}Proc\_Fail\text{”} = Block1\_Fail|Block2\_Fail|Block3\_Fail|Block4\_Fail \tag{6}$$

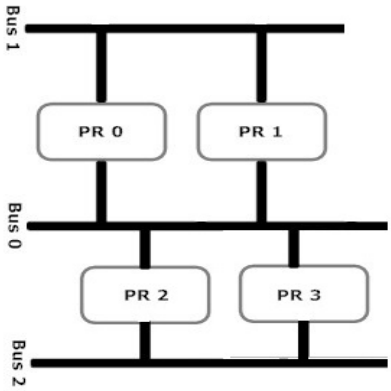


Figure 11: Hardware topology

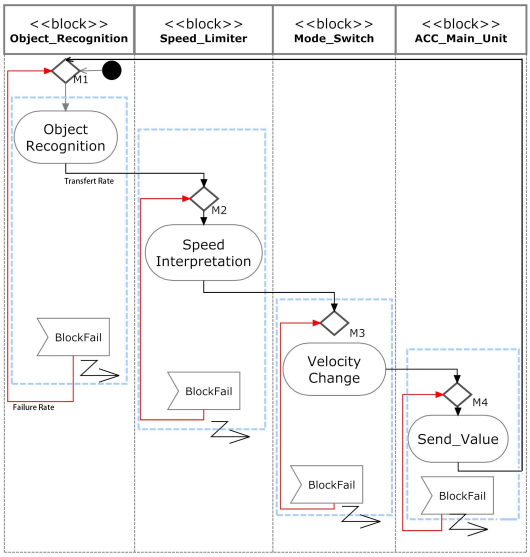


Figure 12: Activity diagram for Adaptive Cruise Control

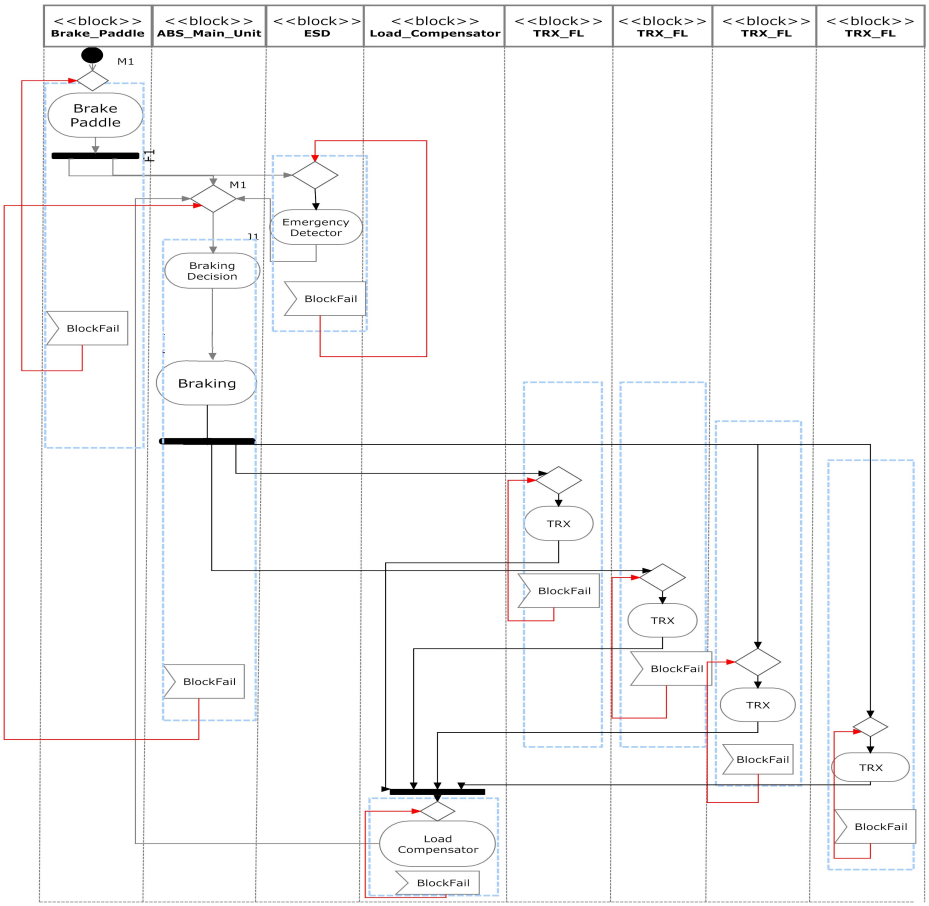


Figure 13: Activity diagram for Anti-lock Brake System

```
dtmc

const double P1;
const double P2;
const double P3;
const double P4;
module ACC
Initial : bool init true;
Object_Recognition: bool init false;
Speed_Interprestation:bool init false;
Velocity_Change : bool init false;
Send_Value : bool init false;
Final: bool init false;

Proc_Fail: bool init false;
.........

D1: bool init false;
D2: bool init false;
...........

M11: bool init false;
M12: bool init false;
M1: bool init false;
[Initial] Initial -> (Initial'=false)&(M11'=true);
[M11]M11 -> (M11'=false)&(M1'=true);
[M12]M12 -> (M12'=false)&(M1'=true);
[M1]M1 -> (M1'=false)&(Object_Recognition'=true);
[Object_Recognition] Object_Recognition -> 1.0:(D1'=true)&
(Object_Recognition'=false);

[D1] D1 -> P1:(M21'=true)& (D1'=false)+(1-P1):(Block1_Fail'=true)&(D1'=false);
[Block1_Fail] Block1_Fail-> (Block1_Fail'=false)& (M12'=true);
[Speed_Interpretation]Speed_Interpretation -> 1.0:(SpeedInterprestation'=false)
&(D2'=true);
[D2] D2 -> P2:(M31'=true)& (D2'=false)+(1-P2):(Block2_Fail'=true)&(D2'=false);
[Block2_Fail] Block2_Fail-> (Block2_Fail'=false)& (M22'=true);
[Velocity_Change] Velocity_Change -> 1.0:(Velocity_Change'=false)&
(D3'=true);
[D3] D3 -> P3:(M41'=true)& (D3'=false)+(1-P3):(Block3_Fail'=true)&(D3'=false);
[Block3_Fail] Block3_Fail-> (Block3_Fail'=false)& (M32'=true);
[Send_Value] Send_Value -> 1.0:(Send_Value'=false)&(D4'=true);
[D4] D4 -> P4:(Final'=true)& (D4'=false)+(1-P4):(Block3_Fail'=true)&(D4'=false);
[Block4_Fail] Block4_Fail-> (Block4_Fail'=false)& (M42'=true);
endmodule

label "ProcFail" = Block1_Fail|Block2_Fail|Block3_Fail|Block4_Fail;
```

Listing 3: ACC PRISM code fragment

The analysis results of reliability property obtained from PRISM are as follows: The result of the property in case of ACC is shown in Fig.14; the result of property in case of ABS is shown in Fig.15. 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.

11.2. System constraints and optimization

In our experiment, we have accepted a few properties to make our model simple and applicable to larger system. The software failure are 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).

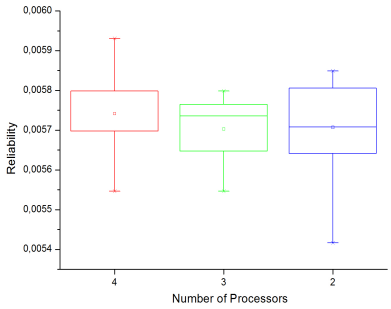


Figure 14: ACC Reliability

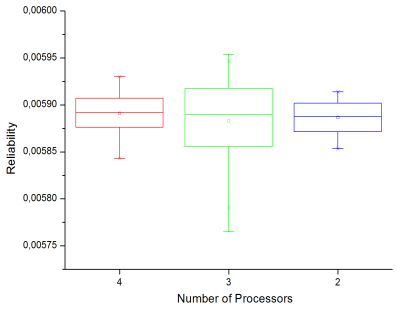
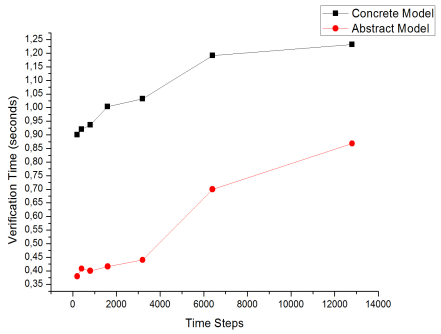
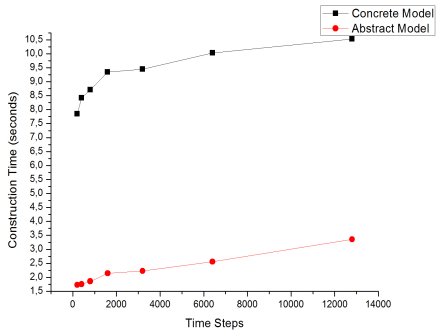


Figure 15: ABS Reliability



(a) Verification Time



(b) Construction Time

Figure 16: The abstraction effects on ABS diagram

Time Steps	Concrete Model		Abstract Model		Results
	Tc	Tv	Tc	Tv	
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 .

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*.

For this purpose we apply the algorithm and rules defined by (Ouchani et al., 2014a) to observe the abstraction effects on SysML Activity diagram in Fig.13 with the best deployment candidate parameters values. The abstraction approach produce a reduced activity diagram without the irrelevant actions (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 evaluate the verification cost, we measure the time required for model checking, denoted by Tv and time required to construct the model denoted by Tc. The results are depicted in Fig.16. The number of states and transitions for the concrete model are 16461 states and 93412 transitions, respectively and 3025 states, 15491 transitions for the abstracted model. we notice that the abstraction rules preserves the results while verification and construction time are optimized.

In addition, PRISM model checker implements an interesting 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 approximate probabilistic model checking using statistical methods. So, with the capability of PRISM engine, it is possible to analyze larger systems using our methodology.

12. Discussion and threats to validity

12.1. 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 behavioral and structural aspects of 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 a specific properties. In case of our methodology, we introduce MARTE profile as shown in Fig.3. We found that profile very useful for specifying hardware and software properties to perform our analysis.

12.2. 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 fulfills the reliability requirement expressed in temporal logic. The inputs parameters of each alternative is computed from the properties of our electronic automotive system. The overhead of such computation is negligible in our experiments since it occurred before the design space exploration.

12.3. 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 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 activity diagram partitioned on software components. We assume that the external environment factors do not influence the system reliability. In addition, 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.

13. Conclusion

In this paper, we presented a deployment-exploration approach of embedded software modeled by using 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 use SysML blocks diagram with additional real time system annotations using MARTE profile where behavior is expressed using 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 use a scheduling algorithm for allocation in AADL, leads to a difficulty to obtain a good 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 behavior to PRISM for model checking.

The research contribution of this work consists on capturing the system behavior as Discrete-Time Markov chains (DTMC) supported by PRISM language. We proposed a calculus dedicated to flow observation on blocks behavior that captures precisely the underlying semantics. In addition, we formalized PRISM language and showing its semantics. Moreover, we proved the soundness of our proposed approach by defining the relation between the semantics of the mapped diagrams and the resulting PRISM models. By this relation, we proved the preservation of the satisfiability of PCTL properties. We have shown the effectiveness of our approach in realistic case study: Embedded automotive control system.

The practical advantages of the proposed approach in the context of deployment, consists on providing key 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.

The presented work can be extended in the following two directions. First, we intend to extend our approach to support more constraints that affect the entire system behavior such as energy, bus loads and memory restrictions. Finally, we intend to translate the system into specific language 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.

Christel Baier, Joost-Pieter Katoen, Holger Hermanns, and Verena Wolf. Comparative branching-time semantics for markov chains. *Information and Computation*, 200(2):149 – 214, 2005. ISSN 0890-5401.

Abdelhakim Baouya, Djamal Bennouar, Otmane 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, Djamal Bennouar, Otmane 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.

Gerd Behrmann, Alexandre David, and KimG. Larsen. A tutorial on uppaal. In Marco Bernardo and Flavio Corradini, editors, *Formal Methods for the Design of Real-Time Systems*, volume 3185 of *Lecture Notes in Computer Science*, pages 200–236. Springer Berlin Heidelberg, 2004. ISBN 978-3-540-23068-7.

Loïc Besnard, Adnan Bouakaz, Thierry Gautier, Paul Le Guernic, Yue Ma, Jean-Pierre Talpin, and Huafeng Yu. Timed behavioural modelling and affine scheduling of embedded software architectures in the {AADL} using polychrony. *Science of Computer Programming*, 106:54 – 77, 2015. ISSN 0167-6423. Special Issue: Architecture-Driven Semantic Analysis of Embedded Systems.

Etienne Brosse, Imran Quadri, Andrey Sadovykh, Frank Ieromnimon, Dimitrios Kritharidis, Rafael Catrou, and Michel Sarlotte. Enosys fp7 eu project: An integrated modeling and synthesis flow for embedded systems design. In *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2012 7th International Workshop on*, pages 1–5, July 2012.

Jan Carlson, John Håkansson, and Paul Pettersson. Saveccm: An analysable component model for real-time systems. *Electronic Notes in Theoretical Computer Science*, 160:127 – 140, 2006. ISSN 1571-0661. Proceedings of the International Workshop on Formal Aspects of Component Software (FACS 2005)Proceedings of the International Workshop on Formal Aspects of Component Software (FACS 2005).

Abraham Cherfi, Michel Leeman, Florent Meurville, and Antoine Rauzy. Modeling automotive safety mechanisms: A markovian approach. *Reliability Engineering & System Safety*, 130:42 – 49, 2014. ISSN 0951-8320. doi: http://dx.doi.org/10.1016/j.ress.2014.04.013. URL http://www.sciencedirect.com/science/article/pii/S0951832014000817.

Lenny Delligatti. *SysML Distilled: A Brief Guide to the Systems Modeling Language*. Addison-Wesley Professional, 1st edition, 2013. ISBN 0321927869, 9780321927866.

Francisco Assis Moreira do Nascimento, MarcioF.S. Oliveira, and FlÃ¡vioRech Wagner. A model-driven engineering framework for embedded systems design. *Innovations in Systems and Software Engineering*, 8(1):19–33, 2012. ISSN 1614-5046.

Peter Feiler. Model-based validation of safety-critical embedded systems. In *Aerospace Conference, 2010 IEEE*, pages 1–10, March 2010.

Antonio Filieri, Giordano Tamburrelli, and Carlo Ghezzi. Supporting self-adaptation via quantitative verification and sensitivity analysis at run time. *IEEE Transactions on Software Engineering*, 42(1):75–99, 2016.

Areski Flissi, Jérémy Dubus, Nicolas Dolet, and Philippe Merle. Deploying on the grid with deployware. In *Eighth IEEE International Symposium on Cluster Computing and the Grid*, pages 177–184, France, 2008.

Vojtěch Forejt, Marta Kwiatkowska, Gethin Norman, and David Parker. *Formal Methods for Eternal Networked Software Systems: 11th International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM 2011, Bertinoro, Italy, June 13-18, 2011. Advanced Lectures*, chapter Automated Verification Techniques for Probabilistic Systems, pages 53–113. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. ISBN 978-3-642-21455-4.

João M. Franco, Francisco Correia, Raul Barbosa, Mário Zenha-Rela, Bradley Schmerl, and David Garlan. Improving self-adaptation planning through software architecture-based stochastic modeling. *Journal of Systems and Software*, 115:42 – 60, 2016. ISSN 0164-1212.

Sanford Friedenthal, Alan Moore, and Rick Steiner. *A Practical Guide to SysML: Systems Modeling Language*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008. ISBN 0123743796, 9780080558363, 9780123743794.

Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979. ISBN 0716710447.

Günter Heiner and Thomas Thurner. Time-triggered architecture for safety-related distributed real-time systems in transportation systems. In *Digest of Papers: FTCS-28, The Twenty-Eigth Annual International Symposium on Fault-Tolerant Computing, Munich, Germany, June 23-25, 1998*, pages 402–407, 1998. doi: 10.1109/FTCS.1998.689491.

Fernando Herrera, Héctor Posadas, Pablo Peñil, Eugenio Villar, Francisco Ferrero, Raúl Valencia, and Gianluca Palermo. The {COMPLEX} methodology for uml/marte modeling and design space exploration of embedded systems. *Journal of Systems Architecture*, 60(1):55 – 78, 2014. ISSN 1383-7621.

Khaza Anuarul Hoque, Otmane Ait Mohamed, Yvon Savaria, and Claude Thibeault. Early analysis of soft error effects for aerospace applications using probabilistic model checking. In Cyrille Artho and Peter Csaba Ã–lveczky, editors, *Formal Techniques for Safety-Critical Systems*, volume 419 of *Communications in Computer and Information Science*, pages 54–70. Springer International Publishing, 2014. ISBN 978-3-319-05415-5.

Remy Houssin and Amadou Coulibaly. Safety-based availability assessment at design stage. *Computers & Industrial Engineering*, 70: 107 – 115, 2014. ISSN 0360-8352. doi: http://dx.doi.org/10.1016/j.cie.2014.01.005.

Zai J. Jia, Antonio Núñez, Tomás Bautista, and Andy D. Pimentel. A two-phase design space exploration strategy for system-level real-time application mapping onto mpsoc. *Microprocessors and Microsystems*, 38(1):9–21, 2014.

Gideon Juve and Ewa Deelman. Automating application deployment in infrastructure clouds. In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, pages 658–665, 2011.

Marta Kwiatkowska, Gethin Norman, and António Pacheco. *Process Algebra and Probabilistic Methods: Performance Modeling and Verification: Second Joint International Workshop PAPM-PROBMIV 2002 Copenhagen, Denmark, July 25–26, 2002 Proceedings*, chapter Model Checking CSL until Formulae with Random Time Bounds, pages 152–168. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002. ISBN 978-3-540-45605-6.

Marta Kwiatkowska, , Gethin Norman, and David Parker. Stochastic model checking. In M. Bernardo and J. Hillston, editors, *Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM'07)*, volume 4486 of *LNCS (Tutorial Volume)*, pages 220–270. Springer, 2007.

Marta Kwiatkowska, Gethin Norman, and David Parker. Stochastic games for verification of probabilistic timed automata. In J. Ouaknine 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.

Robin Milner. *Communicating and Mobile Systems: The &Pgr;-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 mpsoc 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, Otmane Aït Mohamed, and Mourad Debbabi. A property-based abstraction framework for sysml activity diagrams.

Knowledge-Based Systems, 56(0):328–343, 2014a.

Samir Ouchani, Otmane 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 mpsocs. *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. Simulink. 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).