A Methodology for Model-based Verification of Safety Contracts

Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability
000(00):1–13

DOI:doi number
http://mms.sagepub.com

Elena Gómez-Martínez[†,*], Ricardo J. Rodríguez[‡], Clara Benac Earle[†], Leire Etxeberria Elorza[♯] and Miren Illarramendi Rezabal[♯]

[†]*Babel Group, ETSINF, Universidad Politécnica de Madrid, Spain*

[‡]*Research Institute of Applied Sciences in Cybersecurity, University of León, Spain*

[♯]*Embedded Systems Research Group, MGEP, Mondragon Unibertsitatea, Arrasate-Mondragón, Spain*

Abstract

The verification of safety requirements becomes crucial in critical systems where human lives depend on their correct functioning. Formal methods have often been advocated as necessary to ensure the reliability of software systems, albeit with a considerable effort. In any case, such an effort is cost-effective when verifying safety-critical systems. Often, safety requirements are expressed using safety contracts, in terms of assumptions and guarantees.

To facilitate the adoption of formal methods in the safety-critical software industry, we propose a methodology based on well-known modelling languages such as UML and OCL. UML is used to model the software system while OCL is used to express the system safety contracts within UML. In the proposed methodology a UML model enriched with OCL constraints is transformed to a Petri net model that enables to formally verify such safety contracts. The methodology is evaluated on an industrial case study. The proposed approach allows an early safety verification to be performed, which increases the confidence of software engineers while designing the system.

* Corresponding author; e-mail: elena.gomez.martinez@upm.es

Keywords
safety contracts, model-based, verification, Petri nets

1. Introduction

The growing adoption of software in safety-critical systems has put safety assessment in the spotlight, becoming a crucial software engineering task as recognised by several initiatives (e.g., the ARTEMIS JU nSafeCer project [1]). Moreover, the design and development of a system must be done with safety in mind, rather than add it in as an afterthought [2]. Several real examples can be found in the literature showing the impact of a lack of safety assessment in industrial systems, such as the flight errors of Air Canada airline, the explosion caused by an overflow in Ariane 5, or the unavailability of the Patriot Missile control system in US army base camps [3; 4]. The earlier safety assessment is carried out in a system, the sooner it can be redesigned to properly fulfil safety requirements, thus saving production and other costs.

Contract-based design is a popular approach for the design of complex component-based systems where safety properties are difficult to guarantee [5; 6]. A key benefit of using contracts is that they follow the principle of separation of concerns [7], separating assumptions that the environment

of a component obeys from what a component guarantees under such an environment.

The Unified Modelling Language (UML) [8] is widely adopted to model the design of a system. By providing the means to include safety requirements in UML, the integration of safety activities in the normal software lifecycle is facilitated. For safety specification, two approaches have been proposed: (i) to use the Object Constraint Language (OCL) [9], which is a well-known language among the modelling engineering community; and (ii) to use specific UML profiles [10], which extend the semantics of UML models. In [11], the use of both techniques was proposed to express fire prevention requirements of a hospital facility.

Safety requirements modelling is an important part when designing, but some mechanisms must be also provided to assess that safety requirements are indeed fulfilled [2]. To this goal, several formal verification techniques have been proposed (e.g., model checking [6]).

In this paper, we focus on safety requirements and assessment in UML models. Namely, we explore the representation of safety requirements as OCL constraints and their verification using Petri nets [12] as the formal model, obtained after transformation of UML models enriched with OCL, plus UML profiles. By combining standard engineering practice, i.e., UML modelling, with formal verification techniques, i.e., Petri nets, we provide a rigorous safety analysis available for software engineers.

As case study, we evaluate our approach in a real industrial scenario. We model a train door controller with UML, specify its safety requirements, transform these models to Petri nets, and analyse them using well-established analysis tools. The train door controller is in charge of opening and closing train doors and is developed by CAF Power & Automation company[1].

A previous version of this work can be found in [13]. Several enhancements have been carried out with respect to the aforementioned work. In particular, the contribution of this paper is threefold: (i) We propose a three-step methodology for model-based safety verification; (ii) We generalise and formalise the specification of safety requirements as OCL constraints; and (iii) we formalise the transformation from OCL to Petri nets.

[1] http://www.cafpower.com/es/

The rest of the paper contains the following sections. Firstly, Section 2 outlines the basic concepts. Section 3 presents our methodology for model-based verification of safety contracts. Then, Section 4 applies our proposal to an industrial case study. Finally, Section 5 covers related work and Section 6 states some conclusions and future work.

2. Previous Concepts

2.1. UML

The Unified Modeling Language (UML) [8; 14] is a semi-formal general-purpose visual modelling language used for specifying software systems. In this paper, some knowledge of UML is assumed. For more details we refer to [8; 14].

UML can be tailored for specific purposes by profiling. Profiling was introduced by UML to indeed add new capabilities to the language. A UML profile is a UML extension to enrich UML model semantics defined in terms of: *stereotypes* (concepts in the target domain), *tagged values* (attributes of the stereotypes) and *constraints* (formulae that apply to stereotypes and UML elements to extend their semantics). Numerous UML profiles can be found in the literature targeting different specific domains and non-functional properties system analysis (e.g., performance, dependability, security, etc.). For instance, MARTE (Modeling and Analysis of Real-Time and Embedded systems) profile [15] provides support for schedulability and performance analysis in real-time and embedded systems, while DAM (Dependability Analysis and Modelling) profile [16] supports dependability analysis and SecAM (Security Analysis and Modelling) profile [17] focuses on security aspects. In this paper, we use the MARTE profile to indicate the duration of activities in a UML model.

Another extension to enrich UML semantics is the Object Constraint Language (OCL) [9]. OCL is briefly introduced in the following section.

2.2. Object Constraint Language

The Object Constraint Language (OCL) [9] is a formal language used to describe constraints on UML models. The main purpose of OCL is to provide additional relevant information to a UML diagram while avoiding ambiguities arising from the use of informal specification languages.

Compared to other formal languages, OCL is sufficiently simple as to be usable in an industrial setting.

When an OCL expression is evaluated, it simply returns a value without further effects in the model. OCL allows to specify invariants (on classes and types), to describe pre- and post-conditions (on operations and methods), guards or either constraints (on operations). Note that although an OCL expression can be used to specify a state change (e.g., by means of a post-condition), the state of the system will never effectively change because of the evaluation of an OCL expression (that is, OCL only provides textual description).

Unfortunately, a UML model annotated with OCL and a profile that provides support for non-functional properties specification is not a suitable model to quantitatively or qualitatively evaluate such properties. For this aim, we propose the use of Generalized Stochastic Petri Nets, which are introduced in the following section.

2.3. Generalized Stochastic Petri Nets

In this paper, we consider Petri nets [12] as the formal modelling language. More precisely, we translate the annotated UML diagrams into Generalized Stochastic Petri Nets (GSPNs) [18], following the guidelines proposed in [19].

GSPNs are a graphical and mathematical modelling tool for describing concurrent systems. A gentle introduction to GSPN can be found in [18]. A GSPN system is a 8-tuple $\mathcal{S} = \langle P, T, \Pi, I, O, H, W, M_0 \rangle$, where:

- P is the set of places.
- T is the set of immediate and timed transitions, $P \cap T = \emptyset$.
- $\Pi : T \to \mathbb{N}$ is the priority function that maps transitions onto priority levels, by default, timed transitions have priority equal to zero.
- $I, O, H : T \to 2^P$ are the input, output, inhibition functions, respectively, that map transitions onto the powerset of P.
- $W : T \to \mathbb{R}_{\geq 0}$ is the weight function that assigns rates of timed transitions and weights to immediate transitions.
- $M_0 : P \to \mathbb{N}$ is the initial marking.

The *pre-* and *post-sets* of a node $v \in P \cup T$ are respectively defined as ${}^\bullet v = \{u \in P \cup T | (u, v) \in F\}$ and $v^\bullet = \{u \in P \cup T | (v, u) \in F\}$, where $F \subseteq (P \times T) \cup (T \times P)$ is the set of directed arcs. Graphically, a GSPN is a bipartite graph of places and transitions joined by arcs; places and transitions are respectively represented by circles and bars, arcs are shown by arrows. They describe the flow of the system with concurrency and synchronous capabilities. Places can hold tokens, which represent system resources or system workload, while transitions represent system activities. The firing of transitions represents a change in the system state. When a transition fires, tokens from input places are placed in output places. A GSPN distinguishes two kind of transitions: immediate transitions, which fire at zero time (i.e. its firing does not consume any time); and timed transitions, which may follow different firing distributions such as uniform, deterministic or exponential distributions. In this paper, we consider timed transitions with exponentially distributed random firings. Immediate transitions, depicted as thin black bars, can have also associated probabilities to represent the system routing alternatives. Exponential transitions, drawn as white boxes, account for the time that takes an activity to complete.

3. A Methodology for Model-based Safety Assessment

In this paper, we present a scenario-based methodology to verify safety requirements at early stages. Thus, it allows to assess safety as a "by-product" of the software life-cycle. The proposed methodology is an extension of the performance analysis methodology presented in [20], which is based on principles and techniques of Software Performance Engineering (SPE) [21]. The aforementioned methodology brings safety analysis in by means of UML diagrams enriched with OCL constraints.

The proposed methodology comprises three different phases, as depicted in Figure 1.

Safety-Oriented Design Phase. The software system is modelled using UML diagrams, mainly UML Composite Diagram (UML-CD), UML Sequence Diagrams (UML-SD), and UML State Machine Diagrams (UML-SM). The goal is to capture the structure and dynamics (i.e., behaviour) of the system. The activity output is therefore a set of UML diagrams representing the software system.

Safety Specification Phase. Each potential scenario where safety issues may arise is further specified by means

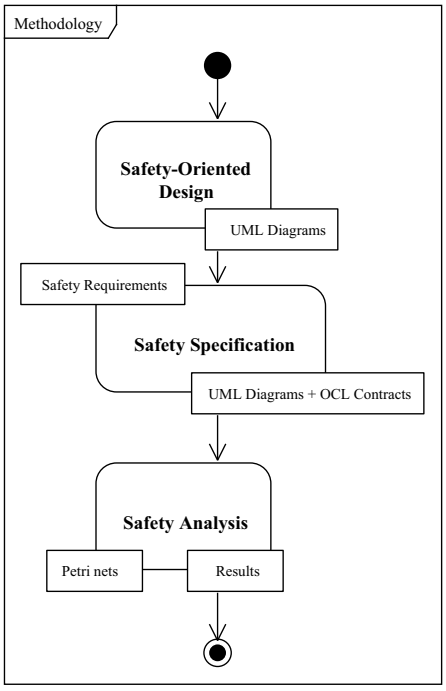


Fig. 1. Methodology for model-based safety analysis.

of safety requirements. These requirements are first expressed as Safety Contracts Fragments, and then translated into OCL within UML models. UML diagrams annotated with OCL constraints are obtained as results.

Safety Analysis Phase. Lastly, UML models annotated with OCL constraints are translated into the so-call *safety model* where a safety analysis is carried out. As safety model, we use the formalism of Generalized Stochastic Petri nets [18]. As outcomes of this activity, we obtain this Petri net and the result of safety analysis.

In the sequel, we describe in detail these phases. The proposed methodology is evaluated in Section 4, where safety properties of an industrial case study are verified.

3.1. Safety-Oriented Design Phase

The first step in our methodology is to describe the structure of a critical system and its behaviour. Since reuse of software components is key in, for instance, aerospace and automotive domains [22], a component-based design is followed. This component-based design is described using UML [8]. For the structural part of the model we use Composite Structure Diagrams, in particular, the SysML [23] extension. The reason is that in SysML input and output ports can be defined and they are used in the safety contract as explained in Sect. 3.2 In addition to Composite Structure Diagrams, Sequence Diagrams and State Machine Diagrams are used to model the dynamics of the system or component.

A UML Composite Structure Diagram (UML-CS) is a type of static structure diagram which represents the internal structure of a structured classifier or collaboration to describe a functionality. Thus, a Component Structure Diagram represents runtime instances collaborating over communication links to achieve some common objectives. This diagram can include, among others, *parts*, a set of one or more instances which are owned by a containing classifier instance, and *ports*, which define a property of a classifier that specifies a distinct interaction point between that classifier and its environment or between the (behaviour of the) classifier and its internal parts. Ports may specify inputs, outputs as well as operating bidirectionally. In contract-based design each safety critical component of the system and non-critical components are seen as separated components [24] which interact with their environment. In the proposed methodology, we use Composite Structure Diagrams to represent components and subcomponents of a critical system.

In component-based design, the internal states of a component are modelled using a UML State Machine Diagram (UML-SM), which describes its lifetime. A state represents a time period in the life of an object during which the component satisfies some condition, performs some action or waits for an event.

Potential critical scenarios are modelled using Sequence Diagrams. A UML Sequence Diagram (UML-SD) is a type of behavioural UML diagram which shows object interactions. More specifically, the messages exchanged between the system components arranged in time sequence. It provides useful constructors such as loops, alternatives or parallel execution. It is used to model usage scenarios with respect to a timeline. We augment these diagrams to specify duration activities by means of MARTE profile [15]. The stereotype provided by MARTE to this goal is `gaStep` (`hostDemand` tagged value), within the MARTE analysis framework called Generic Quantitative Analysis Model (GQAM).

3.2. Safety-Oriented Specification Phase

The second step encompasses the definition of safety requirements to be verified. In this paper, we assume that safety requirements are informally captured from natural language and formally specified as Safety Contract Fragments (SCF) [25]. These SCF are later transformed to OCL and integrated into UML models to be analysed in the next phase.

A SCF defines a safety contract as a set of assumptions and a set of guarantees, for a given component and under a given environment. An assumption is what it is expected to be met by the environment, while a guarantee specifies how the component behaves in such an environment. A component of a component-based system can be formally defined as:

Definition 1. *A component $\mathcal{C} = \langle \mathcal{I}, \mathcal{O} \rangle$ of a system is composed of a set $\mathcal{I}$ of input ports and a set $\mathcal{O}$ of output ports.*

Given Definition (1), a SCF $\mathcal{S}_\mathcal{C}$ of a component $\mathcal{C} = \langle \mathcal{I}, \mathcal{O} \rangle$ can be defined as:

Definition 2. *A SFC $\mathcal{S}_\mathcal{C} = \langle \mathcal{A}, \mathcal{G} \rangle$ of a component $\mathcal{C} = \langle \mathcal{I}, \mathcal{O} \rangle$ is a tuple of a superset of disjoint sets $\mathcal{A} = \mathcal{A}^+, \mathcal{A}^*$ of OR and AND safety assumptions, respectively, and a superset of disjoint sets $\mathcal{G} = \mathcal{G}^+, \mathcal{G}^*$ of OR and AND guarantees[2]. Besides, a safety assumption $a \in \mathcal{A}$ is a proposition that relates one or more of the input ports of a component, i.e., $a : \mathcal{A} \rightarrow \mathcal{I}^n, n \geq 1$. Similarly, a safety guarantee $g \in \mathcal{G}$ is a proposition that relates one or more of the output ports of a component, i.e., $g : \mathcal{G} \rightarrow \mathcal{O}^m, m \geq 1$.*

Thus, a SCF $\mathcal{S}_\mathcal{C}$ defined over a component $\mathcal{C} = \langle \mathcal{I}, \mathcal{O} \rangle$ relates the input and output ports of the component with the assumptions (i.e., what it is expected) and guarantees (i.e., what it is performed), respectively. Note that for us how the guarantees are achieved is a black-box operation. Besides, the guarantees are only assured when the assumptions are fulfilled. Otherwise, the result of the component is not guaranteed and thus, cannot be trusted as a well-performed operation.

Recall that OCL is a UML extension to express constraints acting over a context into UML models (see Section 2.2). Among other constraints, an OCL can define invariants (`inv`) as state conditions always fulfilled, or pre/post-conditions fulfilled before/after an operation is performed. In this paper, we focus on OCL invariants.

Definition 3. *An OCL constraint $\mathcal{R} = \langle \mathcal{X}, \mathcal{V} \rangle$ is a tuple conformed by the context $\mathcal{X}$ where is defined, and the invariant formula $\mathcal{V} = \langle ls, rs \rangle$, where ls, rs are two logical propositions joined by a boolean or* `implies` *operator.*

Following the above definitions, we can straightforwardly map an SCF into a OCL constraint:

Definition 4. *An OCL constraint $\mathcal{R} = \langle \mathcal{C}, \mathcal{S}_\mathcal{C} \rangle$ describes a context defined by a component $\mathcal{C}$, and an invariant formula defined by the SCF $\mathcal{S}_\mathcal{C} = \langle \mathcal{A}, \mathcal{G} \rangle$.*

Thus, a SCF defined over a component can be mapped into a OCL constraint, and integrated within UML models. In the next phase, these OCL constraints are transformed to Petri nets to drag the safety requirements into the analysis step.

3.3. Safety-Oriented Analysis Phase

Finally, we translate UML diagrams annotated with MARTE and OCL profiles into Petri nets, namely generalized stochastic (GSPN) [18]. This choice has been driven by two main factors: (i) GSPNs provide a formal notation which avoids any source of ambiguity while representing the stochastic behaviour of systems; (ii) GSPNs have a clear graphical notation and several tools have been developed for analysis (for instance, GreatSPN [26], TimeNET [27], or PeabraiN [28], among others).

The translation process is carried out in two steps: First, UML annotated with MARTE profile; and then, OCL constraints. To perform the first step, we use ArgoSPE [29], a tool that automatically translates UML diagrams augmented with duration activities using MARTE profile into GSPNs. This tool implements the algorithms given in [30] and [19]. In broad outline, object states and resources are map into places within the Petri net. Events and actions are translated into immediate and timed transitions, respectively. The average execution of an action is specified with an exponentially distributed random variable. Alternative fragments or conditions are translated into transition with probabilities.

OCL constraints are transformed to GSPN as follows. Recall that each OCL invariant $\mathcal{S}_\mathcal{C}$ of a constraint $\langle \mathcal{C}, \mathcal{S}_\mathcal{C} \rangle$ is a

[2] As in [11], for the sake of simplicity we restrict the logic of SCF assumptions and guarantees to AND and OR logic operators.

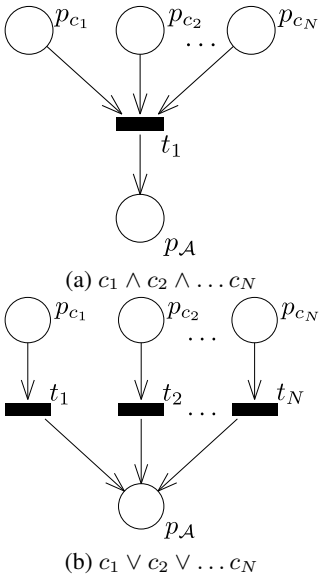


Fig. 2. From assumption clauses to Petri nets (following guidelines given in [31]).

proposition of a set of assumptions $\mathcal{A}$ implying (`implies` binary operator, $\rightarrow$) a set of guarantees. Thus, we firstly transform each OCL invariant in its equivalent form, following the equivalence rule of the `implies` binary operator in classical logic (i.e., $p \rightarrow q$ is logically equivalent to $\neg p \vee q$). The proposition in equivalent form is used to build a representative GSPN model of the OCL constraint.

The transformation is performed following a bottom-up approach. First, a place $p_{\mathcal{SC}}$ representing the fulfillment of $\mathcal{SC}$ is added. Then, two places $p_{\mathcal{A}}, p_{\mathcal{G}}$, with two output transition $p^{\bullet}_{\mathcal{A}} = \{t_{\mathcal{A}}\}$, $p^{\bullet}_{\mathcal{G}} = \{t_{\mathcal{G}}\}$ are added, and ${}^{\bullet}p_{\mathcal{SC}} = \{t_{\mathcal{A}}, t_{\mathcal{G}}\}$. Now, assumptions and guarantees are processed: for each assumption $a \in \mathcal{A}$, guarantee $g \in \mathcal{G}$, a places p_a, p_g, are created. These places are interconnected among them and with the aforementioned places $p_{\mathcal{A}}$, $p_{\mathcal{G}}$ depending on the logic operator that join the clauses and following the guidelines given in [31]. Figure 2 depicts an example of transformation to Petri nets of a set of N assumptions joined with AND and OR logic (the transformation of guarantees is equivalent, but the last place is $p_{\mathcal{G}}$ instead of $p_{\mathcal{A}}$).

However, the translation of OCL constraints is not automatically integrated by ArgoSPE. Thus, some manual tuning is needed. This manual tuning encompasses to include a place for each constraint related to an event. Each OCL constraint is represented by a place of the GSPN. Nevertheless, according to the algorithms proposed in [30] and [19], events are translated into GSPNs by means of transitions. Therefore, we need to transform each transition into a pair transition-place in order to compute the probability.

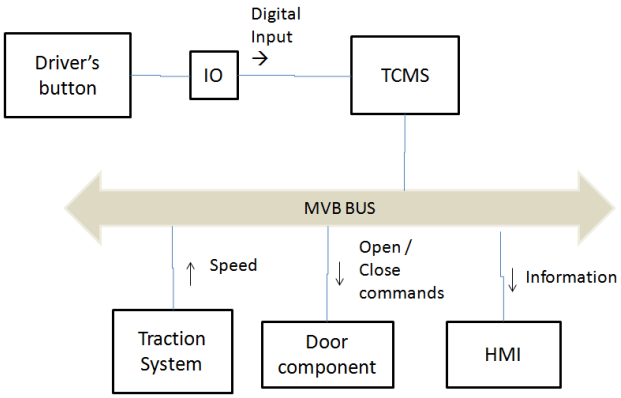


Fig. 3. The TCMS System and others components.

Once obtained the Petri nets representing the system and the safety constraints, both nets are merged using the transitions that create tokens in places representing the same issue. The composition is made by the algebra tool of GreatSPN [26].

Finally, we use the GreatSPN to compute the steady-state probability of places that represent conditions. A marking greater than zero (i.e. the place is eventually marked) will indicate that OCL constraints are fulfilled.

4. Case Study: A Train Doors Controller

In this section, we pursue to perform an early safety verification in a real industrial case study, following the methodology proposed in Section 3.

4.1. System Description

As case study, we consider the door control management performed by a Train Control and Monitoring System (TCMS). The TCMS is a complex distributed system that controls many subsystems such as the door control, traction system control, air conditioning control, video surveillance, passenger information system, etc. The TMCS provides information to the driver, such as the state of doors, the state of the traction, or the state of the alarm system, which is gathered by a set of Input/Output (IO) modules. Figure 3 shows the communication architecture among TMCS and other train subsystems.

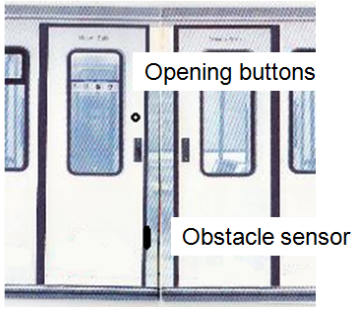


Fig. 4. The image of the door.

The system level requirements concerning the operation of opening and closing of doors are satisfied by the following components:

- The `TCMS` component decides whether to enable or disable the doors considering the driver's requests and the train movement. Thus, doors must be enabled before they can be opened, and disabled before closing;
- The `Door` component controls and commands the opening and closing of a door;
- The `Traction` component controls and commands the train movement; and
- The `MVB` (Multifunction Vehicle Bus) component intercommunicates the components.

Door control systems differ depending on the type of train where they are acting. For instance, a door of a suburban or underground train has a button that enables passengers to open it upon request, while in the case of long distance and high speed trains, doors have no buttons since they are opened only upon the driver's request. In this paper, we consider a door control system in a suburban train, i.e., a door has open buttons inside and outside the train coach. Note that in this case the driver must first enable doors before they can be opened upon passenger's request. Doors also include an obstacle sensor to prevent a closing operation when an obstacle is detected. Figure 4 depicts the door considered in this system.

The train subsystems such as the door control system are safety-critical systems and, therefore, railway standards must be applied during their development. The major standards are the European EN5012x family of railway standards:

- EN50126 [32]: Railway specifications — The specification and demonstration of Reliability, Availability, Maintainability and Safety (RAMS);
- EN50128 [33]: Railway applications — Communication, signalling and processing systems — Software for railway control and protection systems, it is known as the Railway Software Standard and is a specialisation of IEC 61508 for railway;
- EN50129 [34]: Railway applications — Communication, signalling and processing systems — Safety related electronic systems for signalling. EN 50129 gives precise guidance how to build a safety case and particularly what has to be included in the various parts of it;
- EN 50159-x [35]: Safety related communication. EN 50159-1 is dedicated to closed transmission systems and EN 50159-2 is dedicated to open transmission systems;
- EN 50121-x [36] is relevant for EMC;
- EN 50125-x [37] for environmental conditions of railway equipment;

The Safety Integrity Level (SIL) of a door control system is **SIL 2**. A SIL specifies a target level of risk reduction and is typically defined in components that operate in a safety-critical system [38] [33].

The case study presented here concerns a real system where some simplifications were made. Namely, the interaction with other components of the TCMS, the dependencies with other subcomponents, and their communication were omitted.

4.2. On Safety-Oriented Design Phase

According to the methodology proposed in Section 3, the first step is to design the critical system. In the following, we describe each safety-critical component in detail.

Figure 5 shows the UML-CS of the Train System. The system is composed by a `TCMS` component, N `Door` components, a `Traction` component and a `MVB` component. The Train System has two external input ports, connected to the input ports of `TCMS` component (namely, *open_door* and *close_door*), which receive the driver requests for enabling or disabling the doors. Output ports of the system report about the status of the overall system: A *door_status* enumerated value to indicate whether the doors are being opened, closed, already open, or already closed; a *door_enabled*

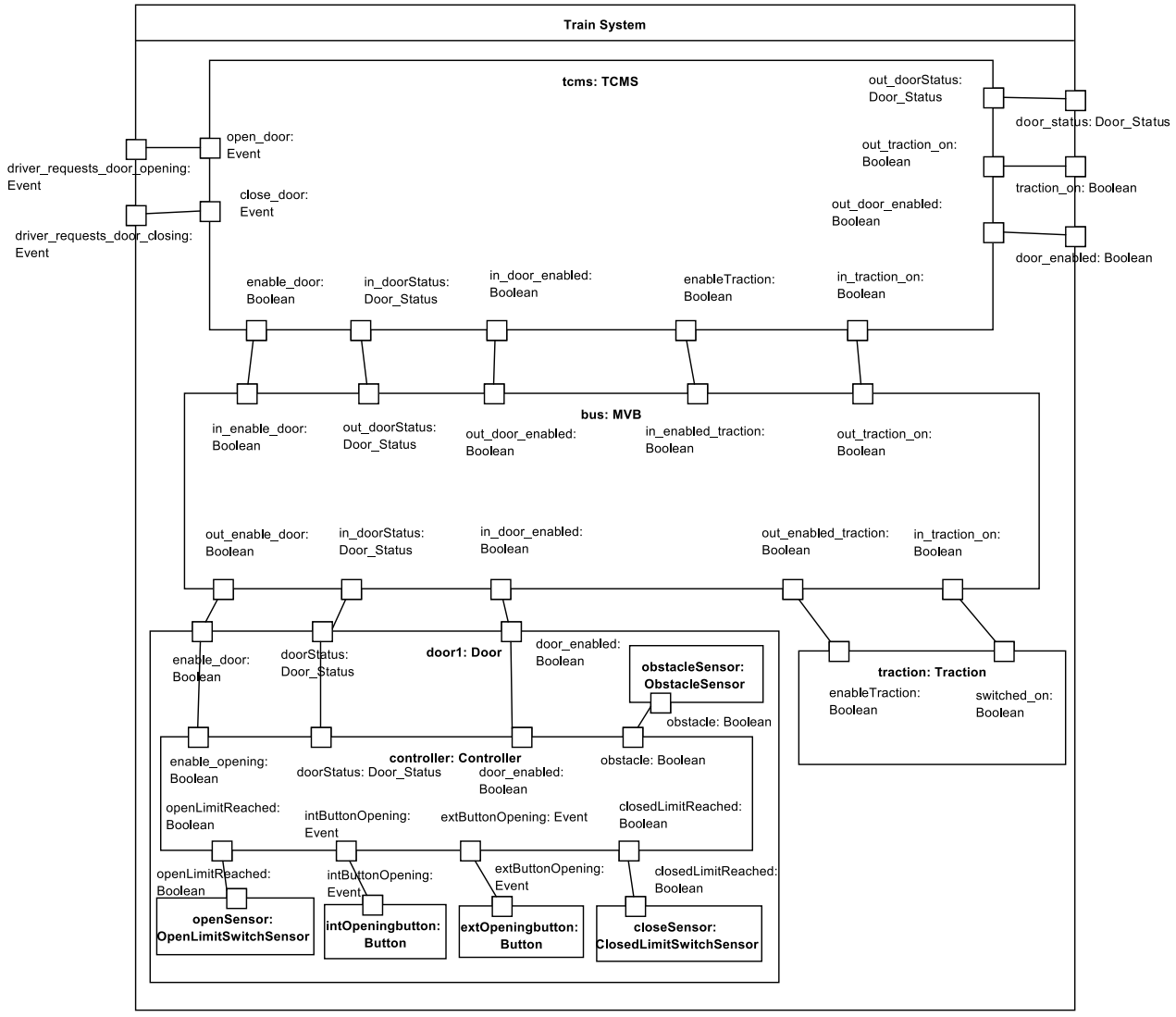


Fig. 5. UML Composite Structure Diagram of the Train System.

boolean value to indicate whether the doors are enabled or disabled; and a *traction_on* boolean value to express whether the traction system is on or off.

As input, the `Door` component receives the command to enable or disable the door (*enable_door* boolean value) from `TCMS` component. As output, the `Door` component reports about the status of the door (*doorStatus* enumerated value), and whether the door is enabled (*door_enabled* boolean value). Note that these outputs are in fact inputs port for `TCMS` component.

`Traction` component receives as input an enable/disable traction command (*enableTraction* boolean value) from `TCMS` component; and provides as output a boolean flag to indicate the traction status (*switched_on* boolean value).

Finally, the `MVB` component represents the communication among the components of the Train System.

Figure 5 also shows the subcomponents of the `Door` component, i.e., the controller (in the following we name it as `DoorController`), the limit sensors, the obstacle sensor, and the interior/exterior opening buttons.

Figure 6 shows the UML-SM of `DoorController`. It has four states: *opening*, *isOpen*, *closing*, and *isClosed* (initial state). The interior/exterior opening buttons trigger when pushed the *intButtonOpening* and *extButtonOpening* events, which lead the `DoorController` state to *opening* state, if *enableDoor* is true. Once the door is totally open, `openSensor` triggers an *openLimitReached* event that causes the `DoorComponent` to change to *isOpen* state. It remains in this state until the door is disabled and no obstacle is detected, moving to *closing* state. In this state, two exits are possible: When an obstacle is detected, or the interior/exterior opening buttons are pushed and the door is enabled, the `DoorController` state moves to *opening* state again; When the `closeSensor` component triggers a *closedLimitReached* event, since the door has been totally closed, the `DoorController` is lead to *isClosed* state.

As critical operations, we focus on the control of doors. In the following, we present the UML Sequence Diagrams (UML-SD) for the opening and closing of doors.

Figure 7 depicts the UML-SD for door opening scenario. When a train driver requests the opening of doors, the TCMS first checks whether the train status is suitable for opening the doors without risk, i.e., the train is really stopped. Whether

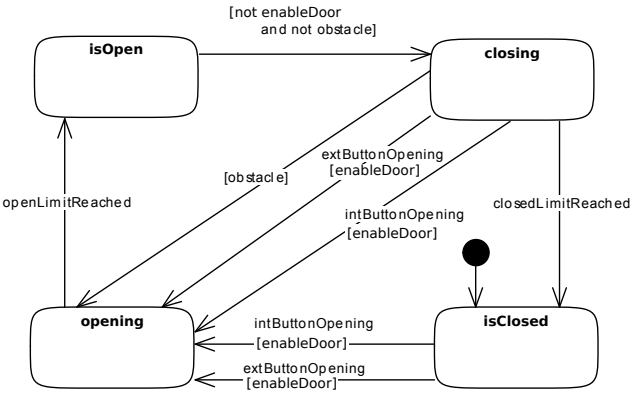


Fig. 6. UML State Machine Diagram of the `DoorController`.

this safety constraint is fulfilled, the "enable door" command is sent to the `DoorController` component. Then, the `DoorController` component opens the door when enabled and upon passenger's request, which is sent when a passenger press the interior/exterior opening door button.

Similarly, the door closing scenario is shown in Figure 8. When the driver commands doors closing, the TCMS system sends the "not enable door" command to the `DoorController` component. The `DoorController` component disables the door and closes it when the operation can be safely completed, i.e., there is no any obstacle detected. Otherwise, the door is opened, and closing operation is again carried out. Recall that this closing/opening loop occurs until the door can be safely closed.

4.3. On Safety-Oriented Specification Phase

A safety engineer defines the following safety requirements (SR) in the context of this case study:

SR1. A door can be opened when enabled and traction is off.

SR2. A door cannot be closed when an obstacle is detected.

SR3. A door is closed when the door opening is enabled and the close event is received.

In this phase, these requirements are expressed in terms of SCFs considering the component-based system depicted in Figure 5:

- $\mathcal{SR}_1 = \langle door\_enabled \wedge \neg traction\_on, doorStatus = OPENING \rangle$, defined on the `TCMS` component.

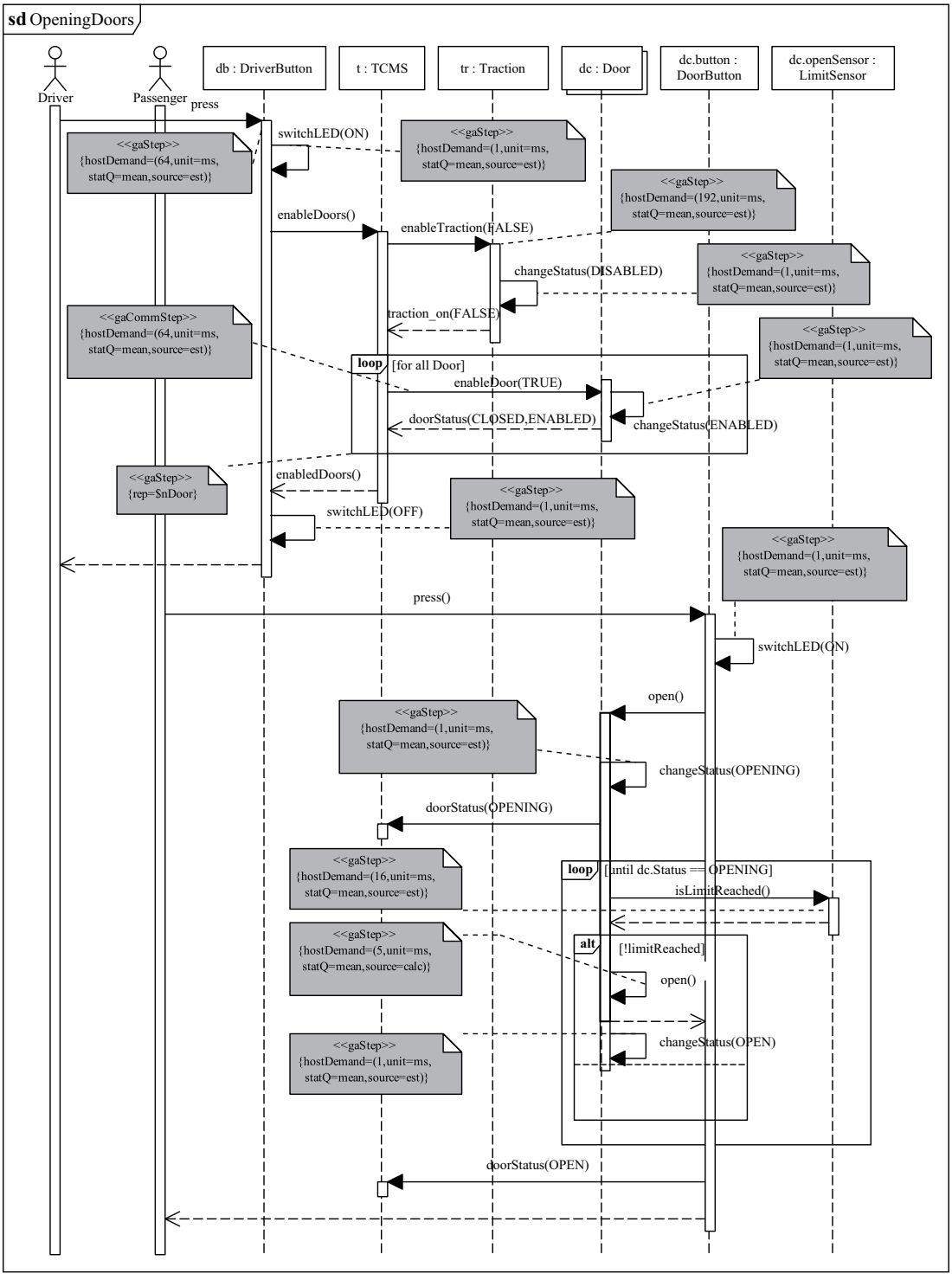


Fig. 7. UML Sequence Diagram representing the door opening operation.

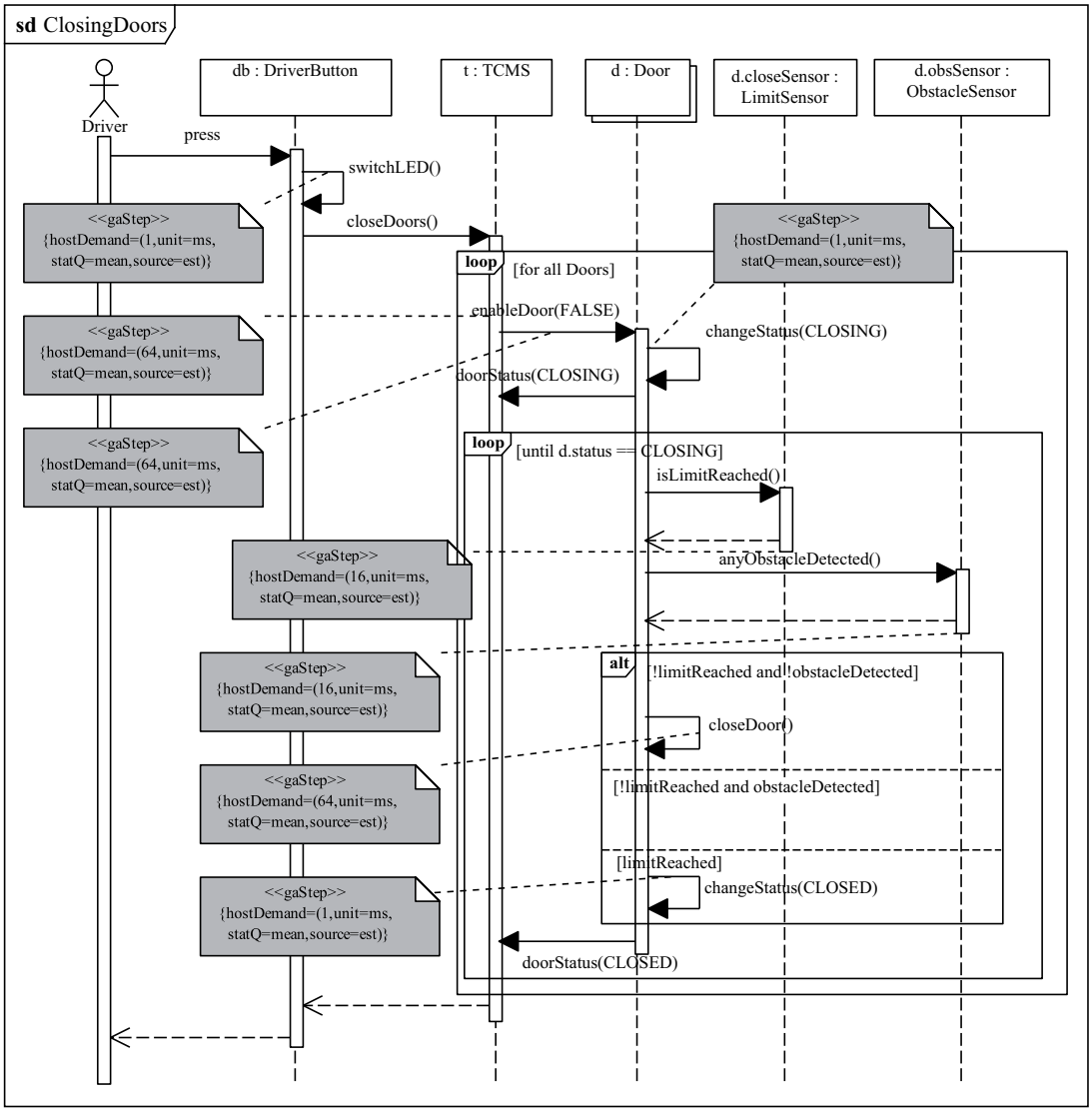


Fig. 8. UML Sequence Diagram representing the door closing operation.

- $\mathcal{SR}_2 = \langle obstacle, doorStatus = OPENING \rangle$. In this case, defined on the `DoorController` component.
- $\mathcal{SR}_3 = \langle door\_enabled \wedge close\_door, doorStatus = IS\_CLOSED \rangle$. This SCF is defined also on the `TCMS` component.

Note that $\mathcal{SR}_1, \mathcal{SR}_3$, requirements are defined on the `TCMS` component, while the context of $\mathcal{SR}_2$ is the `DoorController` component since the input and output ports that relate the $\mathcal{SR}_2$ belong to `DoorController`. As we have previously defined in Section 3.2, assumptions and guarantees of an SCF relate input and output ports of the components where they are defined.

Following Definition 4, these SCF are transformed into OCL, embedded within the UML-CD of the system. Namely, the constraints expressed as OCL language are shown at Listing 1.

The context of OCL is directly taken from were SCF are defined. Finally, these OCL rules are transformed to Petri nets and integrated within the GSPN of the case study, as we explain in the next section.

4.4. On Safety-Oriented Analysis Phase

Following the methodology proposed in Section 3, this phase encompasses the transformation of the safety scenarios described by UML-SDs, enriched with MARTE and OCL constraints, into GSPNs.

For this purpose, we use the ArgoSPE tool developed by [29]. Figure 9 depicts the GSPN obtained after transformation of UML-SD shown in Figures 7 and 8. The left-hand side of the figure represents the door opening scenario, while the right-hand side represents the door closing. The transformation process is partially done in an automatic way by ArgoSPE, since OCL constraints transformation is unsupported by the current release of ArgoSPE and thus some manual tuning is needed.

Some additional modifications are made manually. Specifically, those related to other elements of the system that are not completely considered in the first Safety-Oriented Design Phase. In particular, we have also modelled the Traction operation without considering human interaction, thus, our system automatically speeds up after closing the door and it brakes when the traction receives a traction stop signal.

OCL constraints described in the previous section are now transformed into Petri nets, following the guidelines given in Section 3.3. The Petri nets generated from $\mathcal{SR}_1, \mathcal{SR}_2$, and $\mathcal{SR}_3$ are depicted in Figure 10. Let us briefly exemplify how a PN representing an OCL constraint is built. Consider OCL constraint $TCMS\_SR1$. Applying logic equivalence formulae, such an invariant is equivalent to $(\neg door\_enabled \vee traction\_on) \vee doorStatus = OPENING$. A place is generated to represent each of the clauses, and extra places/-transitions are added to join them into a place that represents the own OCL constraint (place $\mathcal{SR}_1$, in this case, see Section 3.3).

Our aim during analysis is to check whether the places $\mathcal{SR}_1, \mathcal{SR}_2$, and $\mathcal{SR}_3$ are marked with (at least) two tokens, thus indicating that both conditions are fulfilled. Recall that the probability of (eventually) reaching a condition is represented as a place being (eventually) marked. Note that a place eventually marked does not necessary mean a place eventually always marked.

These nets can finally be merged with the PN of the safety scenarios depicted in Figure 9. Both nets are merged using the transitions that create tokens in places representing the same issue, i.e., places $tractionOn$ and $doorStatusOPENING$ in Figure 10 represent the same state than $p\_traction\_on\_TRUE$ and $p\_door\_CLOSING$, respectively, in Figure 9. The connection to places representing safety contracts have been highlighted (grey colour) in Figure 9. Places in Figure 10 starting with NOT are connected to places representing the complementary issue, but using a test arc instead of a normal arc to express the negation concept.

The merged PN is finally used with the GreatSPN tool [26] to compute the probability of places SR_1, SR_2, SR_3 having a marking greater than zero (i.e., the place is eventually marked). When this situation occurs, it indicates that the OCL constraints `TCMS_SR1`, `DoorController_SR2` and `TCMS_SR3` are fulfilled. A single run execution of the net is performed, and returns positive values for these probabilities, thus safety contracts are fulfilled in the system model. Let us finally remark that final effort must be focused on assuring that the system implementation matches the UML models. Otherwise, although a safety verification of models have been proved, the system may reach unsafe states.

Code 1. OCL constraints obtained from SCF transformation.

```
context TCMS_SR1
  inv: door_enabled and  not traction_on
                implies doorStatus = OPENING

context DoorController_SR2
  inv: obstacle
                implies doorStatus = OPENING

context TCMS_SR3
  inv: door_enabled and close_door
                implies doorStatus = IS_CLOSED
```

Note that the UML models that we described here are enriched with MARTE profile annotations, but these enriched data are not used for the safety analysis. However, these data can be necessary for verifying some safety properties where timing become relevant [39]. To this aim, we may use OCL/RT [40], an extension of native OCL to specify time issues, in conjunction with the MARTE profile, and translate such an information into the GSPN models. We consider this an interesting issue which deserves further study.

5. Related Work

Several methodologies have been proposed for the verification of safety properties on critical systems, see [41; 42]. Both [41; 42] propose similar methodologies to the one we propose although both of them also include a code generation step.

Our methodology extends the performance analysis methodology presented in [20], which is based on principles and techniques of Software Performance Engineering (SPE) [21]. Concerning methodologies based on SPE principles, to the best of our knowledge, there are very few initiatives and all of them are focussed on performance analysis. For instance, the PASA (Performance Assessment of Software Architectures) method, proposed by [43] is a performance scenario-based software architecture analysis method that provides a framework for the whole assessment process. Nevertheless, some steps of PASA entrust in the software engineer expertise to be applied and to identify alternatives for improvements. [44] defined Continuous Performance Assessment of Software Architecture (CPASA). This method adapts PASA to the agile development process. Unlike our proposal, these methodologies only analyse performance issues.

Regarding contracts, many formalisms have been proposed to express contracts, such as the Requirements Specification Language (RSL) [5], the Othello language [6], which is based on Linear Temporal Logic, or Modal Transmission Systems [45]. Unlike OCL, these languages are more expressive but OCL is a well-known language among modelisation engineering community. However, a major drawback of these formalisms is that the requirement engineers need to learn a new formalism each time they need to write contracts in a specific domain. In contrast, OCL is a well-known language in industry. Besides, to the best of our knowledge some of the proposed formalisms lack the means to verify that a component model fulfils their contracts [5; 45], or only focus on verification of functional properties [6]. In this work, we have shown that OCL contracts can be used to perform safety assessment by translating the UML models to Petri nets. Although currently we also focus on functional properties, the use of UML profiles enables to analyse other non-functional properties that can affect to safety, such as performance, dependability or security.

Representing safety contracts using OCL has been previously proposed in [39]. The novelty of our work is that we propose a translation from safety contracts in the form of assumptions and guarantees to OCL. Our work complements the work of OTHELLO language [6] and OCRA [46]. In particular, the analysis of non-functional properties can complement the work on verifying functional properties in OCRA [46]. Other work similar to ours is [47], where UML/OCL is used to express system invariants, transformed to Place/Transition nets (without time) and to LTL logic for

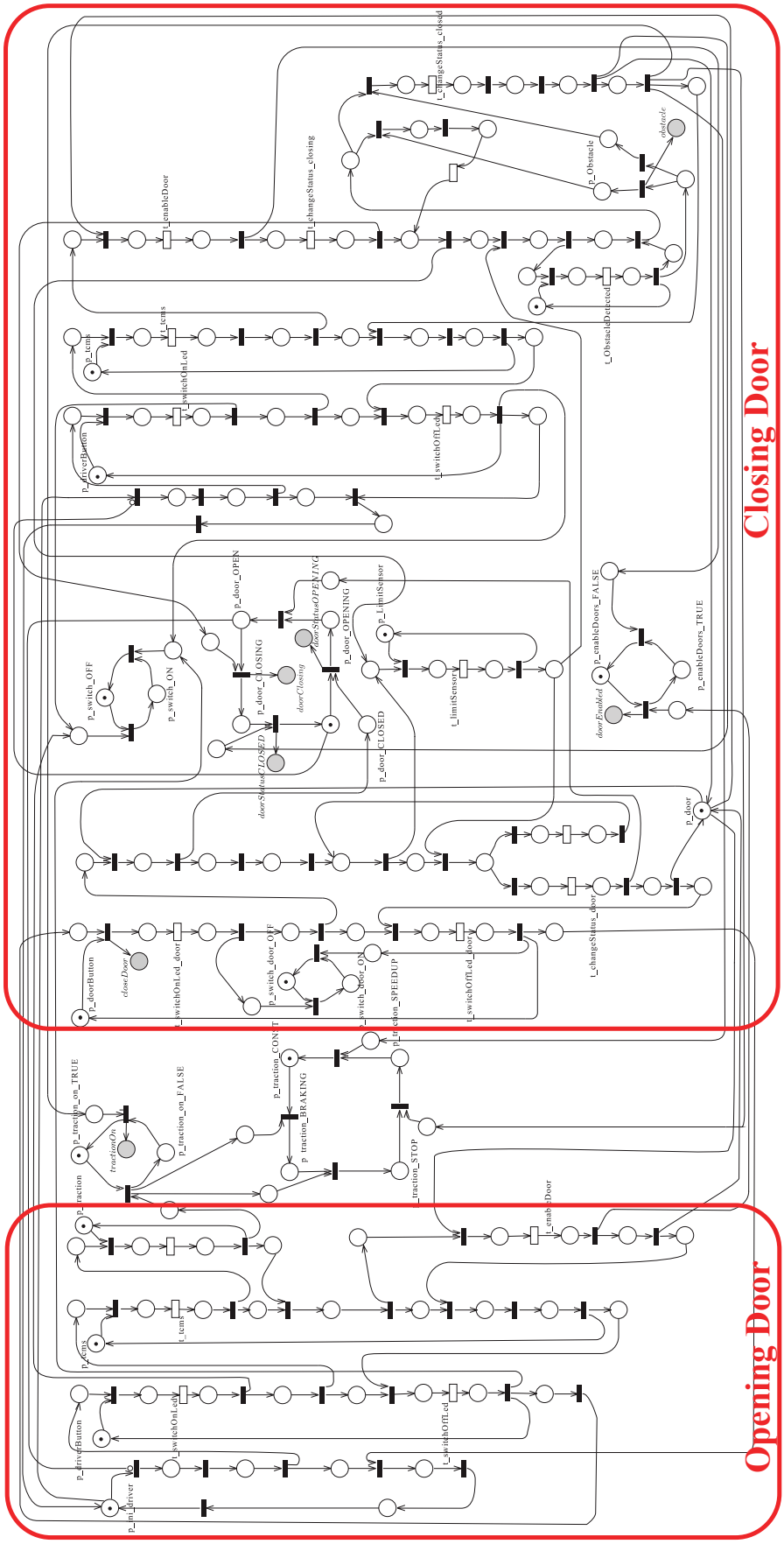
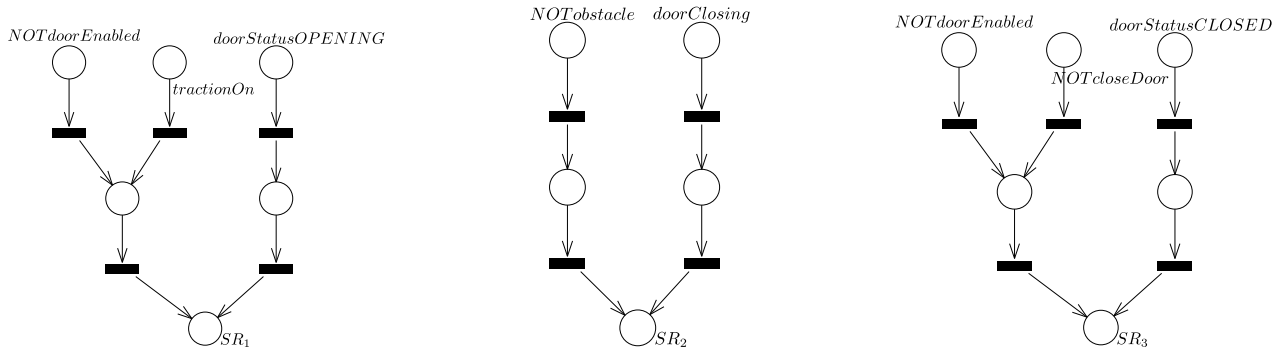


Fig. 9. Petri net corresponding to the opening and closing of a door.



(a) OCL constraint `TCMS_SR1` (b) OCL constraint `DoorController_SR2` (c) OCL constraint `TCMS_SR3`

Fig. 10. Petri net representation of OCL constraints of the case study.

the verification. In contrast to their work, we formalise the safety contracts, and, moreover, our Petri net models capture the timing information.

Some works refine safety contract assumptions in strong and weak assumptions [5; 48]. Strong assumptions specify what always is fulfilled by the environment, context-independently, while weak assumptions provide additional information about the context where a component could operate (e.g., the expected timing between input signals). In this paper, we consider the definition of safety contract as given in [25], having only strong assumptions. In our case, the weak assumptions can be implicitly described by UML annotations. As future work, we aim at extending our safety contract specification to explicitly express timing issues.

6. Conclusions and Future Work

Safety-critical systems need to assure that safety requirements are fulfilled before they are deployed. Safety assessment normally relies on methods applied during the normal operation of the system, detecting design problems that lead to a non-compliance with the system safety requirements. Thus, these systems are redesigned, normally inducing a huge cost overhead since the development process has to be is redone. To alleviate this problem, safety assessment should be performed in early stages of the development lifecycle.

To this aim, in this paper we present a model-based methodology for the safety assessment of critical systems. The methodology consists of three phases: Safety-oriented design, Safety-oriented specification, and Safety-oriented analysis. In the safety-oriented design phase, a component-based UML model is sketched using Composite Structure Diagrams and Sequence Diagrams. These diagrams capture the structure and behaviour of the software systems to be analysed. In the safety-oriented specification phase, safety requirements are first expressed as Safety Contracts Fragments, and then translated into OCL. Finally, in the safety-oriented analysis phase, the aforementioned UML models enriched with OCL constraints obtained from the previous phase are translated into a formal model (in particular, Generalized Stochastic Petri nets), where the analysis is carried out. The proposed methodology is evaluated with a real industrial case study from the railway domain, namely, a train door controller system.

Our methodology is an extension of the methodology for performance analysis presented in [20]. The advantage of extending this existing methodology is clear, since performance and safety can be assessed using a similar approach.

The specification of safety contracts in terms of OCL within UML models allows to express safety requirements and system characteristics in a single picture. The adoption of formal models in early stages of development lifecycle, obtained after the transformation of UML/OCL models to Petri nets, ensures an early verification of fulfilment of safety requirements. Besides, this early adoption becomes also easy since UML/OCL are modelling languages familiar to the industry engineers community. This approach complements other approaches where more expressive languages than OCL, for instance, Linear Temporal Logic (LTL), are used to specify safety contracts. In our approach, we sacrifice expressiveness in favour of simplicity in the contracts specification language. However, this issue can be overcome by

extending OCL with more operators, for instance, temporal operators.

Another interesting aspect that deserves further study is the analysis of safety contracts where concrete time values are considered, for instance, "a door is closed within 5 seconds when the door opening is enabled and the close event is received". To allow the analysis of such time constraints, we would need to extend the safety analysis phase of our proposed methodology. Specifically, we would need to provide the translation process from OCL constraints into Petri nets with time intervals.

Acknowledgements

The research leading to these results has received funding from the ARTEMIS Joint Undertaking under grant agreement n^o 295373 (project nSafeCer) from National funding, from the Spanish National Institute of Cybersecurity (INCIBE) accordingly to the rule 19 of the Digital Confidence Plan (Digital Agency of Spain), from the University of León under the contract X43, from the Spanish MINECO project TIN2012-39391-C04-02, *STRONGSOFT* and from nGreens (S2013/ICE-2731) from the Madrid Regional Government .

References

[1] nSafeCer project. Safety Certification of Software-Intensive Systems with Reusable Components. Project Grant Agreement n^o 295373;. http://safecer.eu/, (2015, accessed 15 February 2015).

[2] Lutz RR. Software Engineering for Safety: A Roadmap. In: Proceedings of the Conference on The Future of Software Engineering. ICSE '00. New York, NY, USA: ACM; 2000. p. 213–226.

[3] Grottke M, Trivedi KS. Fighting Bugs: Remove, Retry, Replicate, and Rejuvenate. Computer. 2007 February;40(2):107–109.

[4] Feiler PH. Model-Based Validation of Safety-Critical Embedded Systems. In: IEEE Aerospace Conference (AERO); 2010. p. 1–10.

[5] Damm W, Hungar H, Josko B, Peikenkamp T, Stierand I. Using Contract-based Component Specifications for Virtual Integration Testing and Architecture Design. In: Proceedings of the Design, Automation Test in Europe Conference Exhibition (DATE); 2011. p. 1–6.

[6] Cimatti A, Tonetta S. A Property-Based Proof System for Contract-Based Design. In: Proceedings of the 38th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA); 2012. p. 21–28.

[7] Kath O, Schreiner R, Favaro J. Safety, Security, and Software Reuse: A Model-Based Approach. In: Proceedings of the Fourth International Workshop in Software Reuse and Safety; 2009. .

[8] OMG. Unified Modeling Language (UML); 2011. Version 2.4.1, http://www.omg.org/spec/UML/2.4.1/, accessed 10 February 2015.

[9] OMG. Object Constraint Language (OCL); 2010. V2.2, formal/2010-02-01.

[10] OMG. UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms (QoS & FT); 2008. Version 1.1, http://www.omg.org/spec/QFTP/, accessed 9 February 2015.

[11] Rodríguez RJ, Gómez-Martínez E. Model-based Safety Assessment using OCL and Petri Nets. In: Proceedings of the 40th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA); 2014. p. 56 – 59.

[12] Murata T. Petri Nets: Properties, Analysis and Applications. In: Proceedings of the IEEE. vol. 77; 1989. p. 541–580.

[13] Gómez-Martínez E, Rodríguez RJ, Etxeberria L, Illarramendi M, Benac C. Model-Based Verification of Safety Contracts. In: Canal C, Idani A, editors. Software Engineering and Formal Methods - SEFM 2014 Collocated Workshops: HOFM, SAFOME, OpenCert, MoKMaSD, WS-FMDS, Revised Selected Papers. vol. 8938 of Lecture Notes in Computer Science. Springer; 2014. p. 101–115.

[14] International Organization for Standardization. ISO/IEC 19505-1: Information technology – Object Management Group Unified Modeling Language (OMG UML) – Part 1: Infrastructure; 2012.

[15] OMG. A UML profile for Modeling and Analysis of Real Time Embedded Systems (MARTE); 2011, http://www.omgmarte.org/, accessed 4 February 2015. Version 1.1.

[16] Bernardi S, Merseguer J, Petriu DC. Dependability modeling and analysis of software systems specified with UML. ACM Comput Surv. 2012;45(1):2.

[17] Rodríguez RJ, Merseguer J, Bernardi S. Modelling and Analysing Resilience As a Security Issue Within UML. In: Proceedings of the 2nd International Workshop on Software Engineering for Resilient Systems. SERENE '10. New York, NY, USA: ACM; 2010. p. 42–51.

[18] Ajmone Marsan M, Balbo G, Conte G, Donatelli S, Franceschinis G. Modelling with Generalized Stochastic Petri Nets. Wiley Series in Parallel Computing. John Wiley and Sons; 1995.

[19] Bernardi S, Merseguer J. Performance evaluation of UML designs with Stochastic Well-formed Nets. Journal of Systems and Software. 2007 November;80(11):1843–1865.

[20] Gómez-Martínez E, González-Cabero R, Merseguer J. Performance assessment of an architecture with adaptative interfaces for people with special needs. Empirical Software Engineering. 2014;19(6):1967–2018.

[21] Smith CU. Increasing Information Systems Productivity by Software Performance Engineering. In: Proc. 7th Int. Conf. Computer Measurement Group (CMG'81); 1981. p. 5–14.

[22] Sljivo I, Gallina B, Carlson J, Hansson H, Puri S. A Method to Generate Reusable Safety Case Fragments from Compositional Safety Analysis. In: Schaefer I, Stamelos I, editors. Software Reuse for Dynamic Systems in the Cloud and Beyond - 14th International Conference on Software Reuse, ICSR 2015, Miami, USA, January 4-6, 2015. vol. 8919 of Lecture Notes in Computer Science. Springer; 2015. p. 253–268.

[23] OMG. Systems Modeling Language (SysML); 2012. Version 1.3, http://www.omgsysml.org/, accessed 13 February 2015.

[24] Sangiovanni-Vincentelli A, Damm W, Passerone R. Taming Dr. Frankenstein: Contract-based design for cyber-physical systems. European Journal of Control. 2012;18(3):217–238.

[25] Söderberg A, Johansson R. Safety Contract Based Design of Software Components. In: IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW); 2013. p. 365–370.

[26] Baarir S, Beccuti M, Cerotti D, De Pierro M, Donatelli S, Franceschinis G. The GreatSPN tool: recent enhancements. SIGMETRICS Perform Eval Rev. 2009;36(4):4–9.

[27] Zimmermann A. Modeling and evaluation of stochastic Petri nets with TimeNET 4.1. In: Proceedings of the 6th International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS); 2012. p. 54–63.

[28] Rodríguez RJ, Júlvez J, Merseguer J. PeabraiN: A PIPE Extension for Performance Estimation and Resource Optimisation. In: Proceedings of the 12th International Conference on Application of Concurrency to System Designs (ACSD). IEEE; 2012. p. 142–147.

[29] Gómez-Martínez E, Merseguer J. ArgoSPE: Model-based Software Performance Engineering. In: Proc. 27th Int. Conf. on Applications and Theory of Petri Nets and Other Models of Concurrency (ICATPN'06). vol. 4024 of Lecture Notes in Computer Science. Springer; 2006. p. 401–410. Tool available at: http://argospe.tigris.org.

[30] López-Grao JP, Merseguer J, Campos J. From UML Activity Diagrams to Stochastic Petri Nets: Application to Software Performance Engineering. In: Proc. 4th Int. Workshop on Software and Performance (WOSP'04). ACM; 2004. p. 25–36.

[31] Liu TS, Chiou SB. The application of Petri nets to failure analysis. Reliab Eng Syst Safe. 1997;57(2):129–142.

[32] CENELEC. EN50126 Railway applications - The specification and demonstration of Reliability Availability Maintainability and Safety (RAMS); 1999.

[33] CENELEC. EN50128 Railway applications - Communications signalling and processing systems - Software for railway control and protection systems; 2001.

[34] CENELEC. EN50129 Railway applications - Communication signaling and processing systems - Safety related electronic systems for signaling; 2003.

[35] CENELEC. EN 50159 Railway applications - Communication, signalling and processing systems - Safety-related communication in transmission systems; 2010.

[36] CENELEC. EN 50121 Railway applications - Electromagnetic compatibility; 2006.

[37] CENELEC. EN 50125 Railway applications - Environmental conditions for equipment; 2003.

[38] International Electrotechnical Comission. IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems; 2010.

[39] Bate I, Hawkins R, McDermid J. A Contract-based Approach to Designing Safe Systems. In: Proceedings of the 8th Australian Workshop on Safety Critical Systems and Software - Volume 33. SCS '03. Australian Computer Society, Inc.; 2003. p. 25–36.

[40] Cengarle MV, Knapp A. Towards OCL/RT. In: FME 2002: Formal Methods – Getting IT Right. vol. 2391 of Lecture Notes in Computer Science. Springer Berlin Heidelberg; 2002. p. 390–409.

[41] Pérez ZAD. Model-driven development methodology for hybrid embedded systems based on UML with emphasis on safety-related requirements [PhD]. Faculty of Electrical Engineering and Computer Science, University of Kassel; 2014.

[42] Knorreck D, Apvrille L, de Saqui-Sannes P. TEPE: A SysML Language for Time-constrained Property Modeling and Formal Verification. SIGSOFT Softw Eng Notes. 2011 Jan;36(1):1–8.

[43] Williams LG, Smith CU. PASA^{SM}: A Method for the Performance

Assessment of Software Architectures. In: Proc. 3rd Int. Workshop on Software and Performance (WOSP'02). ACM; 2002. p. 179–188.

[44] Pooley RJ, Abdullatif AAL. CPASA: Continuous Performance Assessment of Software Architecture. In: Proc. 17th IEEE Int. Conf. and Workshops on the Eng of Computer-Based Systems (ECBS'10). IEEE Computer Society; 2010. p. 79–87.

[45] Bauer SS, David A, Hennicker R, Guldstrand Larsen K, Legay A, Nyman U, et al. Moving from Specifications to Contracts in Component-Based Design. In: Lara J, Zisman A, editors. Proceedings of the 15th International Conference on Fundamental Approaches to Software Engineering (FASE). vol. 7212 of Lecture Notes in Computer Science. Springer Berlin Heidelberg; 2012. p. 43–58.

[46] Cimatti A, Dorigatti M, Tonetta S. OCRA: A tool for checking the refinement of temporal contracts. In: 28th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE; 2013. p. 702–705.

[47] Bouabana-Tebibel T, Belmesk M. Integration of the Association Ends within UML State Diagrams. Int Arab J Inf Technol. 2008;5(1):7–15.

[48] Sljivo I, Gallina B, Carlson J, Hansson H. Strong and Weak Contract Formalism for Third-Party Component Reuse. In: IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW); 2013. p. 359–364.