Model-Based Verification of Safety Contracts

Elena Gómez-Martínez[1], Ricardo J. Rodríguez[1(✉)], Leire Etxeberria Elorza[2], Miren Illarramendi Rezabal[2], and Clara Benac Earle[1]

[1] Babel Group, Universidad Politécnica de Madrid, Madrid, Spain
{egomez,rjrodriguez,cbenac}@babel.ls.fi.upm.es
[2] Embedded Systems Research Group, Mondragon Goi Eskola Politeknikoa (MGEP), Arrasate-Mondragón, Spain
{letxeberria,millarramendi}@mondragon.edu

Abstract. The verification of safety becomes crucial in critical systems where human lives depend on the correct functioning of such systems. 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. Safety requirements are usually 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 the use of well-known modelling languages, such as UML, to model a software system, and the use of OCL to express the system safety contracts within UML. A UML model enriched with OCL constraints is then transformed to a Petri net model that enables to formally verify such safety contracts. We apply our approach to an industrial case study that models a train doors controller in charge of the opening and closing of train doors. Our approach allows to perform an early safety verification, which increases the confidence of software engineers while designing the system.

Keywords: Safety contracts · Model-based · Verification · Petri nets

1 Introduction

With the growing adoption of software in safety-critical systems, safety assessment has become a crucial software engineering task as it has been recognised by several initiatives, for instance, the ARTEMIS JU nSafeCer project [1]. Moreover, software system safety engineering must be incorporated early in the software design process and be part of the development and operational lifecycle of the system.

Contract-based design is a popular approach for the design of complex component-based systems where safety properties are difficult to guarantee [2,3]. A key benefit of using contracts is that they follow the principle of separation

The research leading to these results has received funding from the ARTEMIS Joint Undertaking under grant agreement n^o 295373 (project nSafeCer) and from National funding.

C. Canal and A. Idani (Eds.): SEFM 2014 Workshops, LNCS 8938, pp. 101–115, 2015.
DOI: 10.1007/978-3-319-15201-1_7

of concerns [4], separating assumptions that the environment of a component obeys from what a component guarantees under such an environment.

The Unified Modelling Language (UML) [5] 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) [6] which is a well-known language among modelisation engineering community, or (ii) to use specific UML profiles [7]. In previous work [8], we have proposed a technique that combines both approaches. In this paper, in contrast, we focus on the representation of safety contracts as OCL constraints.

For the verification of safety contracts, several formal verification techniques have been proposed, for instance [3], which uses model checking. Our proposal is to translate UML to Petri Nets and perform the analysis by computing probabilities using the GreatSPN tool [9]. By combining standard engineering practice, i.e., UML, with formal verification techniques, i.e. Petri nets, we provide a rigorous safety analysis available for software engineers.

Our approach has been used to verify a set of safety contracts on an industrial case study where the UML model of a train doors controller has been analysed. The train doors controller is the component in charge of opening and closing train doors. The CAF Power & Automation company[1] develops these train components. Thus, components like the train doors controller are modelled in UML previous to their implementation.

In summary, the contributions of the work presented in this paper are the following:

- a formal definition of the proposed transformation of a safety contract into an OCL constraint.
- an (informal) transformation of OCL constraints into Petri nets by means of the case-study.
- a (partly automatic/partly manual) translation of the case-study UML diagrams annotated with OCL to Petri Nets.
- the safety analysis of the case-sudy.

The rest of the paper contains the following sections. Firstly, Sect. 2 outlines the basic concepts. Section 3 details the train doors controller. Then, Sect. 4 describes a proposal of safety contract specification in OCL, and its transformation to Petri nets. It also introduces the safety contracts of the case study, which are analysed in Sect. 5. Finally, Sect. 6 covers related work and Sect. 7 states some conclusions.

2 Previous Concepts

UML [5,10] is a semi formal general-purpose visual modelling language used for specifying software systems. UML can be tailored for specific purposes by

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

profiling. 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 [11] provides support for schedulability and performance analysis in real-time and embedded systems, while DAM (Dependability Analysis and Modelling) profile [12] supports dependability analysis and SecAM (Security Analysis and Modelling) profile [13] focuses on security aspects. In this paper, we use the MARTE profile to indicate the duration of activities in a UML model. The stereotype provided by MARTE to this goal is `gaStep` (`hostDemand` tagged value), within the MARTE analysis framework called Generic Quantitative Analysis Model (GQAM).

Another extension to enrich UML semantics is the Object Constraint Language (OCL) [6]. OCL is a pure expression language for describing constraints that apply to UML models. 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, formal methods may help. In this paper, we consider Petri nets [14] as the formal modelling language. More precisely, we translate the annotated UML diagrams into Generalized Stochastic Petri Nets (GSPNs [15]), following the guidelines proposed in [16].

A GSPN is a graphical and mathematical formalism used for the modelling of concurrent and distributed systems. A gentle introduction to GSPN can be found in [15]. Informally, a GSPN is a bipartite graph of places and transitions joined by arcs (graphically represented by circles, bars and arrows respectively). 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 Case Study: Train Doors Controller

As a case study in this paper, we consider the door control management performed by a Train Control and Monitoring System (TCMS). The TCMS is a complex distributed (along the train) system that controls many subsystems. It contains several Input/Output (IO) modules that gather data and send it to a PLC (Programmable Logic Controller) via a communication bus. Each of the IO modules has a CPU, digital/analogical inputs and outputs and is connected to the communication bus. The logic of the TCMS is performed in the PLC.

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

- the `TCMS` component that decides whether to enable or disable the doors. Doors must be enabled before they can be opened and disabled before closing;
- the `Door` component that effectively controls the opening or closing of a door;
- the `Traction` component that controls the train movement; and
- the `MVB` (Multifunction Vehicle Bus) component that communicates the components among them.

Figure 1 shows the composite diagram of the system. The subcomponents of the `Door` component, i.e., the controller (in the following we will refer to this component as the `Door Controller`), the limit sensors, the obstacle sensor, and a button for opening doors, are also depicted in the diagram.

In this paper, we focus on the control of doors. The case study presented here concerns a real system where some simplifications have been made. Namely, the interaction with other components of the TCMS and the dependencies with other subcomponents, and their communication has been omitted. Besides, concerning the closing of doors, in the original design there were different versions of the existence of obstacles, while here we have chosen only one of them.

In the following, we present the UML Sequence Diagrams (UML-SD) for the opening and closing of doors. Figure 2(a) depicts the UML-SD for opening the door. When a train driver requests the opening of doors, first the TCMS checks whether the train status is suitable for opening the doors without risk, checking that the train is really stopped before sending the "enable door" order to the `Door Controller` component. Thus, the TCMS system sends the "enable door" command request to the `Door Controller` component only when the train is in a safe condition (e.g. speed is zero) to perform the request properly and without risk for passengers. The Door Controller component opens the door only if it is enabled, i.e., it has received the "enable door" order from the TCMS and if some passenger has request the opening of a door ("open request") using any of the buttons (interior or exterior) of the door.

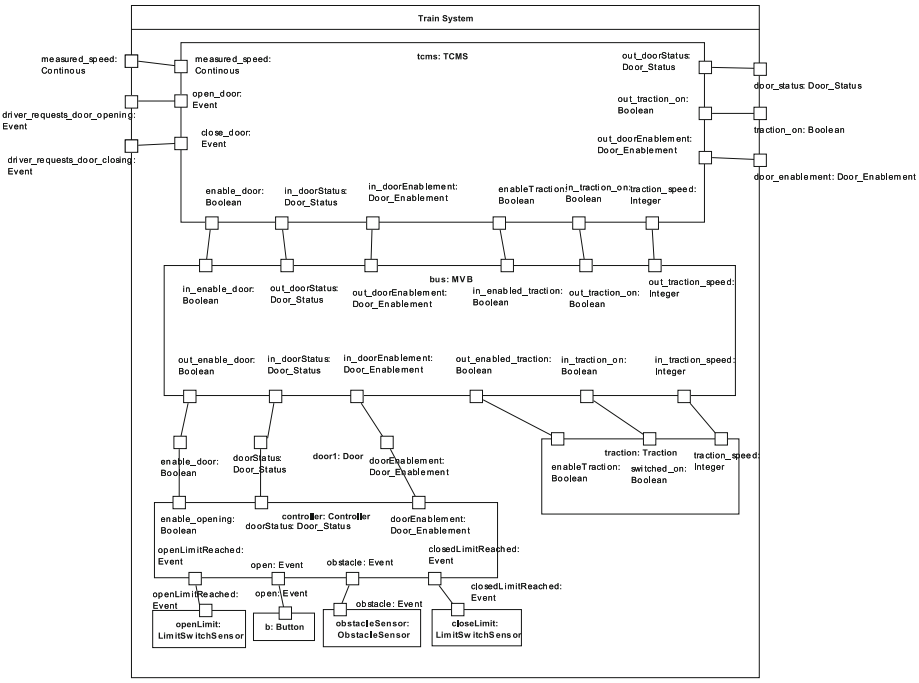


Fig. 1. The Composite diagram of the system.

The door closing operation is depicted in Fig. 2(b). When the driver commands doors closing, the TCMS system sends the "not enable door" command to the `Door` component. The `Door` component disables the door and closes the door if it is safe, i.e. there is no detected obstacle. When there is an obstacle, the door is opened and closed once such an obstacle has disappeared.

In order to enable an incremental certification process and to demonstrate the benefits of reusability, this case study adopts the methodology of contract-based design. In contract-based design each safety critical component of the system and non-critical components are seen as separate components [17] which interact with their environment. As we formally explain in the next section, we associate to each safety critical component $\mathcal{C}$ a safety contract, i.e. an abstract specification in the form of a tuple $\mathcal{S}_\mathcal{C} = \langle \mathcal{A}, \mathcal{G} \rangle$, where $\mathcal{A}$ represents the assumptions on the environment of the component, and $\mathcal{G}$ represents what the component guarantees under these assumptions. A contract is intended to expose enough information about the component, but not more than necessary. We say that a component *implements* its contract if it satisfies the guarantees when the environment meets the assumptions.

In the following section we introduce a framework for safety contract specification and the transformation to OCL constraints, which will be later used for formal safety assessment using Petri nets.

4 Specification of Safety Contracts as OCL and Petri Nets

In a component-based system a contract defines the obligations to be met by a certain component and its dependencies [18]. As it is claimed in [19], a safety contract is similar to a (software) contract but instead of pre/post-conditions contains assumptions and guarantees that endorse a certain level of integrity of functional properties depending on the component's environment.

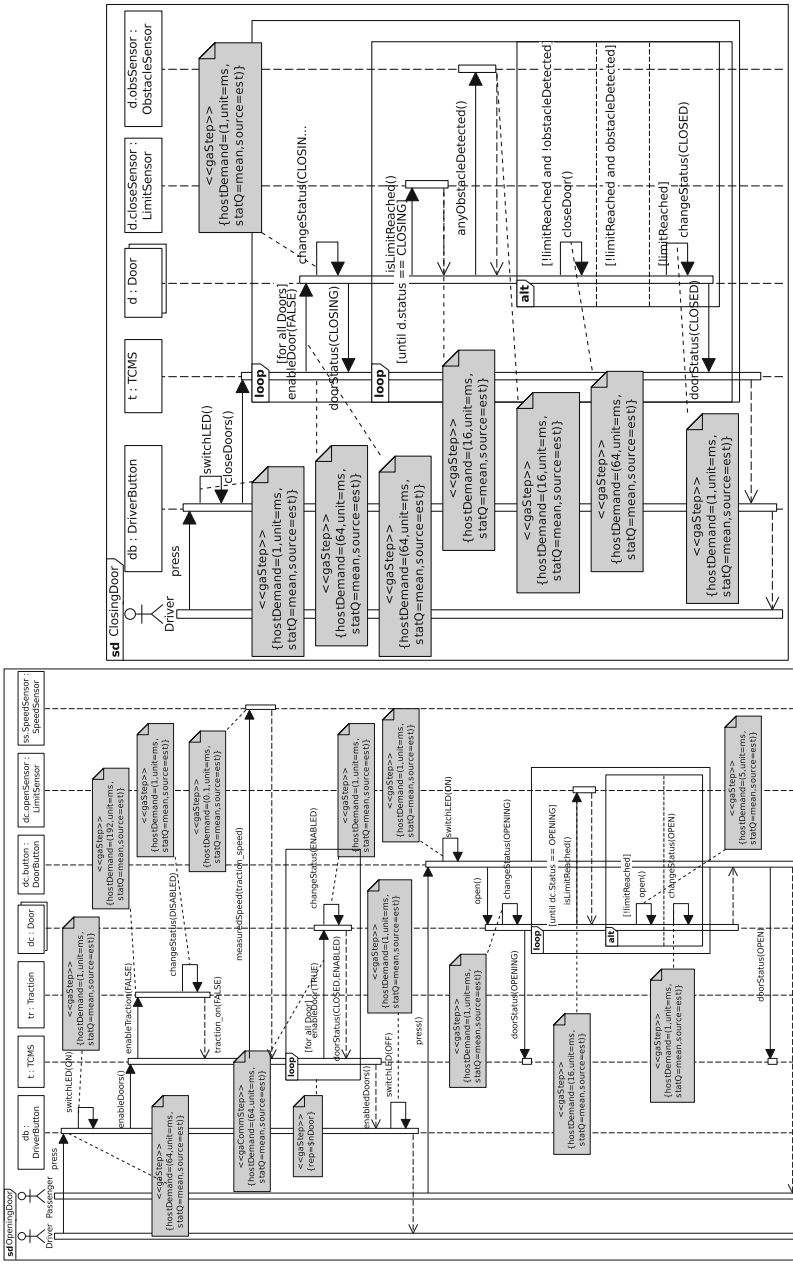
In this paper, we adhere to the definition of a Safety Contract Fragment (SCF) given in [19]. A SCF conforms a safety contract as a set of assumptions – what it is expected to be met by the component's environment – and a set of guarantees, which specify the behaviour of a component under such an environment. In a previous work we have explored the idea of transforming an SCF to an OCL invariant within UML models [8]. In this work, we revise and formalise our model-based transformation approach. In the sequel, we formally define a SCF and the transformation from an SCF to an OCL invariant.

Let us assume a system composed of a set of components that interact between them. Let $\mathcal{C} = \langle \mathcal{I}, \mathcal{O} \rangle$ be a component of such a system having a set $\mathcal{I}$ of input ports and a set $\mathcal{O}$ of output ports. Let $\mathcal{S}_\mathcal{C} = \langle \mathcal{A}, \mathcal{G} \rangle$ be a SCF [19] defined over a component $\mathcal{C}$, where $\mathcal{A} = \mathcal{A}^+ \bigcup \mathcal{A}^*$ is a superset of disjoint sets $\mathcal{A}^+, \mathcal{A}^*$ of OR and AND safety constraints, respectively, and $\mathcal{G} = \mathcal{G}^+ \bigcup \mathcal{G}^*$ is a superset of disjoint sets $\mathcal{G}^+, \mathcal{G}^*$ of OR and AND guarantees[2]. A safety contract assumption $\mathcal{A}$ is a proposition that relates one or more of the input ports of a component. Similarly, a safety contract guarantee $\mathcal{G}$ is a proposition that relates one or more of the output ports of a component.

Recall that OCL is a UML extension to express constraints into UML models. An OCL constraint is defined over a context that describes where such a constraint is acting. As it is introduced in Sect. 2, OCL defines different constructs, such as `inv` to define invariants, which state conditions that must always be met by all instances of a context type, `pre` to state a condition that must be true when an operation starts its execution, or `post` to state a condition that must be true when an operation ends its execution. In this paper, we consider only OCL invariants. An OCL constraint can be formally defined as follows. Let $\mathcal{R} = \langle \mathcal{X}, \mathcal{V} \rangle$ be an OCL constraint defined over a context $\mathcal{X}$ and having an invariant formula $\mathcal{V} = \langle ls, rs \rangle$. An invariant formula is conformed by two propositions ls, rs joined by a boolean or `implies` operator. Note that the right-hand side of an invariant formula can be empty.

As it has been previously mentioned, an OCL constraint is defined over a context that describes where such a constraint is acting. In the proposed translation, a SCF corresponds to an OCL constraint. Since a SCF is specified over a component, it is reasonable to match the context of the corresponding OCL constraint to such a component as well. Thus, a transformation from SCF to OCL invariant can be straightforwardly defined as follows:

[2] As in [8], we restrict the logic of SCF assumptions and guarantees to AND and OR logic operators.



(a) Door opening UML-SD

(b) Door closing UML-SD

Fig. 2. (a) Door opening and (b) door closing UML sequence diagrams.

Proposition 1. *Let $\mathcal{C}$ be a component of a system on which a Safety Contract Fragment $\mathcal{S} = \langle \mathcal{A}, \mathcal{G} \rangle$ has been defined. Thus, an OCL $\mathcal{R} = \langle \mathcal{X}, \mathcal{V} \rangle$ can be built considering $\mathcal{X} = \mathcal{C}$ and $\mathcal{V} = \langle \mathcal{A}, \mathcal{G} \rangle$.*

As it can be seen, the component $\mathcal{C}$ defines the context $\mathcal{X}$ of the OCL constraint, while the content of such an OCL constraint (the invariant) is defined by the assumptions and guarantees of the Safety Contract Fragment $\mathcal{S}$ defined on $\mathcal{C}$.

Let us describe how our transformation approach works by means of the case study described in Sect. 3. Consider the following safety requirements given by the engineers designing the system:

SR1. The door opening is not enabled when the traction is on or the train speed is distinct than zero.
SR2. The door must be closed but remains open when some obstacle has been detected.
SR3. The door is closed when the door opening is enabled and the close event is received.

The above safety requirements can be expressed in terms of Safety Contract Fragments, considering the component-based system depicted in Fig. 1, as follows:

- $\mathcal{S}_1 = \langle (traction\ OR\ (tractionSpeed \neq 0)), (NOT\ enableOpening) \rangle$, defined on the `TCMS` component.
- $\mathcal{S}_2 = \langle obstacle, doorStatus = opening \rangle$. In this case, the component on which this SCF is defined is `DoorController`.
- $\mathcal{S}_3 = \langle (enableOpening\ AND\ close), doorStatus = isClosed \rangle$. This SCF is defined on the component `Door`.

Note that the assumptions and guarantees of the former SCFs relate, respectively, input and output ports of the components where they are defined.

Following the Proposition (1), the above SCFs can be straightforwardly converted to OCL invariants as it is listed in Code 1.1. Here, the task of a requirement engineer is to interpret the safety requirements in terms of SFC. This task is accomplished by matching the safety requirements to the UML component-based design. This task is surely a difficult one but once this task has been performed the transformation to OCL invariants becomes trivial. Recall that these OCL invariants that express safety requirements allow to perform safety assessment in a system, as shown in the following section.

Code 1.1. OCL constraints obtained from SCF transformation.

```
context TCMS_SR1
   inv: (traction or tractionSpeed <> 0)
                implies not enableOpening

context DoorController_SR2
   inv: obstacle
```

```
                implies (doorStatus = opening)

context Door_SR3
   inv: (enableOpening and close)
                implies doorStatus = isClosed
```

Let us show how this OCL invariants can be transformed to Petri nets. Note that we use only the `implies` binary operator ($\rightarrow$) within the OCL invariant. Recall that in classical logic the `implies` binary operator can be transformed to an equivalent form using `or` and `not` operators, i.e., $p \rightarrow q$ is logically equivalent to $\neg p \vee q$. If we consider each proposition of OCL invariant as Petri net places, and transform the invariant to its logically equivalent, we obtain the Petri net models depicted in Fig. 3 for each safety contracts considered for the case under study[3].

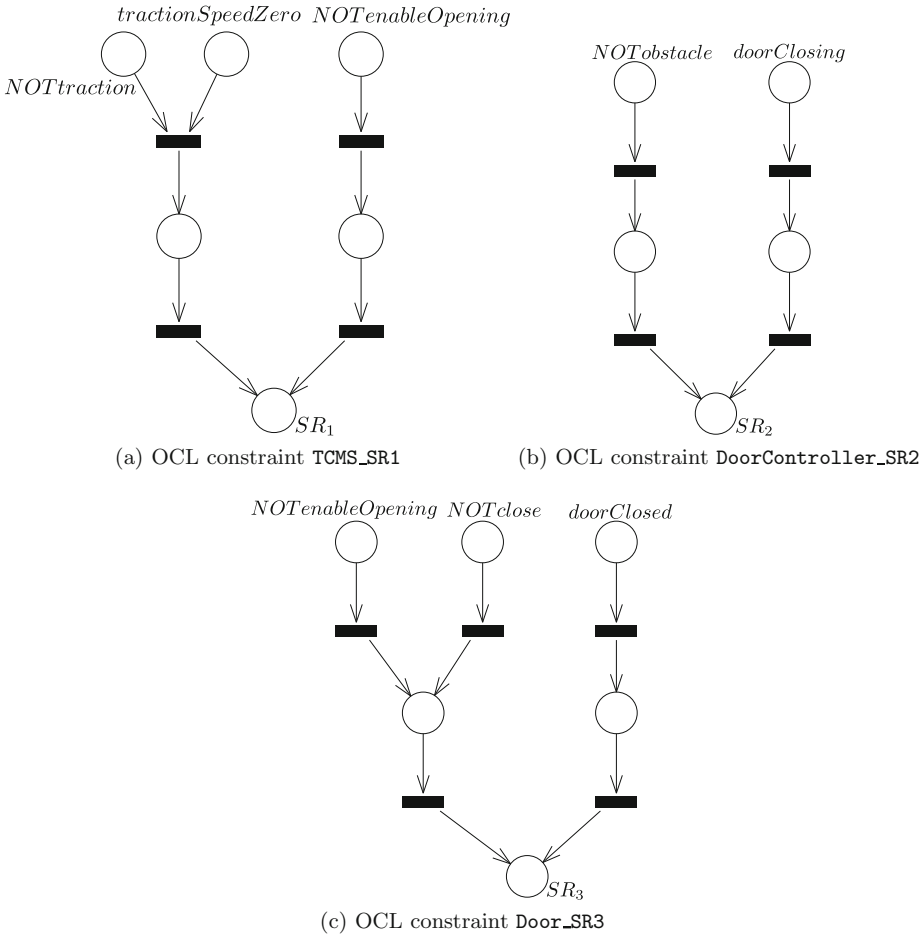
The sink places (without output transitions) of each Petri net representation depicted in Fig. 3(a), (b) and (c) allow us to compute the probability of having a marking in such a place (post-condition) greater than zero, indicating that preconditions are fulfilled. Note that this solution does not provide us with information regarding the event order or any other kind of temporal information. This is an interesting issue that deserves further study, as discussed in the following section.

5 Safety Analysis

We describe the safety analysis we propose by means of the case study. In order to analyse the safety scenarios, i.e. the opening and closing of doors, the corresponding UML-SD diagrams annotated with OCL, respectively depicted in Fig. 2(a) and (b), are translated into GSPNs using the ArgoSPE tool [20] according to the algorithms proposed in [16]. The resulting GSPN is shown in Fig. 4. The left-hand side of the figure represents the door opening and the left-hand side, the door closing. Even though part of the translation is done automatically using the ArgoSPE tool some simple manual modifications to the GSPN are needed to represent OCL constraints. In particular, modifying this GSPN with the GreatSPN [9] tool, we have manually modelled the obstacle detection event as a place, named *p_Obstacle*, since it has associated an OCL constraint, as we explain in the following paragraph. Moreover, we have 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.

Since the OCL constraints are interpreted in a GSPN, they are equivalent to compute the probability of a condition. Each condition is represented by a place of the GSPN. For instance, the place *p_door_OPEN* represents the status in which a door is open and the place *p_switch_ON* represents when the door button is switched on. The probability of (eventually) reaching a condition is

[3] I. Sljivo, personal communication, April 1, 2014.



(a) OCL constraint `TCMS_SR1`

(b) OCL constraint `DoorController_SR2`

(c) OCL constraint `Door_SR3`

Fig. 3. Petri net representation of OCL constraints of the case study.

represented as a place being (eventually) marked. Note that a place eventually marked does not necessary mean a place eventually always marked.

The Petri nets representing the safety contracts, depicted in Fig. 3 can now be composed with the Petri net of the system depicted in Fig. 4. Both nets are merged using the transitions that create tokens in places representing the same issue, i.e., places $NOTtraction$ and $tractionSpeedZero$ in Fig. 3 represent the same state than $p\_traction\_on\_FALSE$ and $p\_traction\_STOP$, respectively, in Fig. 4. The connection to places representing safety contracts have been highlighted (grey colour) in Fig. 4.

Finally, we use the GreatSPN tool [9] to compute the steady-state probability of places SR_1, SR_2, SR_3 having a marking greater than zero (i.e. the place is eventually marked), which will indicate that the OCL constraints `TCMS_SR1`,

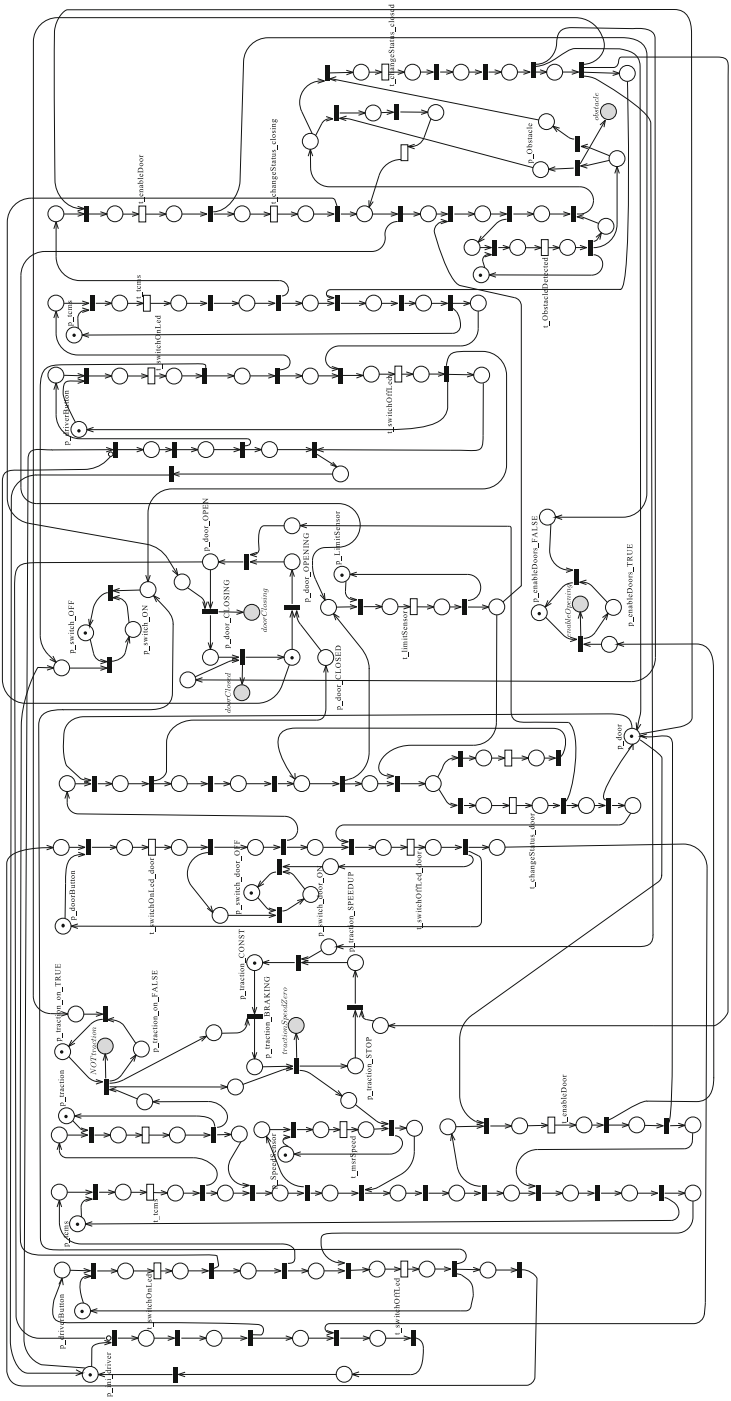


Fig. 4. Petri net corresponding to the opening and closing of a door.

`DoorController_SR2` and `Door_SR3` are fulfilled. A simulation of the net with GreatSPN returns a positive value for these probabilities, thus safety contracts are fulfilled in the system model.

Although the UML models that we use are enriched with MARTE profile annotations, we do not currently use such an information for the safety analysis even though it can be necessary for verifying some safety properties [18]. For this aim, we may use OCL/RT [21], 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.

6 Related Work

Many formalisms have been proposed to express contracts, such as the Requirements Specification Language (RSL) [2], the Othello language [3], which is based on Linear Temporal Logic, or Modal Transmission Systems [22]. 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 [2,22], or only focus on verification of functional properties [3]. 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 [18]. 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 [3] and OCRA [23]. In particular, the analysis of non-functional properties can complement the work on verifying functional properties in OCRA [23]. Other work similar to ours is [24], where UML/OCL is used to express system invariants, transformed to Place/Transition nets (without time) and to LTL logic for 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 [2,25]. 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 [19], 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.

7 Conclusions and Future Work

Safety assessment is a crucial software engineering activity in critical systems, since people integrity, and even their lives may depend on it. In the last years, contract-based design has emerged as a promising approach for designing safe systems, where contracts describe the expected behaviour of a component.

In this paper, we propose a specification of safety contracts as assumptions and guarantees based on the input and output ports of a component, and then translate these contracts to OCL in the UML context. Finally, these UML models are transformed into a formal model, in terms of Generalized Stochastic Petri nets (GSPN), to verify that safety contracts are fulfilled. As a case study, we have analysed three safety contracts on a train door controller designed by CAF Power & Automation. The most challenging tasks regarding the case study were the formalisation of safety contracts and the translation of UML models to GSPN. In the latter, although some automation exists, the complexity of some aspects of the case study (for instance, the existence of obstacles) required a manual translation to GSPN.

The specification of safety contracts in terms of OCL within UML models allows to recap safety requirements and system description in a single picture. Besides, the adoption of formal models, obtained after the transformation of UML/OCL models to Petri nets, are facilitated as UML/OCL are languages familiar to the industry engineers. The result is that we have sacrificed expression power to keep safety contracts expressed with OCL easier to understand than contracts written in more expressive languages like, for instance, Linear Temporal Logic (LTL). This issue can be overcome in the future by extending the native OCL with more operators.

As for further work, our aim is to keep on formalising more complex contracts expressed in OCL, as well as exploring how to provide the event order or any other kind of temporal information (or other non-functional property). Improving the automatic translation from the UML models to GSPN deserves also further study. In addition, we also plan to propose a well-established methodology to assess safety and to develop a tool that implements this methodology.

References

1. nSafeCer project: Safety Certification of Software-Intensive Systems with Reusable Components. Project Grant Agreement n^o 295373. More information at: http://safecer.eu/
2. 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), pp. 1–6, March 2011

3. 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), pp. 21–28, September 2012
4. 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)
5. OMG: Unified Modeling Language (UML). Version 2.4.1, August 2011. Specification available at: http://www.omg.org/spec/UML/2.4.1/
6. OMG: Object Constraint Language (OCL). Object Management Group, v2.2, formal/2010-02-01, February 2010
7. OMG: UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms (QoS & FT). Version 1.1 (2008). Specification available at: http://www.omg.org/spec/QFTP/
8. Rodríguez, R.J., 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), pp. 56–59 (2014)
9. Baarir, S., Beccuti, M., Cerotti, D., De Pierro, M., Donatelli, S., Franceschinis, G.: The GreatSPN tool: recent enhancements. SIGMETRICS Perform. Eval. Rev. **36**(4), 4–9 (2009)
10. ISO/IEC: 19505–1:2012-Information technology-Object Management Group Unified Modeling Language (OMG UML)-Part 1: Infrastructure (2012)
11. OMG: A UML profile for Modeling and Analysis of Real Time Embedded Systems (MARTE). Version 1.1 (2011). Specification available at: http://www.omgmarte.org/
12. Bernardi, S., Merseguer, J., Petriu, D.C.: Dependability modeling and analysis of software systems specified with UML. ACM Comput. Surv. **45**(1), 2 (2012)
13. Rodríguez, R.J., 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 2010, pp. 42–51. ACM, New York (2010)
14. Murata, T.: Petri Nets: properties, analysis and applications. Proc. IEEE **77**(4), 541–580 (1989)
15. Ajmone Marsan, M., Balbo, G., Conte, G., Donatelli, S., Franceschinis, G.: Modelling with Generalized Stochastic Petri Nets. John Wiley Series in Parallel Computing, Chichester (1995)
16. Bernardi, S., Merseguer, J.: Performance evaluation of UML design with Stochastic Well-formed Nets. J. Syst. Softw. **80**(11), 1843–1865 (2007)
17. Sangiovanni-Vincentelli, A., Damm, W., Passerone, R.: Taming Dr. Frankenstein: contract-based design for cyber-physical systems. Eur. J. Control **18**(3), 217–238 (2012)
18. 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, SCS 2003, vol. 33, pp. 25–36. Australian Computer Society, Inc. (2003)
19. Söderberg, A., Johansson, R.: Safety contract based design of software components. In: IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pp. 365–370 (2013)
20. Gómez-Martínez, E., Merseguer, J.: ArgoSPE: model-based software performance engineering. In: Donatelli, S., Thiagarajan, P.S. (eds.) ICATPN 2006. LNCS, vol. 4024, pp. 401–410. Springer, Heidelberg (2006)

21. Cengarle, M.V., Knapp, A.: Towards OCL/RT. In: Eriksson, L.-H., Lindsay, P.A. (eds.) FME 2002. LNCS, vol. 2391, pp. 390–409. Springer, Heidelberg (2002)
22. Bauer, S.S., David, A., Hennicker, R., Guldstrand Larsen, K., Legay, A., Nyman, U., Wąsowski, A.: Moving from specifications to contracts in component-based design. In: de Lara, J., Zisman, A. (eds.) FASE 2012. LNCS, vol. 7212, pp. 43–58. Springer, Heidelberg (2012)
23. 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), pp. 702–705. IEEE (2013)
24. Bouabana-Tebibel, T., Belmesk, M.: Integration of the association ends within UML state diagrams. Int. Arab. J. Inf. Technol. **5**(1), 7–15 (2008)
25. 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), pp. 359–364, November 2013