# A Case Study in Formal Analysis of System Requirements

Dimitri Belli[1][0000−0003−1491−6450] and Franco Mazzanti[1][0000−0003−4562−8777]

Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo" CNR, Via G. Moruzzi 1, 56124 Pisa, Italy, {dimitri.belli, franco.mazzanti}@isti.cnr.it

**Abstract.** One of the goals of the 4SECURail project has been to demonstrate the benefits, limits, and costs of introducing formal methods in the system requirements definition process. This has been done, on an experimental basis, by applying a specific set of tools and methodologies to a case study from the railway sector. The paper describes the approach adopted in the project and some considerations resulting from the experience.

**Keywords:** Critical Systems of Systems · Formal Methods · Standard Interfaces · Systems Modeling Language · Railway Signaling System.

## 1 Introduction

The railway infrastructure is constituted by a large, heterogeneous, and distributed system with components that are on board, trackside, centralized, crossing regional and national borders, managed by different authorities, and developed by different providers. Not surprisingly, the current trend is to standardize the requirements of the various system components together with their interfaces (see, e.g. EULYNX[1]). Standardization is expected to increase market competition, reduce vendor lock-in, and promote the reduction of long-term maintenance costs. However, to produce the desired outcomes, the defined standard requirements for the various system components must be precise, i.e., not suffer from ambiguous interpretation issues, and correct, i.e., not give rise to interoperability problems and not suffer of inconsistencies or missing points. The current state of the art is based on the use of natural language requirements possibly associated with SysML/UML graphical artifacts [2–6]. Such a choice is not risk-free because natural language and SysML/UML are usually not rigorous enough to allow a precise system specification [7, 8]. One of the goals of the 4SECURail [9] project is to observe the impact of the integration of formal methods inside the requirements definition process. This has been achieved with the definition of a "Demonstrator", i.e., an example of requirements construction process based on formal methods, and its application to a case study selected from the railway signaling sector[12].

## 2 The Case Study and Demonstration Process

The 4SECURail case study is derived from the communication layers specified by UNISIG-39 [10] and UNISIG-98 [11], describing the establishment, supervision, and management of the RBC[1]-RBC communication line used to support the RBC-Handover protocol. The full system can be modeled as a set of four UML state machines interacting with other three state machines modeling other parts of the execution environment (see Fig. 1). In our modeling, we introduced an additional abstract "Timer" component that allows the various components to proceed in parallel but in a constrained way with respect to their relative execution speed. The requirements of the Communication Supervision Layer

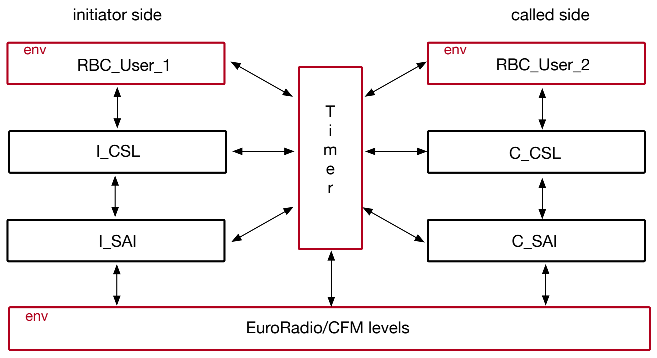


**Fig. 1.** The 4SECURail case study structure

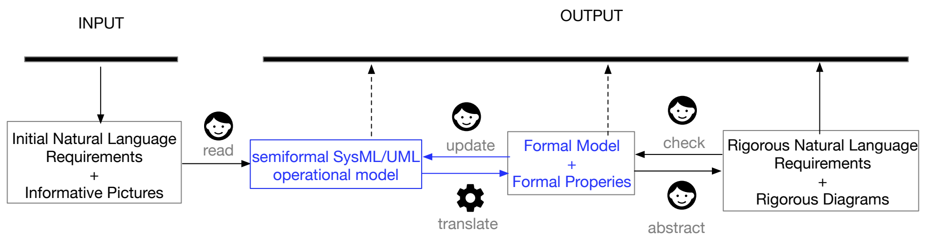

INPUT
OUTPUT
Initial Natural Language Requirements + Informative Pictures
read
semiformal SysML/UML operational model
update
translate
Formal Model + Formal Properies
check
abstract
Rigorous Natural Language Requirements + Rigorous Diagrams

**Fig. 2.** The 4SECURail Demonstrator process

(CSL) and Safe Application Intermediate Sub-Layer (SAI) components are defined in natural language, and their initial specification can be found in Deliverable D2.3[9]. The 4SECURail demonstrator process (see Fig. 2) begins with the analysis of the natural language descriptions of the requirements and with the construction of an operational SysML/UML model of the system components.

[1] Radio Block Centre

The UML designs are complemented by an explicit and precise set of assumptions on the characteristics of inter-state machine communications. We make a restricted use of the features provided by UML so that the design has a clear and simple semantics allowing, with a low effort, its mechanical translation into the different notations used for formal analysis. This paper focuses on the presentation of the adopted approach for the translation of the SysML/UML models into the formal notations supported by three different verification frameworks, namely UMC [13–15], ProB [16, 17], and CADP [18, 19]. For all the details of the formalization and analysis process, we refer to the project deliverables [9].

## 3 The Formal Modeling

The first notation used to model the case study is the KandISTI/UMC framework developed by the Formal Methods && Tools (FMT) Laboratory[2] at ISTI-CNR in Pisa. This notation allows us to define a system as a set of UML state machines, expressed in a simple textual form[3], to explore the possible system evolutions, and to verify branching time properties on it. Despite its still prototypical status, this framework has been chosen as the first target since it fits well the needs of fast design prototyping. The resulting graph describing the system evolutions can be analyzed or saved in the form of a Doubly Labeled Transition System (L2TS), where the user has the choice to specify which kind of information should be associated with the L2TS edges and nodes. This information may include the UMC transition label, the outgoing events generated by the effects of a transition, the value of some state variables, or any other custom flag associated with the transition firing. The second notation is the B language accepted by the ProB tool. ProB is an animator, constraint solver, and model checker for the B-Method developed by the Institute for Software and Programming Languages of the Heinrich-Heine University in Germany. The B-method-based tool appears to be one of the most widely used tools for the formal development and analysis of railway-related systems [26]. The third notation is the LNT[27] language of the CADP[18] framework. CADP is an advanced process algebra-based toolset that leverages Labeled Transition Systems (LTS) theory to support compositional verification, system minimization, animation, and testing. The LNT notation has an imperative style of process descriptions that is well-suit to the description of the behavior of UML state machines.

In UMC, a system is defined as a static instantiation of a set of state machines from their template defined as a Class definition. The event pool associated with a state machine can be qualified as FIFO or RANDOM queue, and in our case, we rely on the UML FIFO default choice. The behavior of a UMC state machine is described by a set of rules in the form:

```
Transition_Label:
SourceStates -> TargetStates {Trigger [Guard] / Effects}
```

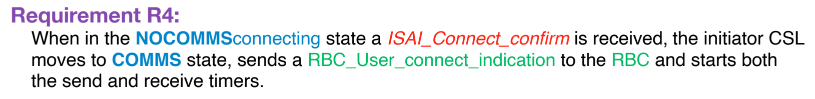
[2] https://fmt.isti.cnr.it

[3] UMC is freely accessible online at http://fmt.isti.cnr.it/umc and a detailed description of the syntax can be found in http://fmt.isti.cnr.it/umc/DOCS/sdhelp.html

In ProB, our encoding models a system as a single B Machine that includes the local state and the behavior of all the UML state machines constituting the system. The main difference between the ProB model and the UMC/LNT models is that in ProB, the UML event pools are modeled by global variables manipulated by the (atomic) state machine operations, while in UMC and LNT the event pools are handled locally inside each state machine, and their manipulation occurs via synchronizations or message exchange. A UML transition of a state machine is mapped on a ProB *OPERATION*, appropriately conditioned with respect to the trigger and guard, and performing the specified effects. The sending of an event is explicitly modeled with the insertion of data into a FIFO buffer modeling the event pool of the target state machine.

In the LNT encoding, a state machine is represented by an LNT process, and the various LNT processes are composed in parallel, appropriately synchronizing the sending/accepting actions. Each process executes a loop inside which several alternatives are non-deterministically possible. These alternatives model either the condition and effects of the triggering of state machine transitions, or the unconditioned acceptance in the event pool of incoming events. Also in this case, the event pool of the state machine is explicitly modeled as a FIFO buffer in the local state of the process.

Figure 3 shows one of the natural language requirements for the initiator CSL subsystem, while Figure 4 shows the graphical layout of the state machine diagram of the CSL system component on the initiator side of the communication line. We can see how the requirement *R4* is modeled by the corresponding transition in the state machine diagram. Fig. 5 shows, from left to right, the encoding of the R4 transition for UMC, ProB, and LNT. Clearly, the executable model contains more implementation details than the abstract UML design shown in Fig. 4, which just describes the system requirements in a semi-formal notation acting as a bridge between the natural language and the executable/formal notations. The colors in the figure help to see the matching of the various information present in each encoding. We can see that the transition label in UMC becomes the operation name in ProB, that the change of state is modeled in ProB and LNT by the change of the value of a variable, and that signaling-related operations are modeled in ProB and LNT as explicit operations on lists/tuples. An essential consequence of using a UML subset (e.g., no composite states, no parallel states, no deferred events, no competition between triggered and completion transitions) is that it becomes rather easy to implement a mechanical translation from the UMC encoding to the ProB and LNT notations.


**Requirement R4:**

When in the **NOCOMMS**connecting state a *ISAI_Connect_confirm* is received, the initiator CSL moves to **COMMS** state, sends a RBC_User_connect_indication to the RBC and starts both the send and receive timers.

**Fig. 3.** The R4 requirement for the initiator CSL in natural language

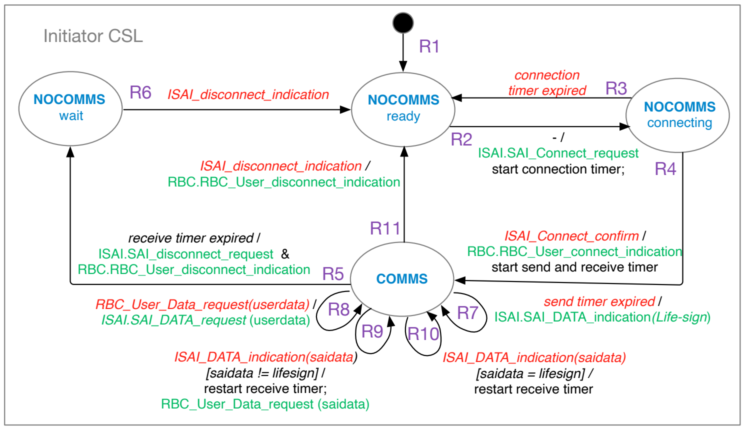

Initiator CSL
R1
NOCOMMS wait
R6
ISAI_disconnect_indication
NOCOMMS ready
connection timer expired
R3
NOCOMMS connecting
R2
- / ISAI.SAI_Connect_request start connection timer;
R4
ISAI_disconnect_indication / RBC.RBC_User_disconnect_indication
R11
receive timer expired / ISAI.SAI_disconnect_request & RBC.RBC_User_disconnect_indication
R5
COMMS
ISAI_Connect_confirm / RBC.RBC_User_connect_indication start send and receive timer
RBC_User_Data_request(userdata) / ISAI.SAI_DATA_request (userdata)
R8
R9
R10
R7
send timer expired / ISAI.SAI_DATA_indication(Life-sign)
ISAI_DATA_indication(saidata) [saidata != lifesign] / restart receive timer; RBC_User_Data_request (saidata)
ISAI_DATA_indication(saidata) [saidata = lifesign] / restart receive timer

**Fig. 4.** The state machine diagram of the CSL component on the initiator side

```
Class ICSL is
  ...
Behaviour
  ...
R4_ICSL:
NOCOMMSconnecting -> COMMS
 { ISAI_CONNECT_confirm /
    RBC.IRBC_User_Connect_indication;
    receive_timer := 0;
    connect_timer := max_connect_timer;
     send_timer := 0; }
  ...
end ICSL;
```

```
MACHINE SYS
  ...
OPERATIONS
  ...
R4_ICSL =
PRE
    ICSL_buff /= [] &
    first(ICSL_buff) = ISAI_CONNECT_confirm &
    ICSL_STATE = NOCOMMSconnecting
THEN
    IRBC_buff := IRBC_buff <-
                  RBC_User_Connect_indication;
    ICSL_connect_timer :=
                  ICSL_max_connect_timer;
    ICSL_receive_timer := 0;
    ICSL_send_timer := 0;
    ICSL_buff := tail(ICSL_buff);
    ICSL_STATE = COMMS
END;
  ...
END;
```

```
process ICSL [..] is
...
 var mybuff: ICSL_BUFF, ... in
  loop
   select
    -- R4_ICSL
    only if
       mybuff /= nil
    and
         head(mybuff) = ISAI_CONNECT_confirm
     and
         STATE = NOCOMMSconnecting
    then
      RBC_User_Connect_indication;
      connect_timer := max_connect_timer;
      receive_timer := 0;
      send_timer := 0;
      mybuff := tail(mybuff);
      STATE = COMMS
    end if
    []
      ...
   end select
  end loop
 end var
end process
```

**Fig. 5.** UMC, ProB, and LNT encoding of the R4 ICSL transition

All these three notations, moreover, natively support data type operations on lists or tuples that can be exploited in an equivalent way to handle FIFO buffer operations. The final effect of the transformations is the generation of formal models with almost the same readability as the first UMC model; also, the original comments present in the UMC code are preserved in the generated ProB and LNT encodings. Because of the strict budget and timing constraints of the project, our goal has been limited to the translation of the set of features currently used in our models. Still, the set of supported features can surely be further extended (e.g., by allowing sequential composite states and constrained forms of parallel states).

The CADP environment allows saving the statespace of an LNT model in the simple textual *.aut*[20] format (as an LTS whose labels denote communication actions). The ProB tool saves the full statespace of a model in textual format which can be easily mechanically converted into the *.aut* format (as an LTS whose labels denote the triggered operation names). Finally, also the UMC environment allows saving the statespace of a model in the *.aut* format, permitting the user to specify which information to encode in the LTS labels (communication actions or transition labels, or both). The strong equivalence of the three models can therefore be easily checked with tools like mCRL2 *ltscompare*[29] or CADP *bcg_cmp*[21]. While defects in the code of the translators can often immediately be put in evidence by just the observation of the size of the generated state spaces, the formal LTS comparison of the *.aut* representations allows observing also one of the specific execution traces that are at the root of the dissimilarity. This proved to be very useful during the testing of our translators.

## 4 Hints on the Formal Analysis

The tool diversity adopted in the project allows us to analyze the system from different perspectives: e.g., state-based linear time properties with ProB, event-based branching time properties with CADP, state- and event-based properties with UMC, information hiding and model reductions with CADP. Because of the parametricity of the system and the presence of several wide-range parameters in communications, formal analysis can only be done by reasoning on selected scenarios where the system parameters are fixed and the environment components have a desired stimulating behavior. Several examples of these scenarios are shown in [30] and described in Deliverable D2.5[9]. Linear (or lazo-shaped) counterexamples or reachability proofs from UMC and ProB can be displayed in a friendly way as sequence diagrams. Due to the complexity of the issue, for more details on the subject, we refer to the final project deliverable D2.5 [9] and the presentations in [22–24].

## 5 Related Works

The goal of the 4SECURail Demonstrator is to show a possible way to improve the quality of standard specifications by exploiting formal methods. The project Formasig[28] has a very similar goal, which is the development of a formal method allowing railway standardization projects to formally verify standardized interfaces. Also in the case of Formasig, the starting point is the EULYNX natural language specification enriched with SysML artifacts. The Formasig approach aims to translate these EULYNX SysML models, developed with the commercial PTC framework[33], into the process specification language mCRL2[36, 37] for formal analysis. Several other Shift2Rail[34] projects have investigated the use of formal methods for the analysis of signaling systems like ASTRAIL[35], which focuses on a survey of the available tools on this subject, and PERFORMINGRAIL[25] (still in progress) more centered on ERTMS[46] moving block specifications. The impact of the adoption of formal methods during railway-related software development has been studied in Shift2Rail projects X2RAIL2 and

X2RAIL5 (in progress). Unfortunately, not all the produced material in these last projects is publicly available. Many studies investigate the formal verifications of UML models (e.g. [49–54]). Because of the ambiguity, variability, and complexity of the OMG UML documents, all these efforts appear as particular personal interpretations of specific UML subsets, without reaching the goals of providing UML with precise and widely recognized semantics. We have focused our effort on the model checking techniques provided by ProB for Event-B specifications. An alternative approach, based on theorem proving to develop formally verified refinements, is supported by Atelier-B[38] and Rodin [39]. When using Rodin the input models can also be derived by UML-B[40–42] designs. A fragment of our case study, i.e., the SAI communications levels, has also been specified and verified, as a spin-off of the project[43], using UPPAAL[44]. Hugo[47, 48] is another interesting example of formal methods diversity that still uses UML state machines as a starting point while exploiting UPPAAL and Spin[45] for formal analysis.

## 6 Conclusions

The effort described in this short paper is just a fragment of the overall activity performed inside the project and does not describe many other points analyzed or discussed in the project deliverables. Among these, an analysis of the cost and benefits from the point of view of Infrastructure Managers for the use of formal methods, the reasons for choosing UMC, Prob, and LNT as reference platforms, the reasons and difficulties implied by the choice of using UML as starting point of the analysis process, the relation between the natural language requirements and the semi-formal and formal artifacts, the kind of easily understandable feedback that the formal analysis can give to the initial standard interface designer. Some of these themes have also been touched in [22–24]. The experience gained in the experimentation has confirmed that a simplified version of UML is a viable choice for the modeling of requirements. A simplified UML can be the base for rigorous, clear, and easy to understand designs that can be mapped more directly with natural language requirements, and that can be translated into still understandable formal notations. A second confirmation coming from our experimentation is that the exploitation of formal methods diversity, i.e., multiple translations of the same specification into different formal notations, allows from one side to reduce and detect as early as possible the introduction of encoding errors, and from the other side the widening of the available formal analysis techniques and tools. The project deliverables, the generated models, the verified scenarios, and the source code of the translators are publicly available from Zenodo repositories [30–32].

**Acknowledgements** This work has been partially funded by the 4SECURail project (Shift2Rail GA 881775). The content of this paper reflects only the author's view and the Shift2Rail Joint Undertaking is not responsible for any use that may be made of the included information.

## References

1. The Eulynx initiative https://eulynx.eu/
2. OMG: Unified Modeling Language, Version 2.5.1 (OMG UML). Dec. 2017 https://www.omg.org/spec/UML/2.5.1
3. OMG: Precise Semantics of UML State Machines (PSSM). May 2019 https://www.omg.org/spec/PSSM/1.0
4. OMG: System Modeling Language version 1.6. Dec. 2019 https://www.omg.org/spec/SysML/1.6
5. OMG Semantics of a Foundational Subset for Executable UML Models (fUML). https://www.omg.org/spec/FUML/1.4
6. OMG: Action Language for Foundational UML (Alf). https://www.omg.org/spec/ALF/1.1
7. Broy,M., Cengarle,M.V.: UML formal semantics: Lessons learned. Software & Systems Modeling, volume 10, pp. 441–446. (2011)
8. Fecher H., Schönborn J. et al.: 29 New Unclarities in the semantics of UML 2.0 State Machines In: Formal Methods and Software Engineering. LNCS 3785. Springer.
9. 4SECURail project Deliverables , https://www.4securail.eu/Documents.html
10. SUBSET, UNISIG. 039, FIS for the RBC/RBC Handover. (2015)
11. SUBSET, UNISIG. 098, RBC-RBC Safe Communication Interface. (2012)
12. 4SECURail: project Deliverable D2.1 https://www.4securail.eu/pdf/4SR-WP2-D2.1-Specification
13. Ter Beek,M. H., Fantechi,A., Gnesi,S., Mazzanti,F.: A state/event-based model-checking approach for the analysis of abstract system properties. Science of Computer Programming, volume 76(2), pp. 119–135. (2011)
14. ter Beek,M., Gnesi,S., Mazzanti,F.: From EU Projects to a Family of Model Checkers. In Software, Services, and Systems, LNCS 8950, (2015) Springer.
15. ter Beek,M.H., Fäntechi,A., Gnesi,S., Mazzanti,F.: States and events in KandISTI. In Models, Mindsets, Meta: The What, the How, and the Why Not?, volume 11200, pp. 110–128. Springer, Cham. (2019)
16. Leuschel,M., Butler,M.: ProB: an automated analysis toolset for the B method. STTT volume 10(2), pp. 185–203. (2008)
17. ProB project home page: https://prob.hhu.de/
18. Garavel,H., Lang,F., Mateescu,R. and Serwe,W.: CADP 2011: a toolbox for the construction and analysis of distributed processes. STTT Volume 15(2), pp. 89–107. (2013)
19. CADP Web site https://cadp.inria.fr/
20. https://cadp.inria.fr/man/aut.html (manual page)
21. https://cadp.inria.fr/man/bcg_cmp.html (manual page)
22. Mazzanti,F., Belli,D.: Formal Modelling and Initial Analysis of the 4SECURail Case Study, in Proc.of 5th Workshop on Models for Formal Analysis of Real Systems, MARS 2022, EPTCS 355, 2022, pp. 118-144
23. Mazzanti,F. and Belli,D.: The 4SECURail Formal Methods Demonstrator. In: The 4th International Conference on Reliability, Safety and Security of Railway Systems (RSSRAIL), LNCS 13294, Springer, 2022.
24. Belli,D., Fantechi,A. et al.: The 4SECURail approach to formalizing standard interfaces between signalling systems components. Transport Research Arena (TRA) Conference 2022 (to appear).
25. Project PerformingRail https://cordis.europa.eu/project/id/101015416

26. ter Beek,M.H., Borälv,A., Fantechi,A., Ferrari,A., Gnesi,S., Löfving,C., Mazzanti,F.: Adopting formal methods in an industrial setting: the railways case. In International Symposium on Formal Methods, volume 11800, pp. 762-772. Springer, Cham. (2019)
27. Champelovier,D., Clerc,X.,et al.: Reference Manual of the LOTOS NT to LOTOS Translator (Version 5.8). Technical report, INRIA/VASY and INRIA/CONVECS. (2013)
28. Formasig home page:
https://www.utwente.nl/en/eemcs/fmt/research/projects/formasig
29. https://www.mcrl2.org/web/user_manual/tools/release/ltscompare.html
30. https://zenodo.org/record/6322392 (formal models)
31. https://zenodo.org/record/5807738 (workstream 1 deliverables)
32. https://zenodo.org/record/5541350 (translation tools)
33. https://www.ptc.com/en/products/windchill/expert-packages
34. Shift2Rail (now Europe'srail) https://rail-research.europa.eu/
35. Project AstRail https://cordis.europa.eu/project/id/777561
36. Groote,J.F., Keiren,J.J.A., Luttik,B.,de Vink,E.P.,Willemse,T.A.C.: Modelling and Analysing Software in mCRL2. FACS 2019, LNCS vol. 12018, pp. 25-48.
37. mCRL2 Web site https://www.mcrl2.org/
38. Atelier B Web site https://www.clearsy.com/outils/atelier-b/
39. Abrial, JR., Butler, M., Hallerstede, S. et al. Rodin: an open toolset for modelling and reasoning in Event-B. Int J Softw Tools Technol Transfer 12, 447–466 (2010).
40. UML-B Web site https://www.uml-b.org/
41. Snook C., Savicks V., Butler M.: Verification of UML Models by Translation to UML-B. In: Formal Methods for Components and Objects. FMCO 2010. LNCS 6957. Springer, Berlin, Heidelberg (2010)
42. Dghaym,D., Dalvandi,M., Poppleton,M., Snook,C.F.: Formalising the Hybrid ERTMS Level 3 specification in iUML-B and Event-B. STTT 22(3)(2020)
43. Basile,D., Fantechi,A., Rosadi,I.: Formal Analysis of the UNISIG Safety Application Intermediate Sublayer. FMICS 2021, LNCS 12863, Springer.
44. Uppaal Wewb sites https://www.uppaal.com/, https://uppaal.org/
45. Spin Wen site https://spinroot.com/spin/whatispin.html
46. ERA - ERTMS Web sire https://www.era.europa.eu/activities/european-rail-traffic-management-system-ertms
47. https://www.uni-augsburg.de/en/fakultaet/fai/informatik/prof/swtsse/hugo-rt/
48. Knapp,A., Merz,S. and Rauh,C.: Model checking timed UML state machines and collaborations. Lecture Notes in Computer Science 2469, 2002.
49. Pétin,JF., Evrot,D., Morel,G., Lamy,P.: Combining SysML and formal models for safety requirements verification.
https://hal.archives-ouvertes.fr/hal-00533311/document
50. Liu S. et al.: A Formal Semantics for Complete UML State Machines with Communications. In: Integrated Formal Methods. IFM 2013. LNCS 7940. Springer.
51. Grumberg O., Meller Y., Yorav K.: Software Model Checking Techniques for Behavioral UML Models. In: Formal Methods. FM 2012. LNCS 7436. Springer.
52. Hvid Hansen H., Ketema J., Luttik B., Mousavi M., van de Pol J., dos Santos O.M.:Automated Verification of Executable UML Models. In: Formal Methods for Components and Objects. FMCO 2010. LNCS 6957. Springer.
53. Ober I., Graf S., Ober I.: Validation of UML Models via a Mapping to Communicating Extended Timed Automata. In: Model Checking Software. SPIN 2004. LNCS 2989. Springer Bouwman,M., Bas Luttik,B., van der Wal,D.:
54. Bouwman,M., Luttik,B., van der Wal,D,: A Formalisation of SysML State Machines in mCRL2, FORTE 2021, LNCS 12719