Int. J. ITS Res. (2018) 16:163–172
DOI 10.1007/s13177-017-0146-2

Modeling the CBTC Railway System of Siemens with *ScOLa*

Melissa Issad [1,2] · Leïla Kloul [3] · Antoine Rauzy [4] · Karim Berkani [2]

Received: 31 March 2016 / Revised: 13 December 2016 / Accepted: 18 July 2017 / Published online: 27 July 2017

Abstract Considering their increasing complexity, industrial systems are, in general, specified in a natural language. In railway systems, the design phase results an ambiguous and laborious system specification. The objective of this paper is to present *ScOLa*, a formal modeling language based on scenarios and built for railway system specifications. Its novelty resides in its restriction to a small set of concepts and its multiple representations (textual and graphical). The language offers means to understand what the system is supposed to do and to be as well as to support a dialog with experts so to be sure that they got everything correctly. The language is depicted on the railway automation solution Trainguard MT CBTC of Siemens.

Keywords System engineering · ScOLa · railway systems

1 Introduction

The CBTC (Communication Based Train Control) is based on the principle that trains determine their positions themselves and transmit it to wayside equipments. CBTC assures that the space between trains is always safe [1]. Given the complexity of these systems, their specifications are spread over thousands of pages written in a natural language, which makes it difficult for engineers to develop, validate and maintain.

In theory, according to the V-cycle (Fig. 1), complex systems must be defined in specifications, analyzed seeking for system acceptance and then developed and integrated and finally validated. But in practice, the system analysis phase is often let to the very last steps of the design which leads to a late detection of errors and ambiguities.

INCOSE [2] promotes multiple model-based system engineering (MBSE) methodologies. Its goal is to widen the use of models instead of documents in the system engineering process. INCOSE defines MBSE as the formalized application of modeling to support system requirements, design, analysis, verification and validation, beginning in the conceptual design phase and continuing throughout development and later life cycle phases. It also promotes the fact that modeling should be used at multiple levels of the system (operational, system and component). But often, modeling languages are used as a graphical notation of the specification (UML and SysML [3] are such modeling languages).

Mainly, two class of approaches for system modeling appeared. On the one hand, the language centric approaches, focusing on a specific modeling language, engineers will use all the items provided by the language to model the system. The result of these approaches is often redundant information. On the other hand, system centric approaches, where engineers modify the language to fit the system, results in a non-generic methodology for system modeling.

Moreover, the use of natural language in system specifications provides an ideal vehicle for eliciting user requirements and describing system functions. However, this method has major drawbacks. Because the freedom of expression leads to more freedom of interpretation, such descriptions provide a non-structured input for system modeling approaches.

✉ Melissa Issad
melissa.issad@ecp.fr

1 LGI, CentraleSupelec, Grande Voie des Vignes, 92290 Chatenay Malabry, France

2 Siemens Mobility, 150 avenue de la République, Châtillon 92320, France

3 DAVID, University of Versailles, 45 avenue des Etats Unis, Versailles 78000, France

4 IPK, NTNU, Høgskoleringen 1, Trondheim 7491, Norway

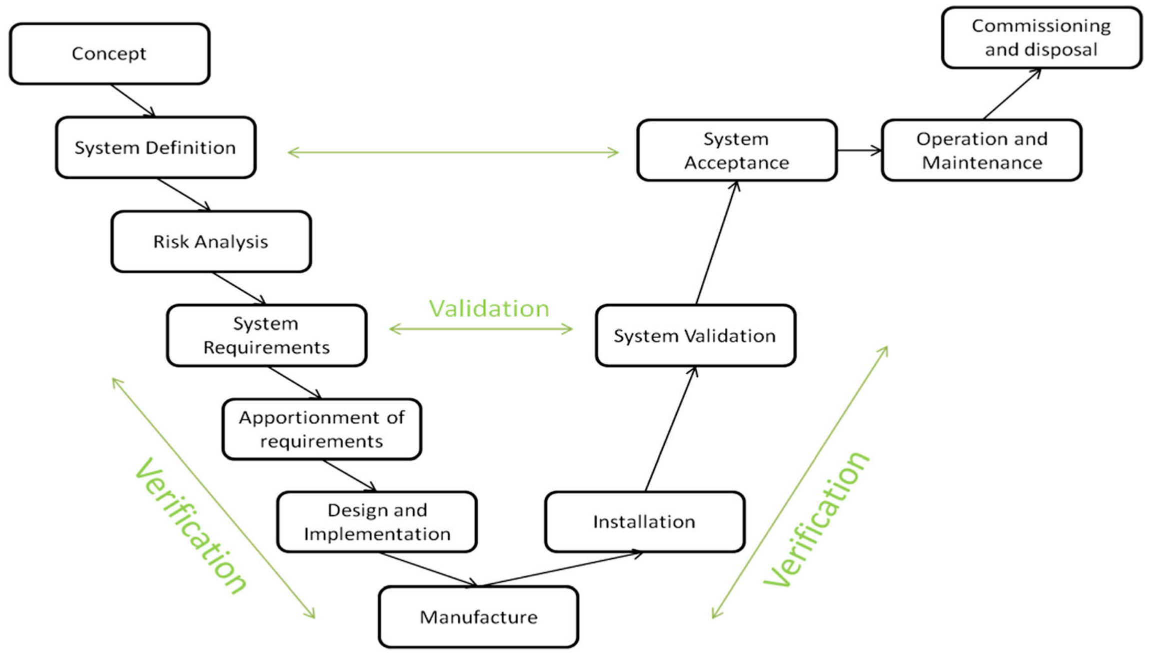


Fig. 1 Railway systems V-Cycle according to EN 50126

The purpose of the definition of *ScOLa* is to define a framework that helps engineers communicate around a unique and clear specification of the system. Therefore, we put scenarios in the center of the system and safety modeling phases.

2 The TGMT CBTC Railway System

TGMT [4] (Trainguard Mass Transit) is a Siemens customized CBTC system. A CBTC is a railway signaling system that uses wired and wireless communications between the train and the track equipments for the traffic and infrastructure control. It significantly improved the way trains were localized.

Old systems used the track occupancy to determine the position of a train while CBTC equipped trains determine independently their localization and forward it to the track equipments.

The Trainguard MT CBTC system represents the operating system of a train. It is composed of two subsystems: the on-board and the wayside.

The on-board subsystem controls the train doors, the braking, the train position, its speed and the stop with the information to the passengers. The wayside mainly determines the train movement authority according to their speed and position.

The specification of TGMT systems consists of a series of documents of up to a thousand pages each. These aim at describing as explicitly and precisely as possible the system. Written in a natural language, these documents describe the constituents of the system as well as its behavior in its different phases.

3 *ScOLa*, a Scenario Oriented Language for Railway Systems

Often models are used as graphical notation to depict systems. *ScOLa* defines a system by means of *concepts*. An accurate complex systems specification is composed of a description of its *system architecture* as well as its *behavioral* description.

System architecture provides a description of the physical parts (also called components) of the system. It consists of a derivation of all the sub-components of the system. These components are the ones realizing, individually or in cooperation, the behavior of the system.

ScOLa is based on numerous concepts that allow modeling the system architecture and its behavior. System architecture is a set of components that build the structural part of the system. For example, the system train is a component and it is composed of two sub-components: the on-board and the wayside. Thus, the *system architecture* in a *ScOLa* model is the hierarchy of components that belong and define the system.

3.1 Concept of Component

A component in *C,* the set of all components, executes scenarios and may receive and send data through these scenarios. It may have interfaces to communicate with the other components of the system.

A component can be either *basic* or *complex*:

– *Basic*: it cannot be decomposed and is thus considered as atomic.

– *Complex*: it can still be decomposed into sub-components.

A component in the system architecture is unique. Still in behavioral descriptions, one might need several instances of a component to depict some situations. In the train systems, the description of the communication between trains requires the definition of two instances of the same train. Therefore, we introduce the notion of *block*. It represents instances of components from the system architecture. A system description may contain several block descriptions; each one may be used to define several behavioral models.

3.2 Concept of Scenario

The *ScOLa* behavioral model is a mathematically defined model based on the concept of scenarios.

The use of scenarios is motivated by their efficiency in the behavioral descriptions; they are operational instances of system use. Moreover, they involve major system architecture artifacts as the functional architecture (scenarios represent system functions triggered by events and specific configurations). Scenarios also include the physical architecture by allocating the components, and also the requirements associated with each scenario.

Furthermore, scenarios are used in multiple steps of the system conception: the software development and the test cases definitions.

The safety analysis phase is also based on the functional scenarios. They are used when failure scenarios have to be derived from the train potential accidents.

The *ScOLa* behavioral model is a set of scenarios S that describe the system behavior at different abstraction levels. A scenario can be decomposed into several sub-scenarios or a set, A, of atomic *actions* that are realized by a set of *components C*, either individually or in cooperation.

Formally, a scenario $s \in S$ can be defined as $s = <\mathrm{Id_s}, \mathrm{L_s}, \mathrm{F}(s)>$, where:

– $\mathrm{Id_s}$ is the unique identifier of the scenario;
– $\mathrm{L_s}$ is the abstraction level of scenario s;
– $\mathrm{F}(s)$ is the set of sub-scenarios or actions composing s, such that $\mathrm{F}(s) \subset \mathrm{S}$.

3.3 Concept of Action

Actions represent a description of a scenario at the lowest abstraction level. While scenarios descriptions are used as a mean to communicate and contain informal descriptions of the system, actions provide the allocation of the system architecture to the behavioral model. An action a in the set of all actions A is formally represented by a tuple $<\mathrm{Id_a}, \mathrm{C_a}, \mathrm{L_a}, \mathrm{T_a}>$, where:

– $\mathrm{Id_a}$ is the unique identifier of a.
– $\mathrm{C_a}$ is the set of components from the system architecture allocated to a.
– $\mathrm{L_a}$ is the corresponding abstraction level of a.
– $\mathrm{T_a}$ is the corresponding type of a such as $\mathrm{T_a} \in \{\mathrm{Simple}, \mathrm{Transfer}, \mathrm{Question}\}$.

The type of an action depends on the number of resources and the input data it requires, and/or the results it produces as follows:

– *Simple action*: when it requires the resources of a single component to be completed. Formally, if $a \in A_s$ the set of simple actions, then $\exists$ c $\in C$ such that $a \in \mathrm{A(c)}$.
– *Transfer action*: this is a shared action between two or more components.

Such an action can be a data transmission between two components of the system, and thus requires the cooperation of both components. Let A_t be the set of transfer actions. If $a \in A_t$ then $\exists$ c_1, $c_2 \in C$ such that $a \in A(c_1) \cap A(c_2)$.

– *Question action:* it Allows the System to Choose Between Two or More Alternative Behaviors

Typically, a question action can be a test on data in order to choose which action to proceed within the next step. Formally, if $a \in A_q$ the set of question actions then $\exists a_1, a_2, \ldots, a_n \in A$ such that executing a leads to the execution of a_1 or a_2 or ... or a_n.

3.4 Concept of Refinement

Because the different views of the system architecture may provide too detailed functions (functional view), components (organic view) and events (event-based view). It becomes necessary, during the system engineering process, to structure this information and introduce a certain hierarchy between them. In that purpose, *ScOLa* provides the concept of *refinement* as one of the main modeling language concept.

This concept implies the possibility to synchronize the refinement of both components and scenarios.

It helps distinguishing between high-level scenarios that can be useful; for example, to define the critical events during safety analysis of the system. One might also need to refine and have more details about the high-level scenarios and components during the system validation and the description of the hardware failures.

Moreover, ScOLa provides concepts that explain how scenarios are sequenced. They are the following:

3.5 Concept of Precedence

Since actions often require resources to be completed. Each action in a sequence of actions must wait for the previous one to finish. *ScOLa* provides the concept of precedence when the execution order of the actions is already known.

Formally, given actions a_1, $a_2 \in A$. a_1 and a_2 follow a precedence order if $t_0(a_2) \geq t_1(a_1)$, $t_1(a_1)$ being the starting time of action a_1 and $t_0(a_2)$ the end time of action a_1.

3.6 Concept of Parallelism

In *ScOLa*, parallel actions occur when the sequence order is unknown or irrelevant as these actions do not share resources.

Formally, in pure parallelism, if a_1, $a_2 \in A$, $\exists$ [t_1, t_2] such that $t_0(a_1)$, $t_1(a_1) \in [t_1,t_2]$, then $t_0(a_2)$, $t_1(a_2) \in [t_1, t_2]$.

In *ScOLa*, parallelism represents a particular case of precedence where $t_0(a_2) \geq t_1(a_1)$, or $t_1(a_1) \geq t_0(a_2)$.

3.7 Concept of Preemption

In *ScOLa* models, scenarios are triggered by events. Sometimes, a choice has to be made between multiple scenarios or actions.

Preemption refers to the cases where the precedence order is set by a choice between alternative actions triggered at the same time.

The difference with parallelism is that only one action is completed and the others discarded. Thus resource sharing between these actions is possible in the context of the preemption concept.

Formally, given actions a, a_1, $a_2 \in A$, preemption concept is defined as follows:

If $a \in A_q$:

– a is followed by a_1 if a is true
– a is followed by a_2, otherwise

4 Modeling a CBTC Scenario Using *ScOLa*

In this section, we explain how, starting from an informal description of a CBTC scenario S = "*The Speed-Dependent Door Supervision Scenario*", we define a model *ScOLa* using both textual and graphical representations.

In this scenario, the train is responsible for the train doors opening and closing. It is also responsible for handling the train speed. The train is allowed to open the doors if and only if the train reaches a minimum speed. Hence, this scenario is a number of interactions between the train's sub-components.

A summary of the scenario would be as follows: when the train is fully berthed at a Platform and stops, the train doors are released and opened by the driver. Then, the train starts to roll away. The emergency brake is applied when the train exceeds a certain minimum speed.

The description of this scenario in the system specifications is the list of the following steps:

– *Step1*: The train approaches the stopping point, it is already fully berthed. The on-board sub-system indicates this to the HMI.
– *Step2*: The train comes to a standstill; the on-board subsystem releases the train doors at the correct side.
– *Step3*: The driver initiates door opening. The on-board subsystem opens the train doors.
– *Step4*: The doors open. This is reported to the on-board subsystem.
– *Step5*: The on-board subsystem indicates the open doors to the HMI. It sets the recommended speed to zero.
– *Step6*: The train starts to move, the configured minimum speed for the door supervision is not yet exceeded. The on-board subsystem reacts by revoking the door release.
– *Step7*: While rolling, the train loses the fully berthed status. The on-board subsystem revokes the fully berthed indication to the HMI.
– *Step8*: The train exceeds the configured minimum speed for the door supervision. The on-board subsystem applies an emergency brake.

In the following, we present a reverse engineering method to model this scenario in *ScOLa.* We start by the lowest abstraction level description and provide a multi-level description.

a. *System architecture*

From the informal description, we retrieve the list of the implied components. According to the eight steps of scenario *S*, the components are the train, the on-board, the driver, the HMI and the Platform. However, these components belong to different abstraction levels (see Fig. 2). For example, the on-board is a sub-component of the train. Thus, the description is not uniform. Moreover, in some steps (Step 4), the use of the

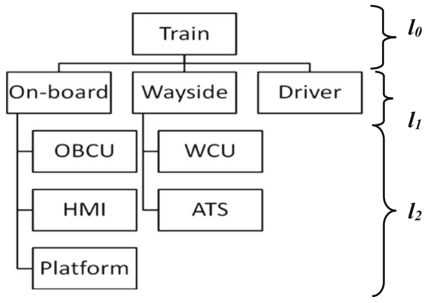


Fig. 2 Architecture of the components involved in S

passive voice does not indicate the responsible for such operations. After a study of the system specifications, we introduce the Wayside sub-component, composed by the WCU and ATS sub-components. They are responsible for the train stop detection and the speed supervision.

b. *Definition of blocks*

They represent the instances of the architecture that are used in the scenario. *S* considers one instance of each element of the architecture defined above (Train, On-board, Wayside, Driver, OBCU, HMI, Platform, WCU and ATS).

c. *Construction of a scenario using ScOLa*

Considering the architecture definition, we consider three levels of abstraction l_0, l_1 and l_2. The scenario as it is depicted in the specifications, is at the lowest abstraction level, noted l_2.

The formalization of such scenario using *ScOLa* concepts starts with the following assumptions:

- *S* is decomposed step by step, each step is decomposed with respect to the system architecture.
- All the sub-scenarios of *S* are actions.
- A simple action requires only one component to be completed.
- A transfer action requires two components. Both need to have the same ancestor in the architecture.
- A question action is a condition or a test.

We note s_{ij} a sub-scenario of *S;* where *i* is the corresponding abstraction level and *j* is the sequencing number of *S* at abstraction level *i*. The words in bold represent the components involved. At abstraction level $l_{2,}$ the sub-components involved in *S* with respect to the system architecture are the OBCU, HMI, Platform, WCU, ATS and Driver.

We proceed to the construction of the abstraction level l_2 of *S* using the following assumptions on the system specifications:

- The OBCU is responsible for detecting the train berthed
- The OBCU is the sub-component of the On-board that communicates with other sub-components.
- The ATS is the sub-components responsible for the train supervision. Thus, it is responsible for detecting the train speed and communicates the information to the train.

For example, *Step 1* is a sequence of two sub-scenarios. The first one consists in the detection of the fully berthed state of the train. The OBCU is responsible for such operations. The second one consists in the indication of the train state to the HMI by the OBCU. We provide the same approach for the remaining steps of S.

Thus, *S* is depicted as follows at abstraction level l_2:

- *Step1:*
- s_{21}: The **OBCU** detects that the train is fully berthed.
- s_{22}: The **OBCU** sub-system indicates this to the **HMI**.
- *Step2*:
- $s_{21:}$ the **OBCU** releases the train doors at the correct side.
- *Step3*:
- $s_{21:}$ The **Driver** initiates door opening.
- $s_{22:}$ The **OBCU** opens the train doors.
- *Step4*:
- $s_{21:}$ The **ATS** detects that doors are open.
- $s_{22:}$ The **ATS** reports the information to the **OBCU**.
- *Step5*:
- $s_{21:}$ The **OBCU** indicates the open doors to the **HMI**.
- $s_{22:}$ The **OBCU** sets the recommended speed to zero.
- *Step6*:
- $s_{21:}$ The **ATS** detects that the configured minimum speed for the door supervision is not yet exceeded.
- $s_{22:}$ The **ATS** transfers the information to the **OBCU**.
- $s_{23:}$ The **OBCU** reacts by revoking the door release.
- *Step7*:
- $s_{21:}$ The **OBCU** revokes the fully berthed indication to the **HMI**.
- *Step8*:
- $s_{21:}$ If the **ATS** detects that the train exceeds the configured minimum speed for the door supervision.
- $s_{22:}$ The **OBCU** applies an emergency brake.

S is composed of several simple and transfer actions. There is a unique question action(s_{21}) in *Step8*.

At abstraction level l_1, *S* is performed by the On-board, Wayside and the Driver as follows:

- *Step1:*
- s_{11}: The **On-board** detects that the train is fully berthed.
- *Step2*:
- $s_{11:}$ the **On-board** releases the train doors at the correct side.
- *Step3*:
- $s_{11:}$ The **Driver** initiates door opening.
- $s_{12:}$ The **On-board** opens the train doors.
- *Step4*:
- $s_{11:}$ The **Wayside** detects that doors are open.
- $s_{12:}$ The **Wayside** reports the information to the **On-board**.
- *Step5*:
- $s_{12:}$ The **On-board** sets the recommended speed to zero.
- *Step6*:
- $s_{11:}$ The **Wayside** detects that the configured minimum speed for the door supervision is not yet exceeded.
- $s_{12:}$ The **Wayside** transfers the information to the **On-board**

– s_{13}: The **On-board** reacts by revoking the door release.
– *Step7*:
– s_{11}: If the **Wayside** detects that the train exceeds the configured minimum speed for the door supervision.
– s_{12}: The **On-board** applies an emergency brake.

At abstraction level l_1, several sub-scenarios from level l_2 were discarded because they consist of interactions from abstraction level l_2 of the architecture.

Finally, the abstraction level l_0 description of S is as follows:

s_{01}: The **Train** is fully berthed**.**
s_{02}: the **Train** releases the train doors at the correct side.
s_{03}: The **Train** initiates door opening.
s_{04}: The **Train** opens the train doors.
s_{05}: The **Train** detects that doors are open.
s_{06}: The **Train** sets the recommended speed to zero.
s_{07}: The **Train** detects that the configured minimum speed for the door supervision is not yet exceeded.
s_{08}: The **Train** reacts by revoking the door release.
s_{09}: If the **Train** exceeds the configured minimum speed for the door supervision.
s_{010}: The **Train** applies an emergency brake.

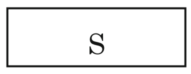
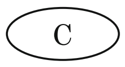
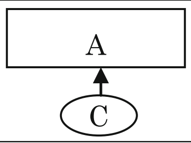
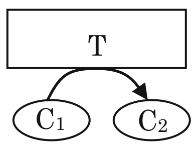
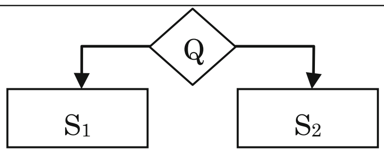
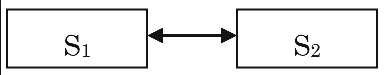
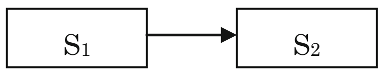
5 Textual and Graphical Representations of S

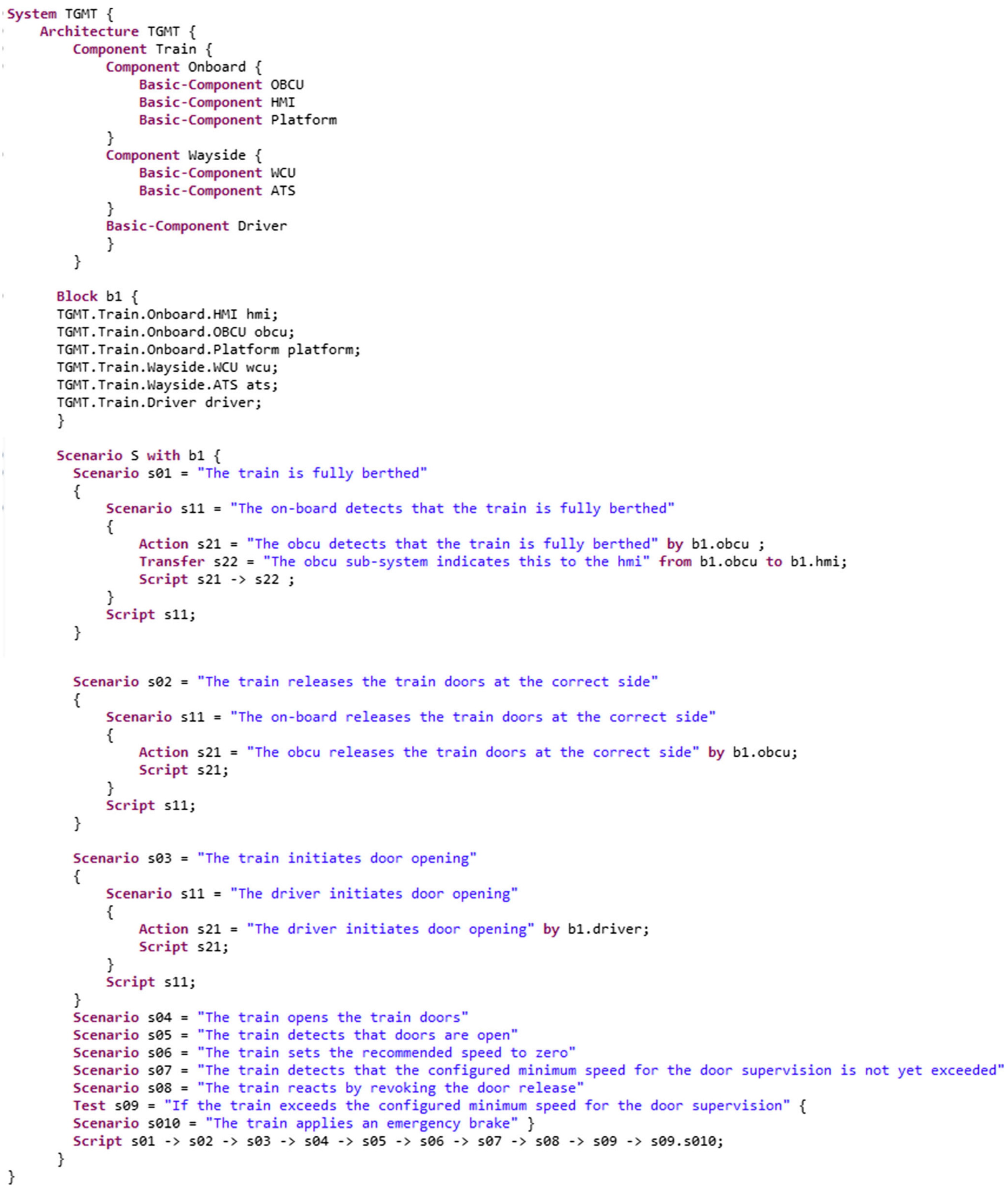
Graphical and textual representations of a model in *ScOLa* use the idiomatic representations in Table 1.

The textual representation of S is composed of three aspects: the architecture, the blocks and the scenarios. The architecture depicts the hierarchical decomposition of the system's physical parts. While scenarios use instantiations of components defined in blocks.

Figure 3 depicts the textual representation of S. We start by declaring the required components in the architecture. After that, we define the block used for scenario S. Finally, we describe the scenarios using the keyword **Scenario**, the

Table 1 Graphical and textual representations of *ScOLa* concepts

Concept	*Graphical representation*	*Textual representation*
Scenario		**Scenario** S
Component		**Component** C, **Basic-component** C
Simple Action		**Action** A **By** C
Transfer Action		**Transfer** T **from** C_1 **to** C_2
Choice Action		**If** (Q) {S_1} **else** {S_2}
Parallelism		$S_1 \parallel S_2$
Precedence		$S1 \longrightarrow S2$



```
System TGMT {
    Architecture TGMT {
        Component Train {
            Component Onboard {
                Basic-Component OBCU
                Basic-Component HMI
                Basic-Component Platform
            }
            Component Wayside {
                Basic-Component WCU
                Basic-Component ATS
            }
            Basic-Component Driver
            }
        }

      Block b1 {
      TGMT.Train.Onboard.HMI hmi;
      TGMT.Train.Onboard.OBCU obcu;
      TGMT.Train.Onboard.Platform platform;
      TGMT.Train.Wayside.WCU wcu;
      TGMT.Train.Wayside.ATS ats;
      TGMT.Train.Driver driver;
      }

      Scenario S with b1 {
        Scenario s01 = "The train is fully berthed"
        {
            Scenario s11 = "The on-board detects that the train is fully berthed"
            {
                Action s21 = "The obcu detects that the train is fully berthed" by b1.obcu ;
                Transfer s22 = "The obcu sub-system indicates this to the hmi" from b1.obcu to b1.hmi;
                Script s21 -> s22 ;
            }
            Script s11;
        }


        Scenario s02 = "The train releases the train doors at the correct side"
        {
            Scenario s11 = "The on-board releases the train doors at the correct side"
            {
                Action s21 = "The obcu releases the train doors at the correct side" by b1.obcu;
                Script s21;
            }
            Script s11;
        }

        Scenario s03 = "The train initiates door opening"
        {
            Scenario s11 = "The driver initiates door opening"
            {
                Action s21 = "The driver initiates door opening" by b1.driver;
                Script s21;
            }
            Script s11;
        }
        Scenario s04 = "The train opens the train doors"
        Scenario s05 = "The train detects that doors are open"
        Scenario s06 = "The train sets the recommended speed to zero"
        Scenario s07 = "The train detects that the configured minimum speed for the door supervision is not yet exceeded"
        Scenario s08 = "The train reacts by revoking the door release"
        Test s09 = "If the train exceeds the configured minimum speed for the door supervision" {
        Scenario s010 = "The train applies an emergency brake" }
        Script s01 -> s02 -> s03 -> s04 -> s05 -> s06 -> s07 -> s08 -> s09 -> s09.s010;
      }
}
```

Fig. 3 Textual representation of the Doors Supervision scenario

simple actions using the keyword **Action** and the transfer actions with the keyword **Transfer**. At the end of each scenario, the script (**Script**) explains the relationship between scenarios.

Figure 4 depicts the abstraction level l_0 graphical description of the door supervision scenario, using the notation depicted in Table 1. Finally, Fig. 5 is the graphical representation of sub-scenarios s_{01}, s_{02} and s_{03}.

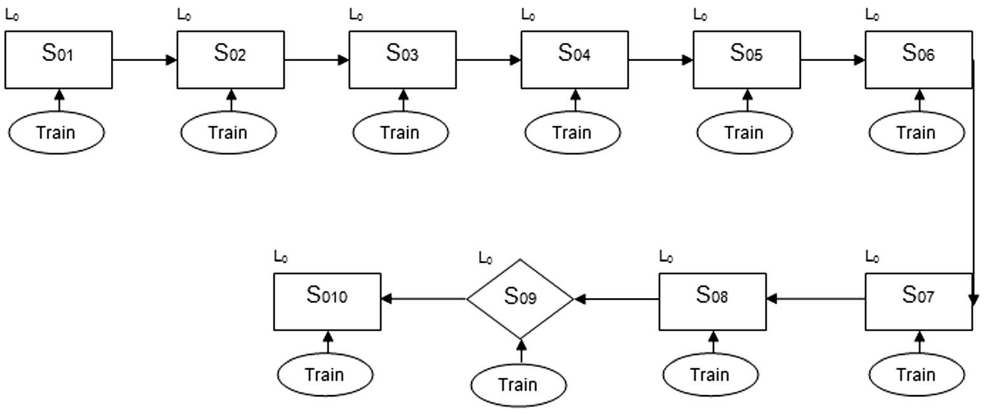


Fig. 4 Graphical representation of S at l_0

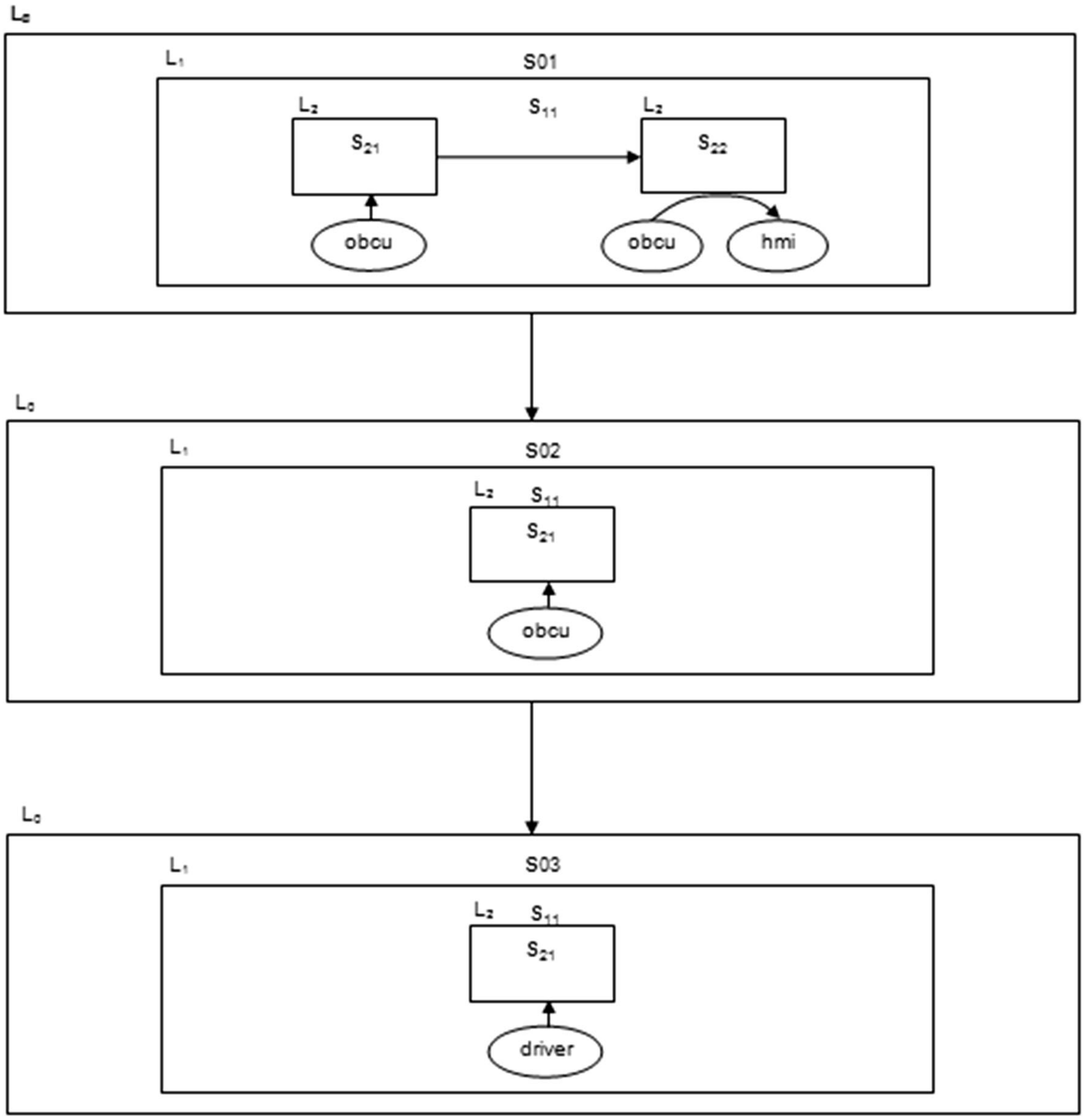


Fig. 5 Graphical representation of sub-scenarios s_{01}, s_{02} and s_{03}

6 Related Work

Model-driven system engineering has been widely studied in the past few years in industry and academy. The main objective is the introduction of the science of models in complex systems modeling. Low-level formal modeling languages are a powerful tool for specifying systems. They are often used at the software development stage and allow a formal validation of systems. For example, the B-method, for specifying, designing and coding software systems based on B [5], is a tool-supported formal method based around an abstract machine notation. It is used in the development of programming language code from specifications. Used in railway automation, it has been used for the specification and validation of the meteor line of Paris [6]. Scade [7] is also a certified formal language used for system development, used in multiple domains. It has been certified by Cenelec EN50128 standard [8]. However, the entry cost is high.

Provided that the main issue is to accurately model a complete complex system starting from a natural-language based system specification. Therefore, semi-formal modeling approaches are proposed. These approaches can be divided into two main methodologies, some are language-centric and others are system-centric. Language-centric methodologies rely on a modeling language. SysML [3] is the more often used in complex systems because it offers a panel of diagrams that allow the graphical representation of systems parts. This approach aims to use all or most of the language properties to express some system views. Thomas Krueger [9] establishes a modeling methodology using SysML trying to represent the functional and organic views of an aerospace system. He uses activity diagrams to model the link between functions and components. This methodology realizes the link between simulation tools with the system for automatic code generation needs. The main advantage is that it represents structural and behavioral views. The inconvenient is in the lack of interaction between components; the system is seen as an independent component which is rarely the case for complex systems. Claver [10] also tried to model astronomical systems by representing the requirements, the logical and physical views with internal and block definition diagrams. The inconvenient is the lack of information, the behavior is forgotten.

System centric methodologies extend or restrain the modeling language they are based on. Wielkiens and Lamm [11] focused on the functional behavior of a complex system. Soares et al. [12] focused on the requirements of the system, the advantage of these methodologies is that the language becomes specific to the system and is customized in order to respond to its properties. The inconvenient is that some views are incomplete. The system is too specific; it cannot be used for formal analysis with other tools.

7 Conclusion

In this paper we presented *ScOLa*, a Scenario Oriented Language, it is a domain specific language for railway systems. We explain the importance of focusing on systems concepts in order to have a coherent and non-redundant model, that is, a trustworthy representation of a system specification. We want the language to be simple but efficient using a small set of concepts. We also focus on the importance to have both textual and graphical representations, which is not the case of the other formal languages. Instead of crossing by semi-formal languages to build a bridge between informal and formal models, we decided to simplify a formal description that relies on a formal semantics.

Our next objective is to define a methodology to perform safety analysis using the execution semantics of ScOLa presented in []. This semantics allows the use of simulation tools to measure the system effects.

In railway systems, safety is still hand-made and relies on the experience of experts. The need for formal methods is important to discover dysfunctional scenarios and find mismatches in the system specifications.

References

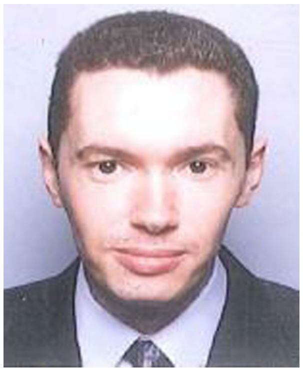
1. 3.1-1999 - IEEE Standard for Communication Based Train Control Performance Requirements and Functional Requirements (2004)
2. Friedenthal S., Griego, R., Sampson, M.: INCOSE Model Based Systems Engineering (MBSE) Initiative INCOSE June 24–27 San Diego (2007)
3. Friedenthal S., Moore, A., Steiner, R., A Practical Guide to SysML, The Systems Modeling Language, MK/OMG Press, ISBN 978–0–12-378607-4 (2009)
4. Trainguard MT CBTC: The moving block communications based train control solutions, Siemens Transportation Systems
5. Abrial, J. R., & Abrial, J. R.: The B-book: assigning programs to meanings. Cambridge University Press (2005)
6. Behm, P., Benoit, P., Faivre, A., & Meynadier, J. M: METEOR: A successful application of B in a large project. In FM'99—Formal Methods (pp. 369–387). Springer Berlin Heidelberg (1999)
7. Abdulla, P. A., Deneux, J., Stålmarck, G., Ågren, H., & Åkerlund, O. Designing safe, reliable systems using scade. In *Leveraging Applications of Formal Methods* (pp. 115–129). Springer Berlin Heidelberg. (2004)
8. CENELEC, EN. 50128: Railway Applications: Software for Railway Control and Protection Systems. European Committee for Electrotechnical Standardization (1997)
9. Krueger, T.: Modeling of a complex system using sysml in a model based design approach. In Proceeding of the ASTRA conference on Automation and Robotics, Noordwijk, The Netherlands (2011)
10. Claver, C. F., Debois-Felsmann, G. P., Delgado, F., Hascall, P., Marshall, S., Nordby, M., ... & LSST Collaboration.: The LSST: A System of Systems. In *Bulletin of the American Astronomical Society* (Vol. 43, p. 25202) (2011)
11. Lamm, J. G., & Weilkiens, T. Functional Architectures in SysML. *Proceedings of the Tag des Systems Engineering (TdSE '10). Munich* (2010)

12. Soares, M.D.S., Vrancken, J.: Model-driven user requirements specification using SysML. Journal of Software. **3**(6), 57–68 (2008)

Melissa Issad Graduate student (CentraleSupélec, 2014), Laboratory of Industrial Engineering of CentraleSupélec and Siemens Mobility. Master of Science in design and management of complex systems (CentraleSupélec, 2013). Engineering degree (Ecole Supérieure d'Informatique, Algeria 2012). Her research focuses on the model based system engineering and safety analysis of railway systems.

Leïla Kloul Assistant professor at the University of Versailles St-Quentin-en-Yvelines, within the DAVID laboratory. Her research interests include the development of analytical solutions for the problem of systems dimensioning, in particular systems with mobile components. The objective of her works is to develop formal methods and tools for performance and reliability analysis of these systems.

Antoine Rauzy Professor at the Norwegian University of Science and Technology, Department of Production and Quality Engineering. Head of the chair Blériot-Fabre "Dependable Embedded Systems" founded by Safran at CentraleSupélec. Member of the board of International Conferences and Journals (Reliability Engineering and System Safety, Journal of Risk and Reliability…).



Karim Berkani Engineer in the Siemens company at Châtillon (France). He develops software processes and tools for the Siemens company. He had different experiences on software and system safety analysis. He worked on different domains like telecommunications or railway systems. In 2003, he received a PhD thesis from the University of Evry-Val D'Essonne (France), under the supervision of Pascale Le Gall. His PhD thesis proposes a method to discover and solve feature interactions in telecommunication systems.