The Journal of Systems and Software 161 (2020) 110478

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

An OSLC-based environment for system-level functional testing of ERTMS/ETCS controllers

Roberto Nardone[a,*], Stefano Marrone[b], Ugo Gentile[a], Aniello Amato[c], Gregorio Barberio[c], Massimo Benerecetti[a], Renato De Guglielmo[d], Beniamino Di Martino[e], Nicola Mazzocca[a], Adriano Peron[a], Gaetano Pisani[c], Luigi Velardi[d], Valeria Vittorini[a]

[a] Università di Napoli Federico II, DIETI, Napoli, Italy
[b] Università della Campania "Luigi Vanvitelli", DMF, Caserta, Italy
[c] Mate Consulting s.r.l., Salerno, Italy
[d] Hitachi Rail STS, Napoli, Italy
[e] Università della Campania "Luigi Vanvitelli", DIII, Aversa, Italy

ARTICLE INFO

Article history:
Received 16 January 2018
Revised 16 September 2019
Accepted 18 November 2019
Available online 19 November 2019

Keywords:
Critical systems
Life-cycle collaboration
Model based testing
OSLC
Testing automation

ABSTRACT

Product and application life-cycle management (PLM/ALM) are the processes that govern a product and a software system, respectively, encompassing the creation, deployment and operation of a system from the beginning to the end of its life. As both PLM and ALM require cross-discipline collaboration and cooperation, tools integration and inter-operation are necessary to enable the efficient and effective usage of tool suites supporting the management of the entire system life-cycle and overcome the limitations of all-in-one solutions from one tool vendor. In this context, the Open Services for Life-cycle Collaboration (OSLC) initiative proposes a set of specifications to allow a seamless integration based on linked data. This paper describes the work performed within the ARTEMIS JU project CRYSTAL to develop an environment for the functional system-level testing of railway controllers, relying on OSLC to enable inter-operation with existing PLM/ALM tools. A concrete realization of the proposed architecture is described also discussing some design and implementation choices. A real industrial case study is used to exemplify the features and the usage of the environment in testing one of the functionalities of the Radio Block Centre, the vital core of the European Rail Traffic Management System/European Train Control System (ERTMS/ETCS) Control System.

1. Introduction

Fostering industrial innovation and competitiveness in global world scenarios requires to cope with collaboration, interoperability and automation issues. In particular, sustainable innovation in the field of critical systems is a serious challenge, as the development of these systems is based on highly complex processes involving specialized tools and different activities throughout the life-cycle. Product Life-cycle Management (PLM) and Application Life-cycle Management (ALM) (Stark, 2015) are the processes that encompass the creation, deployment and operation of a product/software system from the beginning to the end of its life. They need to be supported by set of tools, possibly from different vendors, able to integrate the management of the entire life-cycle. Integration and collaboration efforts are growing both in manufacturing and software industries toward building extended enterprises and promote *life-cycle interoperability*. The current aim, then, becomes the extension of traditional PLM/ALM solutions to manage the process itself (Lacheiner and Ramler, 2011), intended as the possibility to instantiate development processes (or part of them) by "connecting" multi-vendor methodologies, techniques and tools through well-defined "interfaces" (Belkadi et al., 2010).

Several recent European projects (e.g., MBAT Nielsen, 2014, CESAR Jolliffe, 2010, iFEST,[1] R3-COP,[2] CRYSTAL[3]) have taken up this challenge in the field of critical embedded systems, with the objective of defining and implementing multi-domain frameworks, enabling data and resource sharing for sustainable collaboration among all the stakeholders. The multi-domain approach of some

* Corresponding author.
E-mail address: roberto.nardone@unina.it (R. Nardone).

https://doi.org/10.1016/j.jss.2019.110478
0164-1212/

[1] http://www.artemis-ifest.eu
[2] http://www.r3-cop.eu
[3] http://www.crystal-artemis.eu/fileadmin/user_upload/Deliverables/CRYSTAL_D_601_022_v1.0.pdf

of these projects promoted the migration of methods and practices among domains. A key issue of this research has been the usage and the extension of specific PLM/ALM solutions in the direction of data integration, and in the definition of complex and customizable tool chains. This is the approach of the Open Services for Lifecycle Collaboration (OSLC) initiative[4] which is based on the linked data principles.

This paper describes the results obtained in defining, designing and implementing a testing environment within the CRYSTAL project and proposes a reference architecture exploiting OSLC to enable inter-operation with existing PLM/ALM tools. The work originates from the collaboration of four research units: twyo public universities (Università di Napoli Federico II and Università della Campania "Luigi Vanvitelli"), a world-wide company in the design and realization of railway transportation systems (Ansaldo STS[5]) and an ICT SME (Mate Consulting). The focus of the work presented in this paper was on verification and validation (V&V) processes, specifically on the automation of the system-level functional testing of ERTMS/ETCS controllers. The novelties introduced here, however, reside in the proposed OSLC-based architecture, and its implementation realized in accordance with the vision of the CRYSTAL project. Additionally, the components here described implement innovative approaches for automatic generation of test cases and test scripts.

The rest of this paper is organized as follows: Section 2 provides background information on the CRYSTAL project. Section 3 defines the problem tackled in the paper and presents the ERTMS/ETCS system and the system-level functional testing process currently adopted in Ansaldo STS. Furthermore, the section also illustrates the requirements assumed to define and develop the approach. Section 4 provides background information on OSLC. Section 5 proposes an abstract "reference" architecture of the testing environment based on OSLC, which is then instantiated into a "concrete" solution, while the description of the developed components and their interfacing with OSLC is described in Section 6. The approach and its implementation are applied to a real ERTMS/ETCS case-study in Section 8. Section 9 frames this paper in its scientific context and Section 10 ends the paper with a discussion on the most significant aspects arisen during the project. Appendix A provides additional information about the technologies used to develop the proposed architecture.

2. CRYSTAL

The ARTEMIS Joint Undertaking project CRYSTAL (CRitical sYSTem engineering AcceLeration) is a project whose aim was to establish and realize a System Engineering Environment (SEE) to support the design, the development and the deployment of safety-critical embedded systems. The SEE is conceived as a Collaborative Engineering Development Environment and it is based on the definition of a System Life-cycle Development and Management Process. This allows for tool chains to be instantiated and set-up by users *through the integration of the resources made available from different providers on computer networks*, in order to meet desired development goals within specific project scopes and develop products in different application domains (e.g., aerospace, automotive, health-care and rail). The project is ended in 2016, and it was positively evaluated by the European Commission.

To this aim, the CRYSTAL project took up the challenge to establish a Reference Technology Platform (RTP) to provide an environment for embedded systems, allowing loosely coupled tools to share and interconnect their data *based on standardized and open web technologies*. This objective can be pursued if a proper Interoperability Specification (IOS) is adopted that provides a specification for achieving tool and data interoperability in heterogeneous SEEs. The set of solutions, methods, and tools that can be integrated to build the tool chains and to set-up the SEE itself constitutes the CRYSTAL RTP. RTP and IOS, in turn, are defined by collecting requirements from the industrial domains and concrete company scenarios.

The CRYSTAL IOS[6] includes the specifications of:

- communication paradigms and protocols that must be used for exchanging information between integrated tools and data repositories;
- data exchange formats;
- the semantics of the information to be exchanged across tools and data repositories.

As for the communication paradigm and the semantic layer, IOS addresses the usage of the OASIS OSLC standards (OASIS, 2013), which provide the definition of mostly IOS services, meaningful concepts and relationships among them. There is no single technology addressed for the exchange formats: widespread technologies on which IOS is based are RDF/XML, HTTP and core web technologies. This paper presents the results obtained in CRYSTAL with respect to real scenarios from a rail company, specifically from Ansaldo STS, an international transportation leader in the field of signaling and integrated transport systems for passenger traffic and freight operation. The company needs, as expressed by Ansaldo STS in CRYSTAL, were geared towards improving the quality and the efficiency of the *system-level* testing process for railway controllers. As a consequence, the objective of this work was to develop a methodological approach and an interoperable testing environment to be integrated in the RTP and IOS.

3. Problem statement

The problem addressed by CRYSTAL was the integration and interconnection of tools to support collaboration *between business processes* through the development of an open interoperability specification (driven by a large interest group, including representatives from vendors, industrial companies as users, academicians and others stakeholders) and the usage of open and common interoperability technologies supported by the different tools that generate and provide access to data covering the entire product lifecycle. In this broader context, *the problem* we dealt with, was how to improve the efficiency and the quality of the *system-level testing* of railway controllers, by *realizing a complete testing environment, able to inter-operate with standard tools and repositories for requirement and quality management through the CRYSTAL RTP, and according to the CRYSTAL IOS.* This paper is centered on the proposed architecture for the testing environment and presents one possible implementation with a special focus on the components enabling its integration into a System Engineering Environment (including other PLM/ALM tools) according to an OSLC-based interoperability specification. Nonetheless, in order to provide a detailed description of the architecture, also the components implementing the automated testing approach are introduced to a some extent. This Section provides the information needed to clearly state the problem we addressed in the rest of the paper. Hence, the System Under Test (SUT) is described and some details about the development life-cycle, in particular about the validation process adopted in Ansaldo STS, are provided. More important, the requirements to be met in designing and realizing the testing environ-

[4] http://open-services.net/specifications/core-2.0

[5] The company name is Hitachi Rail STS from April 1st, 2019. For consistency with the CRYSTAL project, in this paper we use Ansaldo STS.

[6] http://www.crystal-artemis.eu/fileadmin/user_upload/Deliverables/CRYSTAL_D_601_022_v1.0.pdf

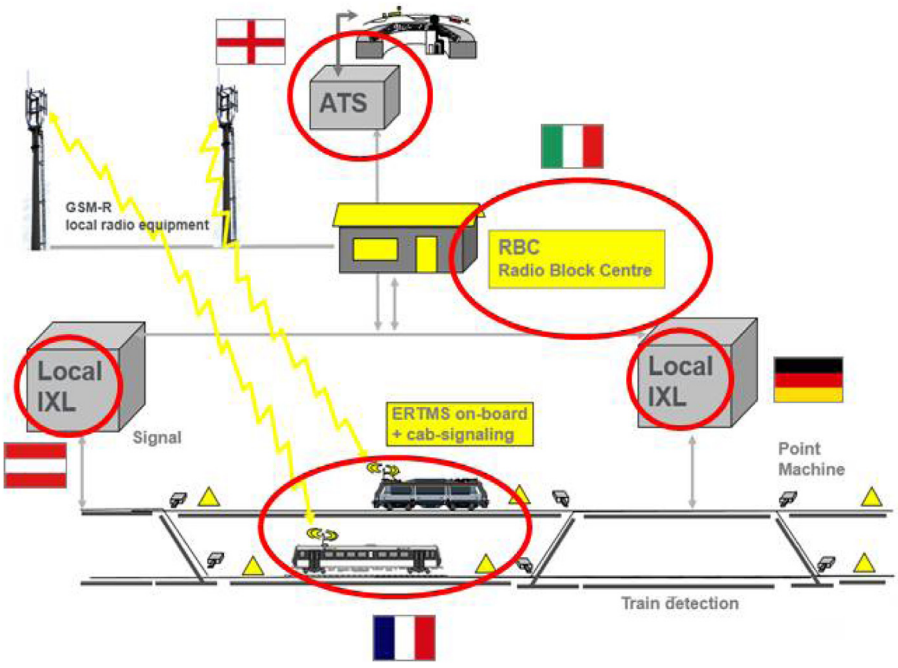

Fig. 1. An ERTMS/ETCS system.

ment are introduced in Section 3.3, these requirements have driven the choices and the implementation described in the following.

3.1. ERTMS/ETCS Radio Block Centre

The European Rail Traffic Management System/European Train Control System (ERTMS/ETCS) is a standard for the interoperability of the European railway signaling systems, ensuring both technological compatibility among trans-European railway networks and integration of the new signaling systems with the existing national interlocking systems (IXL). Exploiting ERTMS/ETCS, a train is enabled to run along the rail infrastructure, regardless of the country the train is traveling in, the infrastructure manager and the supplier providing the ERTMS/ETCS system (Fig. 1). The Radio Block Centre (RBC) system is the vital core of ERTMS/ETCS being responsible for controlling the movements of the set of trains on the track area under its supervision to guarantee a safe inter-train distance. The main goal of RBC is to timely transmit to each train its up-to-date Movement Authority (MA) and the related speed profile. The MA contains information about the distance the train may safely cover, depending on the status of the forward track. As described in Fig. 1, in ERTMS/ETCS Level 2 the RBC continuously interacts with the on-board unit, by managing a communication session according to the EURORADIO protocol on a GSM-R network, and receives information about the track by the interlocking systems (IXL). Finally, an automatic train supervision (ATS) system is in charge of monitoring and controlling the trains to ensure its compliance with the intended schedule. Notice that different subsystems can be produced by different suppliers. A single RBC can simultaneously communicate with different trains and it is also in charge of managing emergency situations, when the communication with one or more trains is compromised.

3.2. Ansaldo STS validation process

Fig. 2 describes the development life-cycle as traditionally implemented in Ansaldo STS for the development of both vital and non-vital computer-based equipments. This life-cycle is compliant with the CENELEC EN50128 standard (CENELEC EN 50128, 2012) and follows the well known V-model, where the left-side reports the decomposition of requirements leading to system low-level design and implementation, and the right-side reports the Verification and Validation (V&V) activities that are executed continuously throughout the development process. This represents the starting point of our analysis and intervention in the CRYSTAL project.

System requirements are analyzed in the early phases of the development process (i.e., during the *system specification*) and a *criticality level* is assigned to each of them, with respect to the safety goals, thus identifying safety-related requirements. System requirements are then apportioned to the different subsystems and components, ending the left-side with the system implementation. Different verification techniques are applied during the right-side of the development model, in order to verify the compliance of components, subsystems and of the integrated system with the corresponding requirements. For safety related-requirements, ALARP (i.e., *as low as reasonably practicable*) principles (Baybutt, 2014) are applied, asking to demonstrate that each critical device must fulfill given thresholds for the maximum probability of dangerous failures (Summers, 1998).

In general, four main kinds of activities are conducted to guarantee that both safety and non-safety related requirements are met: code inspection, functional testing, quantitative analysis and system/acceptance testing. In particular, this paper addresses the system functional testing, that is the system validation against its specification. This activity takes the system functional specification and the concrete implementation as its input and deals with the execution of test cases on the integrated system. The steps of this activity are detailed in the gray area on the right side of Fig. 2. According to the CENELEC EN50128 guidelines, the system testing must not be influenced by the development activities, hence testers cannot access to the software code, but they have to rely only on the initial specification during the definition of test cases. The system has to be considered as a black box, and the execution of test cases consists in sending proper commands and signals to the real system, evaluating the compliance of the observed behavior with its specification. As explicitly recommended by CENELEC EN 50128, models and other artifacts produced during this activ-

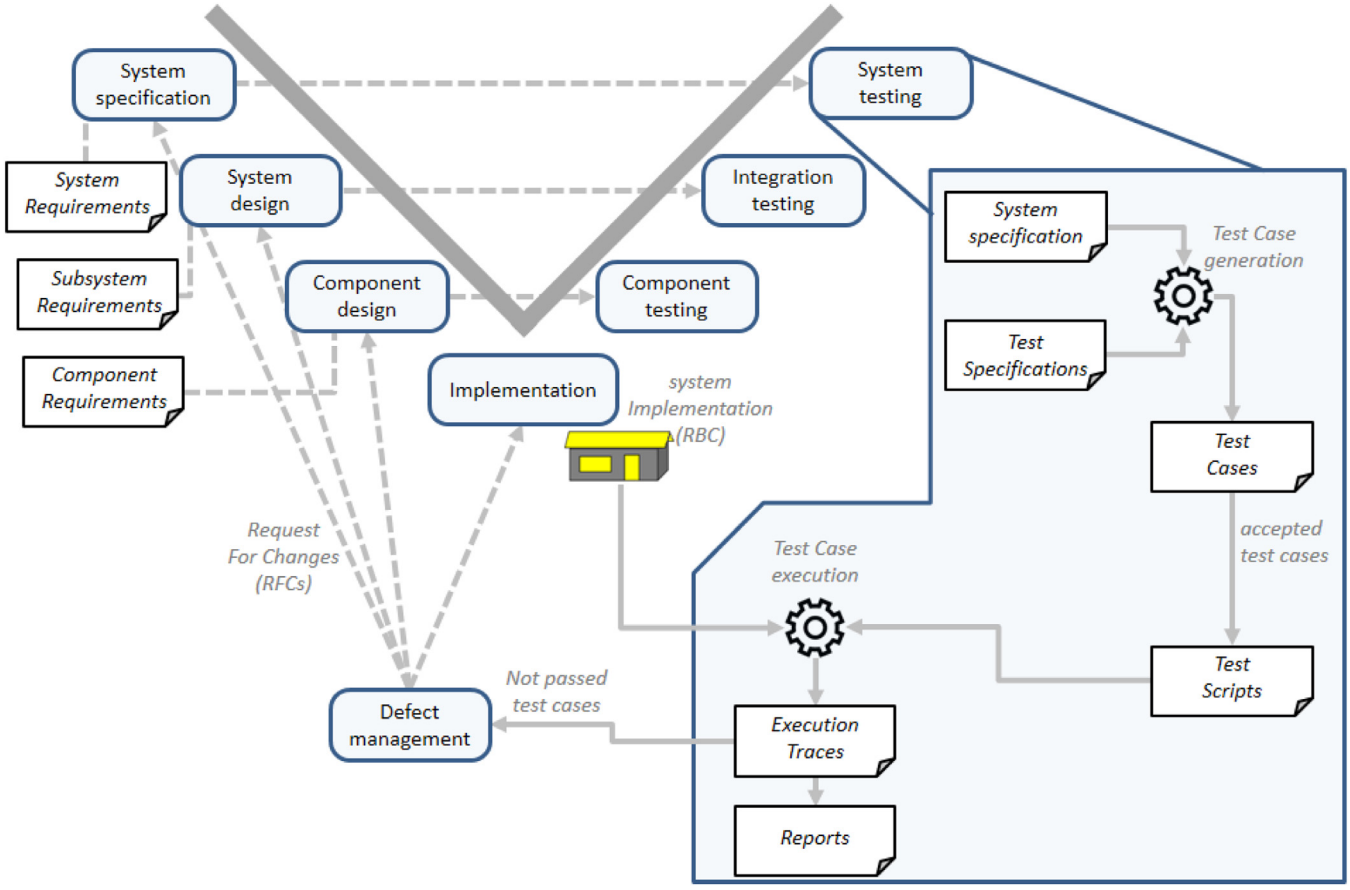

Fig. 2. Ansaldo STS validation process.

ity have to be *specifically developed for V&V purposes*, meaning that they must be completely different from models which brought to the implementation. The goal is to reach the complete coverage of the system behavior, as described in its requirements.

Test specifications, which define the goals of test cases, are obtained from system requirements. Based on the expertise of the V&V engineers and on their past experience, test cases are derived from both the system and test specifications. The entire set of test cases is later revised so as to remove redundant test cases or merge them when appropriate. The final test scripts are written down for the accepted test cases. One of the metrics used to understand the usefulness of a test specification is the coverage, i.e. the capability of the test to stimulate specific parts of the SUT. More precisely, the goal of system-level functional testing is to cover the entire set of possible behaviors that are specified in the system specification. As said before, the V&V engineers cannot access to software code, which has been already tested during both *component* and *integration testing*, so they have to define test cases by relying only on the system specification. It is important to highlight that the system specification for this kind of systems is complete and does not allow for non-deterministic behavior. In this paper we assume completeness and coherence among requirements; the requirement engineering process is out of the scope of the work here described. Starting from the system requirement document, the complete coverage of all the possible behaviors is mandatory. Since it is an hard task to accomplish manually, the mechanisms able to automatically generate test specifications are essential tools for testing teams. Some studies can be found in literature coping with issues of non-deterministic system testing, such as Boroday et al. (2007) which discusses on the applicability of model checking also in this case.

The present work refers to coverage under two aspects. The first is requirement coverage that is a measure of the traceability between test cases and requirements: verifying that all the specified requirements have been covered by at least one test case. The second concept is related to the transition coverage of the State-Based Testing (SBT) (Offutt et al., 2003) approach. In this cited work, some coverage metrics/testing levels are defined: (1) transition coverage; (2) full predicate coverage;(3) transition-pair coverage; and (4) complete sequence. Without going in he details on comparing such levels (see Ray et al., 2019) in this paper, we just refer to transition coverage as an example of how the proposed approach could support such measures, too.

Test scripts are runnable items, possibly written for proprietary simulation environments. They specify the sequences of external stimuli reproducing a desired system behavior, allowing also for asserting internal conditions that need to be verified on the real system. The execution of test scripts produces test logs, which describe the evolution of the system with respect to the external stimuli. If the system behaves as expected, no further action is required, otherwise a bug has been discovered in the implementation (under the hypothesis that the system specifications are correct). At last, test reports are produced from test logs in order to provide documentary evidence of the testing activity.

The discovered bugs are managed by the development team in conjunction with the V&V team within a defect management process: the V&V engineers formally create a Request For Change (RFC) document that is taken in charge by the developers, Fig. 2 also illustrates this practice. A new software release is ready when the problem is solved and fixed: the new release must be tested again (also with the support of regression testing approaches) in order to close all the open RFCs by the V&V engi-

neers. The detection of a defect that impacts on safety-related requirements is a critical event, that could potentially affect all the phases of the development process of the system. An unsolved bug, in fact, may result in a denial of authorization to operate issued by the Safety Authority for railways.

In general, testing activities are supported by a number of IT systems for requirement, architecture and quality management. New Application Life-cycle Management (ALM) solutions are needed to allow independent software and product life-cycle tools to integrate their data and workflow in support of end-to-end life-cycle processes, thereby increasing productivity and reducing time to market. A testing environment for the ERTMS/ETCS must support the generation of the test cases, their transformation into test scripts and the computer-aided analysis of the test logs. In doing this, the environment must inter-operate with existing tools for requirements, architecture and quality management and automate the resulting workflow. In the next subsection we point out the requirements a novel environment for system-level testing of railway controllers has to fulfill.

3.3. Requirements

The problem addressed in CRYSTAL as part of the railway domain case study, was the definition of a *complete and interoperable* environment for *system-level functional testing* to be used in the development process of RBC systems, and the realization of the related prototype. Hence, we had to cope with both test automation issues, in particular the automated generation of test cases, and interoperability issues. In fact, the testing environment must automate the steps described above and enable cross-border testing, i.e. it must support the verification of supplier-to-supplier compatibility to guarantee that the controller in charge of managing the trains can communicate with any European on-board unit. Consequently, *interoperability* of the testing environment is intended here with two different meanings: as i) the capability of supporting the testing activities needed to verify the supplier-to-supplier compatibility; and as ii) the capability of inter-operate with diverse tools, possibly provided by different vendors. As to the aspects related to tools inter-operation, the testing environment has to meet the following main requirements:

R1 *Limited impact and effort*: When tackling the challenge of introducing new approaches and technologies in well-established life cycle processes, and in particular in the development process of critical systems, their impact should be carefully considered. The ultimate goal is to innovate the processes, increase interoperability and competitiveness but this should be reached while also ensuring both quality and cost control.

R2 *Automation*: the steps from the definition of the test specifications to the generation of the test reports should be automated.

R3 *Inter-operation*: the environment should have the capability of exploiting and using different available resources (artefacts, tools, techniques, etc.) supporting the testing process and enabling the sharing of data. The resources may belong to different companies/organizations and may be managed by proprietary dedicated tools.

R4 *Separation of domains*: the artifacts produced during the testing process (requirements, models, test specifications, test cases, test scripts, logs, reports etc.) belong to different conceptual domains (requirements management, definition of specifications, quality). Explicit separation between domains guarantees the necessary independence between the logical steps of the testing process.

R5 *Traceability*: as required by applicable standards, links among the artifacts produced throughout the testing process should be provided and properly handled. In general, the environment must enable the traceability between test cases and related execution traces, and between system requirements and the sets of test cases used to validate them.

R6 *Customization*: the test case generation should be independent of the specific language used to write the test scripts. The environment should allow for easy integration of both standard and proprietary script languages. In addition, the user should be able to specify which information must be included into the reports and the layout of the reports.

The requirements listed above can be divided into groups according to different points of view. As the work was driven by the customer's needs, the business goals were considered.

According to a business perspective merely demanding short time to market and low costs, the primary requirements are limited impact and effort (R1), automation (R2), traceability (R5) and customization (R6), so to obtain quality and flexibility at reduced effort.

According to a business perspective looking at both innovation and competitiveness, the primary requirements are automation (R2), inter-operation (R3), separation of domains (R4) and traceability (R5), so to obtain better products at a sustainable cost.

The intersection between these two groups contains automation and traceability, so these two requirements were of major interest in defining and implementing the testing framework, as well as inter-operation that was the key theme of the research conducted in CRYSTAL.

This paper first proposes a possible *reference architecture* for the testing environment and then describes the actual solution which has been designed and implemented.

4. Background

This section briefly introduces the basic concepts and the terminology used throughout the paper, with a specific focus on OSLC.

4.1. Basic concepts and terminology

A typical interoperability scenario requires an exchange of information between stakeholders, tools, and data repositories. Artifacts internally managed by an engineering tool (hosted on a computer over a network) are usually stored as files or data chunks in databases. As explained in Section 2, artifacts can be shared and they can be accessed via web services. The preferred architectural style for web services in the IOS is the REpresentational State Transfer (REST) (Fielding, 2000) which allows for the mapping of CRUD (Create, Read, Update, Delete) operations into HTTP methods (e.g., Create/Post, Read/Get, Update/Put or Post, Delete/Delete). The servers providing services are called *Providers* and the clients consuming services are called *Consumers*. A Provider offers the functionality to read, query and manipulate the exposed artifacts so that Consumers do not have direct access to them. The IOS defines the basic services that can be used to manipulate the shared artifacts as well as the communication protocols for their serialization. To bridge the gap between different semantics and syntax used internally to define the artifacts, specialized software components (Adapters) are required. Adapters implement IOS compliant interfaces, map the internal artifacts to the syntax and semantics defined by the IOS, and use the proprietary APIs (Application Programming Interfaces) to connect to the back-end of the tool they integrate. Adapters have to be developed on both Provider and Consumer sides, in order to integrate the engineering tools into the RTP and enable the construction of tool chains. Many of the IOS services are defined in the specifications of the OSLC standards; thus, in the following, Adapters are referred as *OSLC Adapters*.

4.2. The Open Services for Life-cycle Collaboration

OSLC is a set of specifications, based on the W3C Linked Data, for integrating tools and easily building development environments via REST-based (RESTful) web services. Linked Data is an approach of publishing structured data, so that data from different sources can be connected and queried. According to this, OSLC may enable the integration between life-cycle data and, in fact, it is a key standard to define integrated Application Life-Cycle Management (ALM) and Product Life-cycle Management solutions. In the OSLC terminology a *resource* is any artifacts — in the life-cycle of a product or software application — identified by an URI (Uniform Resource Identifier) and having an RDF (Resource Description Framework) representation. Resources are connected by using URIs as resource names, including links to other URIs, using HTTP URIs (URLs) to enable look up of names and providing information about resources using standards (e.g., RDF) in accordance with the four Tim Berners-Lee's principles for linked data[7]. The main services powered by the CRYSTAL IOS are defined by OSLC. They are CRUD services, resource linking services using URIs, resource querying services, and user interface preview and delegation.

An *OSLC Resource* is a resource whose type is defined in one of the OSLC specifications. A Resource Type Definition consists of: a name, a type URI, a set of properties and relationships defined by OSLC or coming from other standards, and a set of specific properties and relationships. OSLC Services and Resource Types Definitions are clustered and they pertain to an OSLC domain. In this paper the following domains are considered[8]:

- *Quality Management (OSLC-QM)*: it supports the integration in quality management and testing, by allowing "tools in the software life-cycle (such as requirements and change management tools) to easily create, find, and retrieve resources in a Quality Management system through RESTful APIs"[9];
- *Requirements Management (OSLC-RM)*: it supports integration of requirements management and requirements definition tools by defining "a common set of resources, formats and RESTful services", enabling the effective use of requirements across a development life-cycle[10];
- *Architecture Management (OSLC-AM)*: it supports integration related to models and other artifacts by defining "a common set of resources, formats and RESTful services for use in modeling and Application Life-cycle Management tools".[11]

OSLC provides two main techniques to integrate tools: (i) linking data via HTTP and (ii) linking data via HTML User Interface. The former is a common tool protocol for CRUD life-cycle data. It can be used by a Consumer to communicate with any tool that implements the OSLC specifications: resources are linked by enclosing the HTTP URL of one resource in the representation of another resource.

The latter protocol allows a Consumer to display part of the web User Interface of a Provider, so that a human user can link to a resource in the tool from the Provider or access preview information. The development of the testing environment illustrated in this paper requires exploiting both these techniques and involves resources and/or service definitions from OSLC-QM, OSLC-RM and OSLC-AM.

[7] https://www.w3.org/DesignIssues/LinkedData
[8] http://open-services.net/bin/view/Main/WebHome
[9] http://open-services.net/wiki/quality-management
[10] http://open-services.net/wiki/requirements-management
[11] http://open-services.net/wiki/architecture-management

5. An interoperable testing environment

According to the Section 2, the CRYSTAL project addressed two main problems: the definition of a suitable approach to automate the RBC functional testing and the development of an interoperable environment in which all the required tools can be integrated and exposed to the CRYSTAL RTP.

The former has been addressed by developing suitable modeling languages and tools to automate the generation of test cases and test scripts, as described in Benerecetti et al. (2017), where readers can find all the details about the system model and the automatic generation of test cases including test-data. This paper, instead, focuses on the design and implementation of the testing environment that addresses the requirements described in Section 3.3. This section describes a reference architecture for the OSLC-based automatic testing environment. It is related to the process depicted in Fig. 2 and provides a concrete and feasible implementation of it in a real industrial setting. Hence, any tool chain based on this architecture is well integrated into the system life-cycle, as it does not change the workflow of the activities but improves an existing process. In particular, the main connections with the product life-cycle are realized through the Requirement and Quality Management Resources.

5.1. A reference architecture

The proposed reference architecture is depicted in Fig. 3.

Its main components are: i) engineering *Components* used to automate the testing activities, ii) *Resource containers* storing the OSLC Resources produced or consumed during the validation process, iii) *ALM/PLM tools* providing data management or implementing OSLC automation scenarios. In this architecture, a conceptual distinction is made between OSLC Providers (*PLM/ALM tools*), in charge of providing access to OSLC Resources, and OSLC Consumers (*Components*) that execute the activities needed to automate the testing process. In general a concrete tool can play both the roles of Provider and Consumer. Moreover, it can be conceived for OSLC integration and expose *OSLC Services* or it can be extended to provide tool integration through proper *OSLC Adapters*.

Components may be existing tools already used in the industrial setting or they may be specifically developed. The proposed architecture includes:

- *Human Machine Interface (HMI)*, which implements the user interface, and serves as model editor as well as the control console for the testing process;
- *Model Verifier* in charge of analyzing the specification of the SUT against the syntax and the semantics of the formal notation used to model its behavior;
- *Test Case Generator (TCG)* that produces sequences of abstract steps (test cases) with the associated input values and the expected output values (i.e., test-data) that can lead the system to reach the test goal, starting from its initial state;
- *Test Writer* to transform test cases into low-level test scripts that are executable on the SUT adding pre-setup test steps and post test procedures;
- *Execution Adapter* in charge of launching the execution of the test scripts, collecting the execution traces and creating the traceability links between the execution traces and the test scripts;
- *Trace Analyzer*, which is responsible for the analysis of the execution traces and the production of reports.

Fig. 3 also shows the *Artifacts* produced during the process and their relationships, i.e. the traceability information that should be handled. The dotted arrows represent traceability links between

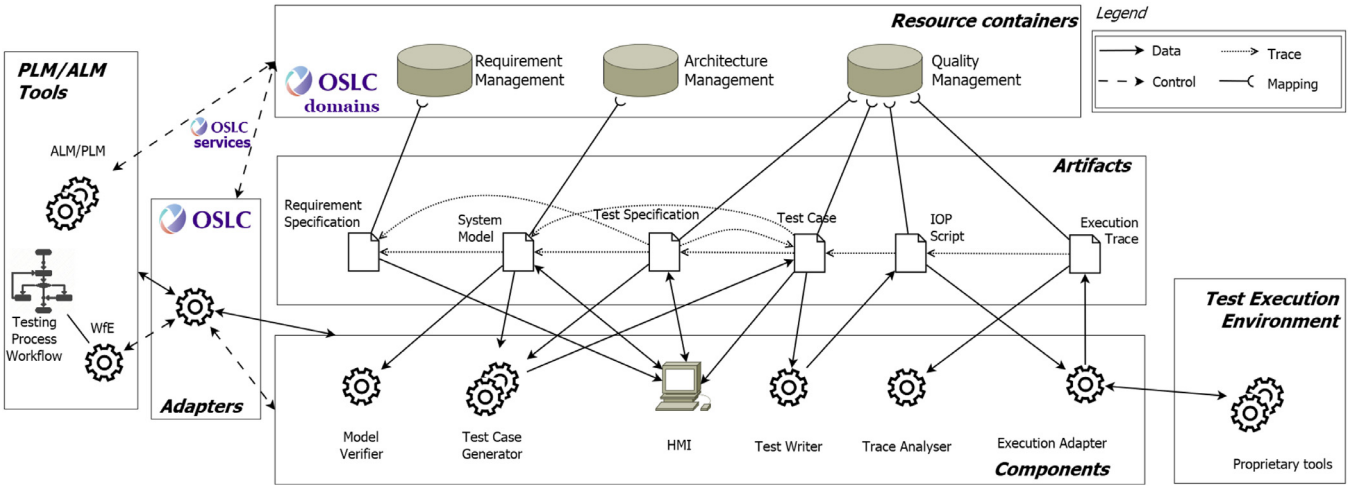

Fig. 3. An interoperable testing environment.

Table 1
Artifacts description table.

Artifact	Description	Related OSLC Resource and Domain	Traceability notes
Requirement Specification	List of the functional requirements expressed in natural language against which the system has to be validated	Requirement Collection (OSLC-RM)	none
System Model	Formal description of the expected behavior of the SUT, expressed through a state-based formalism	Architecture Management Resource (OSLC-AM)	A model element (i.e., a syntactical element of the formalism) that is in turn an AM-OSLC Resource can refer to one or more requirements.
Test Specification	Description of the test goal.	Test Case Resource (OSLC-QM)	A Test Specification can contain a reference to one or more requirements and must refer to a System Model.
Test Case	Ordered sequence of abstract steps (i.e., steps to be realized into test scripts including test-data for each step) needed to execute the test.	Test Case Resource (OSLC-QM)	A Test Case must refer to a Test Specification and it can contain references to one or more model elements.
IOP Script	Encoding of a Test Case into an Executable Interoperable Notation	Test Script Resource (OSLC-QM)	An IOP Script must point to its source Test Case.
Execution Trace	List of events occurred during the execution of a IOP Script	Test Result Resource (OSLC-QM)	An Execution Trace must refer to an IOP Script.

Artifacts: the source of the arrow must contain traceability information about the target. *Artifacts* may be shared with, or by, external environments. Hence, concerning the OSLC Domains, the implementations of this architecture should use or implement suitable tools, or extensions of existing tools, in the RM, AM, and QM Domains, in order to enable the sharing of data as OSLC Resources. According to the OSLC Core Specification, OSLC adopts the convention to provide access to resources through RDF/XML representations. Hence OSLC services should provide and accept RDF/XML representations for each OSLC Resource. Since the attributes of some *Artifacts* (such as *System Models*, *Test Cases* and *IOP scripts*), could not be mapped over common properties of OSLC Resources, it could be necessary to manage *Attachment* files and associate them with the corresponding OSLC Resources. Also in the current version 3.0 of the OSLC standard, the management of attachments is not precisely defined. An OSLC Attachment Descriptor may be used to describe properties of the attached file. This mechanism allows for the creation of attachments to an OSLC Resource by using the HTTP POST method and to update or delete them by using the methods HTTP PUT and HTTP DELETE, respectively.[12]

[12] http://docs.oasis-open.org/oslc-core/oslc-core/v3.0/oslc-core-v3.0-part5-attachments.html

Table 1 illustrates the mapping between *Artifacts* and OSLC Resources and Domains. *System Model* and *Test Specification* do not have an equivalent resource type in OSLC. However, given the definitions of the OSLC domains, it seems natural to assign *System Model* to the Architecture Management Domain and *Test Specification* to the *Test Case* resource type. As a consequence of this mapping, the models produced to derive the test cases should be managed by an AM tool and the test specifications by a QM tool, as shown in Fig. 3. Table 1 for each *Artifact* also sets the traceability links that should be specified by the corresponding OSLC Resource.

Finally, as functional testing is part of a wider verification and validation process within the life cycle management, a workflow of activities that encompasses further steps of the development may be defined. OSLC also addresses workflow automation issues as part of a current work-in-progress, but this is out of the scope of this paper.

5.2. Architecture instantiation

The testing environment described above has been instantiated and applied to the Ansaldo STS Use Case. The instantiation must take into account technological constraints from the application domain and the adopted technologies, but also business constraints

arising from the specific industrial setting. So, when instantiating the reference architecture described above, some difficulties could not be easily overcome, as it will be explained later in this section. As a consequence, the implementation described here differs from the reference architecture in two points: i) it does not implement the OSLC services in the OSLC-AM domain; ii) it customizes the management of attachments concerning the adopted OSLC-QM Resource container.

To better explain how the testing environment has been implemented, let us briefly summarize the test cases generation process developed within the CRYSTAL project. This process exploits model checking techniques to automatically derive test sequences from a test model of the SUT (Gargantini and Heitmeyer, 1999) and model or text transformations to realize a chain of automatic generations. The SUT is modeled by using the Dynamic STate Machine (DSTM) formalism (Benerecetti et al., 2017), a novel state-based language for modeling control systems. The DSTM formalism is an extension of hierarchical state machines, adding the following main key features: (1) a novel semantics for fork and join constructs that allows for the dynamic and recursive instantiation of machines, (2) the introduction of preemptive termination and the possibility of passing parameters to machines at instantiation time. DSTM defines also a formal syntax for the data flow, offering modelers a set of basic data-types (provided the language), the possibility to define enumeration types and more complex data structures, declare variables and communication channels among machines. Hence, this formal data flow have to be used to annotate triggers, conditions and actions of transitions inside machines. Interested readers can find additional details on DSTM in Appendix B. After the modeling activity, the produced System Model (in the DSTM formalism) is verified against the DSTM syntax and semantics and then translated into a network of processes written in Promela (the model specification language of the model checker Spin Holzmann, 2004). The detailed transformation process is detailed in Benerecetti et al. (2019). Similarly, the test specifications are translated into properties, written in Temporal Logic (Baier and Katoen, 2008). The Spin model checker is, then, used to check the properties against the Promela model and to extract counterexamples to violated properties. A counterexample is an execution path of the Promela model that begins at a valid initial state and witnesses the violation of the temporal property. It also provides the flow of data values for that execution (i.e., the test data). As a consequence, a counterexample to a property that negates a test goal actually provides a correct test sequence for that test goal. To turn counterexamples generated by the model checker into abstract test cases, they need to be translated into a proper format, suitable to be used in an actual testing environment. To this end, TESQEL is used, one of the languages specifically developed while defining the entire approach in CRYSTAL. Note, however, that different encodings are possible and quite easy to implement. The final step is to obtain runnable test scripts from the abstract test cases. Again, they are produced by translating the abstract test cases into a proper script notation, where the necessary information needed to automate the execution of the test case is added. The complete generation approach is presented in Benerecetti et al. (2017); Nardone et al. (2014, 2015); Gentile et al. (2014); Flammini et al. (2012); Barberio et al. (2014).

With respect to the reference architecture in Fig. 3, the following *Components* have been developed:

- *HMI*: it provides the Graphical User Interface (GUI), named *RailModel GUI*, used to build DSTM models as well as the functionality that integrates the tools involved in the testing process and supports the user in the testing activities.
- *Model Verifier*: it is a software component, named *DSTM Verifier*, which is in charge of performing lexical, syntactic and semantic analysis of DSTM models and of producing suitable error messages when appropriate.
- *TCG*: the test case generator is a complex toolset, which includes the model transformations from DSTM to Promela, the generation of property specifications as well as the software components handling the invocations of Spin and the transformation of the counterexamples into abstract test cases.
- *Test Writer*: this component, called *IOP Test Writer*, is implemented by plug-ins to provide a flexible and extensible conversion functionality from Test Cases to Test Scripts. It currently allows for producing both human-understandable Test Scripts and Test Scripts used for automatic execution written in the Ansaldo STS proprietary language. *IOP Test Writer* can be easily extended to support other script languages, including a future standard, when it will be available.

As to the *Trace Analyzer*, a third-party tool is currently used to produce analysis reports from Test Scripts and Test Results, as described in the following. The automatic execution of test scripts is performed by an Ansaldo STS proprietary environment and the *Execution Adapter* (Fig. 3) is implemented by the so-called *RTP Adapter*, which provides traceability links among test Results and test scripts and implements the functionality needed to store Test Results Resources into an OSLC-QM Resource container.

Services and Interfaces. The concrete implementation of the proposed OSLC-based testing environment starts from the choice of suitable OSLC Service Provider. According to OSLC, a Service Provider embodies a repository of resources that are hosted by a tool. In this case, two commercial ALM/PLM tools, belonging to the *IBM Rational* software family, have been adopted: *DOORS*, used to define and handle system requirements and their life-cycle; and *Quality Manager* (RQM in the following) that maintains and manages test cases, test scripts and execution traces.

The UML deployment diagram in Fig. 4 shows the relationships between the main components of the implemented architecture. As both DOORS and RQM can host OSLC Service Providers, the proposed implementation is based on the operations that these tools provide to applications that consume services from the OSLC-RM and OSLC-QM domains for data sharing. The functionality of the *Trace Analyser* is partially provided by RQM, which allows for exporting test reports into user-defined Microsoft Excel templates.

RailModel GUI offers functionalities to retrieve Requirements by using the OSLC services provided by DOORS, supports the development of DSTM test models and the execution of the test cases generation process, the creation or modification of OSLC-QM Resources, by using the OSLC services provided by RQM, and specifically, Test Cases and Test Plans. Hence, *RailModel GUI* acts as a Consumer concerning DOORS and RQM.

In a full OSLC scenario, *RailModel GUI* should also play the role of an OSLC Architecture Management Provider and implement the OSLC services needed to share the Artifacts relevant to the development steps of DSTM models. This solution would require a great effort and would affect the existing testing process without any concrete advantages. Since *RailModel GUI* is an OSLC Consumer, *the interoperability and flexibility in the usage of DSTM Verifier and TCG is guaranteed by exposing their functionalities through REST web services.* Thus, *RailModel GUI* interacts with *DSTM Verifier* and *TCG* through RESTful APIs. The *DSTM Verifier* can be invoked directly by *TCG*, so as to allow the DSTM model of the SUT to be verified before executing the test case generation.

RailModel GUI creates and updates the OSLC Resources related to test cases by interacting with RQM. As test cases are documents in TESQEL format, the associated TESQEL file should be provided as an attachment to the corresponding OSLC Resource, if needed. At the time when the testing environment was developed, the management of *Attachments* to OSLC Resources was still under discus-

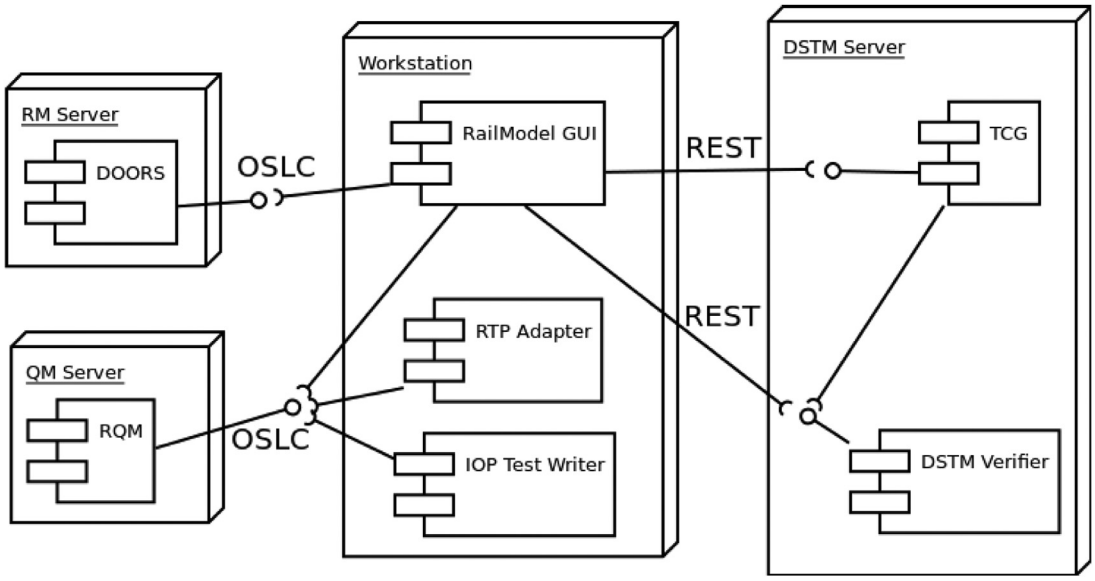

Fig. 4. Interfaces.

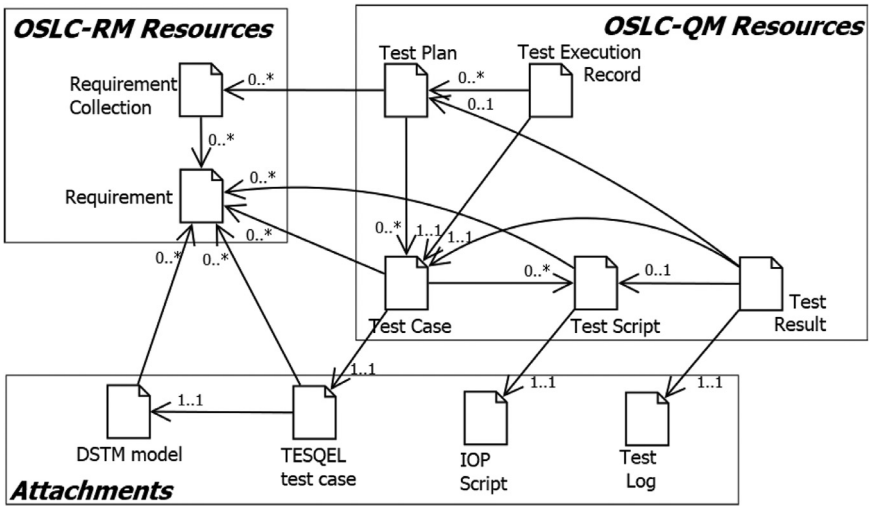

Fig. 5. Traceability links among artefacts.

sion[13] and, to the best of our knowledge, even the current version 3.0 of the standard does not provide normative indications yet. Hence, a customized solution has been adopted, which consists in specifying the URI of the concrete TESQEL file through the specific element provided by the RDF/XML Resource Descriptor of RQM. A similar solution has been adopted for the *Test Writer*, that interacts with RQM to handle Test Script Resources, and for *RTP Adapter*, which enables the interaction with RQM to handle Test Results. In conclusion, two *OSLC Adapters* have been developed to allow *RailModel GUI* to inter-operate with DOORS and RQM. An additional *OSLC Adapter* has been implemented to enable data sharing between *Test Writer* and RQM, whereas *RTP Adapter* is an *OSLC Adapter* itself, provided to share Test Results via RQM.

Fig. 5 depicts the traceability relationships among *Artifacts* and OSLC Resources as implemented by this instantiation. The three boxes refer to the three different domains involved: OSLC RM, OSLC QM and the Inner domain that is realized by a not-OSLC server. While not all the traceability links are defined between OSLC Resources, all the necessary traceability information is guaranteed.

The relationships among Resources (OSLC-RM and OSLC-QM boxes) and their multiplicities are defined by the OSLC specifications[14]. They are created and maintained using suitable OSLC APIs, both intra and inter OSLC Domains. The other links are defined by the adopted modeling languages and notations, i.e., the DSTM and TESQEL meta-models explicitly enable the specification of linked resources using suitable attributes.

The resulting OSLC-based environment, though not fully OSLC, enables, nonetheless, *a step-by-step application of OSLC to a real critical industrial process and at the same time allows exploiting the advantages of OSLC regarding traceability and the possibility of sharing resources.*

Further details about the implementation of the instantiated *Components* and their related *Adapters* are provided in the next section.

[13] http://open-services.net/pipermail/oslc-core_open-services.net/2011-February/thread.html

[14] https://archive.open-services.net/bin/view/Main/QmSpecificationV2.html

6. Components implementation

This section covers the most crucial aspects of the implementation of *Components* and OSLC *Adapters*. The aim of the section is to provide a general picture of the languages, frameworks and technologies involved in the development of the testing environment, including the automated test cases generation process. To help understanding this description, a glossary of all technical terms used throughout is provided in the Appendix. The section ends emphasizing how the implementation meets the user requirements reported in Section 3.3.

6.1. RailModel GUI

RailModel GUI provides the user with a graphical environment for composing DSTM models and generating test cases. It enables the development of a complete DSTM model over multiple files in order to realize a flexible collaborative environment, also implementing exclusive file lock and model comparison mechanisms. It also helps the V&V engineers with typical features such as syntax highlight and automatic graphical arrangements.

RailModel GUI has been entirely realized by using the Eclipse Modeling Framework (EMF) (Steinberg et al., 2008) and the Graphical Modeling Framework (GMF).[15] Both these frameworks allow to define ad-hoc domain specific modeling languages and generate graphical interfaces.

Two *OSLC Adapters* have been realized in order to provide *RailModel GUI* with the capability to access a generic Requirement Management Service Provider and a Quality Management Service Provider (DOORS and RQM in the proposed instantiation).

The first adapter, called *RM-Adapter*, enables the creation of traceability links between elements of the *DSTM model* and *Requirement* resources. Thanks to a specific functionality of *RailModel GUI*, the user can configure the parameters of any external OSLC-RM Service Provider, by specifying the base URL of the application server, the authentication parameters and project details. By means of the *RM-Adapter, RailModel GUI* can access the list of Requirements, retrieve their unique identifiers and additional details. The modeler can examine the available information when building the system specification and she can annotate the Requirement identifiers over structural elements of DSTM models (transitions or states), in this way creating traceability links between elements of the test model and Requirements. All the Requirements available for the project specified in the configuration data are loaded into the *RailModel GUI* cache, thereby avoiding repeated requests and speeding up response times. The *RM-Adapter* also implements a notifications-based functionality, which is triggered whenever the Requirements are updated on the Service Provider. Separation of domains is preserved, by disallowing to edit, add or remove Requirements in the OSLC-RM Service Provider through *RailModel GUI*.

The second adapter, called *QM-Adapter*, is in charge of the communication between *RailModel GUI* and a Quality Management Service Provider. In this way, the set of *Test Cases* generated from the system specification can be stored and handled as QM Resources by means of any OSLC-QM Service Provider. The *QM-Adapter* allows *RailModel GUI* to create or update *Test Plans*, and to add *Test Cases* to *Test Plans*. Again, *RailModel GUI* implements a specific functionality to configure the connection with any Service Provider.

6.2. IOP Test Writer and RTP Adapter

The main functionality of *IOP Test Writer* is the automatic generation of test scripts from test cases. The ultimate goal is to derive test scripts in an interoperable testing language in order to support cooperative testing of ERTMS/ETCS systems. The input to *IOP Test Writer* is the set of test cases produced and written by *TCG* in an ad-hoc defined XML format. Currently, *IOP Test Writer* produces both test scripts understandable by an human user and test scripts written in a proprietary language from Ansaldo STS. However, *IOP Test Writer* has been designed to be easily extensible to other target languages by means of suitable plug-ins and Model-to-Text transformations. In particular, it can be extended to interoperable languages possibly provided in the future by an organization for standardization (e.g., UNISIG, the industrial consortium created to develop the ERTMS/ETCS technical specifications). The tool has been developed as a Windows GUI using the version 4.5.1 of the Microsoft.NET Framework. It communicates with an OSLC-QM Service Provider to retrieve the *Test Cases* and write the resulting *Test Scripts*, creating and updating the related traceability links. Hence, an *OSLC Adapter* enables loading, selecting and displaying previews of *Test Cases* from an OSLC-QM Service Provider. In addition, it allows to generate, save, show and export *Test Scripts*. Provider. In addition, it allows to generate, save, show and export *Test Scripts*.

The main goal of the *RTP Adapter* is the computer-aided analysis of the test outcomes and their storage over a OSLC-QM tool, by providing the user with means to select the test execution traces maintained by the Service Provider. The *RTP Adapter* processes the *Test Execution Records*, generated during the execution of *Test Scripts*, and extracts the information to collect the information needed to create *Test Result*. The aim of the *RTP Adapter* is to store test *Test Result* on the OSLC-QM Service Provider and allow the test cases to be tagged as passed or failed, so managing the needed traceability links. For this reason, the *RTP Adapter* also implements an OSLC Adapter that plays the role of an OSLC Consumer.

6.3. OSLC Adapters implementation hints

In implementing the Adapters, HTTP integration services are used to perform typical C.R.U.D. operations, whereas an HTML User Interface integration service is used to produce a preview of RM resources in a human-readable format (e.g., to visualize details about Requirements).

An OSLC Service Provider offers two main HTTP integration services, namely *Query Capability* and *Creation Factory*), and two main HTML User Interface integration services, namely *Creation Dialog* and *Selection Dialog*.

Query Capability is used for querying resources, by performing an HTTP GET, and provides the base URL for composing queries. Both the *RM-Adapter* and *QM-Adapter* use the *Query Capability* service to access the Requirements list and the Test Plan list, respectively. The *Creation Factory* provides the URL used to create new resources, by performing an HTTP POST. For example, the *QM-Adapter* uses the *Creation Factory* for creating a new Test Plan. It also uses the *Update Resource/Create Resource* service to add a Test Case to an existing Test Plan.

Creation Dialog and *Selection Dialog* allow the Consumer to display a fragment of the web interface of the tool hosting the Service Provider. For instance, the *RM-Adapter* uses the Selection Dialog service to get a preview of Requirement details. Analogously, one of the features implemented by *IOP Test Writer* uses the UI Preview of OSLC to show user in-context information when displaying a link to a Test Case or a Test Script resource.

The OSLC Adapters have been implemented by using the Eclipse Lyo or OSLC4Net SDKs, depending on the used language: the first one supports the development of OSLC solutions in Java, the second one supports solutions in.NET. The listings report some code snippets that show the interactions described above. In particular, Listing 1 shows some methods of the *OSLCRequirementLoader* class

[15] https://www.eclipse.org/gmf-tooling

```
public class OSLCRequirementLoader implements IRequirementLoader {
    private String _serviceURL = null;
    ...
    // Returns a list of requirements associated to the given RM Project.
    public List<ResourceRequirement> getRequirements() {
        List<ResourceRequirement> listReq = null;
        OslcClient oslcClient = getOsclClientInstance();
        // Get the Query Capabilities URL in order to run OSLC queries
        String queryCapability = oslcClient.lookupQueryCapability(_catalogUrl,
            OSLCConstants.OSLC_RM_V2, OSLCConstants.RM_REQUIREMENT_TYPE);
        // Prepare the parameter to retrieve all Requirements in project
        OslcQueryParameters queryParams = new OslcQueryParameters();
        queryParams.setSelect("*");
        // Prepare the query that will be executed
        OslcQuery query = new OslcQuery(oslcClient, queryCapability, queryParams);
        // Run the query and get number of results:
        OslcQueryResult result = query.submit();
        // Each requirement is stored in lisReq list
        for (Requirement rq : result.getMembers(Requirement.class)) {
            GetRequirement(rq, listReq);
        }
        return listReq;
    }

    // Returns the HTML preview of the given requirement identified by its
       identifier
    public String getRequirementUI(String identifier) {
        ResourceRequirement req = getRequirementDetails(identifier);
        OslcClient client = getOsclClientInstance();
        // Give the preview of the Requirement in html format through OSLC client
        ClientResponse response = client.getResource(req.getUrl(), "application/x-
            oslc-compact+xml");
        Compact compactPreview = response.getEntity(Compact.class);
        Preview preview = compactPreview.getSmallPreview();
        URI uri = preview.getDocument();
        String  uriString = uri.toString();
        response = client.getResource(uriString, "application/xml");
        InputStream inputStream = response.getEntity(InputStream.class);
        BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(
            inputStream));
        // Give the content of the preview
        StringBuilder contentPreview = new StringBuilder();
        while((line = inputStream.readLine()) != null) {
            contentPreview.append(line);
        }
        return (String)contentPreview;
    }
}
```

Listing 1. Requirement loader code snippet.

that loads the requirements from RQM on the *RailModel GUI* application. The getRequirements() method uses basic query-related Lyo classes (OslcQueryParameters, OslcQuery, OslcQueryResult) while getRequirementUI(...) uses UI delegation (ClientResponse, Compact, Preview) Lyo classes.

Listing 2 shows the TestPlanCreation(...) method of the QualityManager class in the QM-Adapter of *RailModel GUI*. In this method the following phases can be distinguished: invocation of the TCG (TestCaseGenerator.generate(...)); creation of a OSLC-RQM TestPlan; assignment of each Test Case to the Test Plan; and saving of the related TESQEL file on the storage repository.

6.4. DSTM Verifier and TCG

DSTM Verifier is in charge of checking if a model is correct with respect to syntax and semantics of the DSTM formalism. This phase can be carried out during the editing phase (i.e., it is used as an on-line checker) as well as before the test generation phase. As shown in Fig. 4, *DSTM Verifier* does not communicate with *RailModel GUI* and *TCG* through *OSLC Adapters*, but via RESTful web services.

DSTM Verifier has been entirely realized in Java, using the EMF library. Classical EBNF parsing techniques have been used for ver-

```
public class QualityManager {
    // Create a test plan in the QM tool adding the artefact automatically
        generated by the Test Case Generator
    public ITestCase TestPlanCreation(String TestPlanName , String
        TestPlanDescription, Dstm model, String modelStringData) {
        // Creation of Tesqel Artefact by interacting witn TestCaseGenerator
        TestCaseGeneratorProxy TestCaseGenerator = new TestCaseGeneratorProxy();
        Vector<String> Tesqel_Vectors = TestCaseGenerator.generate(model,
            modelStringData, RequestKind.Transitions);
        // Get an instance of OSLCQMLoader to interact with the QM tool
        OSLCQMLoader qualitiM = new OSLCQMLoader();
        // TestPlan creation in the QM tool
        ITestPlan ITP=qualitiM.createTestPlan(TestPlanName, TestPlanDescription);
        for(int k = 0 ; k < Tesqel_Vectors.size(); k++) {
            // Give the name from the tesqel artefact
            String TestCaseName=getTestCaseName(Tesqel_Vectors.item(k));
            // Give the description from tesqel artefact
            String TestCaseDescription=getTestCaseDescription(Tesqel_Vectors.item(
                k));
            // Give the transition in the Tesqel
            List<String> nameTransition=getTransitionTesqel(Tesqel_Vectors.item(k)
                );
            // Give the requirements from the transition of the model
            List<String> urlRequirements=getUrlRequirements(nameTransition,model);
            // Creation of the TestCase in QM tool
            ITestCase ITC=qualitiM.createTestCase(TestCaseName,
                TestCaseDescription);
            qualitiM.UpdateResourceTestCaseRequirements(ITC, urlRequirements);
            StoreTestCase(Tesqel_Vectors.item(k), ITC.getURI());
            // Add the testCase in the TestPlan
            ((TestPlan)ITP.getTP()).addUsesTestCase(ITC.getUri());
            // Update the TestPlan on the QM tool
            qualitiM.UpdateResourceTestPlan(ITP);
        }
    }
}
```

Listing 2. Requirement loader code snippet.

ifying data definitions and transition decorations. An open source parser and compiler generator (SableCC[16]) has been used to support the development. Data exchange with other components has been realized by using the lightweight data-interchange JSON format.

TCG is in charge of generating test cases from the DSTM model of the SUT, as sketched in Section 5.2. TCG is also based on a client-server paradigm: it operates and communicates with the other components by exposing its functionality through RESTful web services.

The main tasks performed by TCG are: (1) *model merging*, which combines the two separate components (the data definition and the model structure) of a DSTM model before applying the transformation that translates the DSTM model into a Promela mode; (2) *flattening* of the hierarchical structure of a DSTM model, since Promela does not allow hierarchical specifications; (3) *transformation*, where three steps are performed (cross compiling, abstract syntax translation and concrete syntax generation) in order to obtain an actual Promela model that can be automatically analyzed; (4) *claim generation*, which automatically derives the property specifications to be checked on the Promela model from the Test Specifications; (5) *invocation*, where the proper sequence of commands to invoke the Spin model checker is built (including suitable options and parameters), so as to obtain the set of test sequences (counterexamples); (6) *post-processing*, in which the test sequences are processed to obtain abstract test cases written in an EMF-based language. This language, developed as a task within CRYSTAL project, is named TESQEL (TEst SeQuEnce Language). The outputs of *TCG* are TESQEL files containing the generated test cases, which are, then, returned to *RailModel GUI* via RESTful APIs.

The development of *TCG* exploits the following technologies: EMF library for processing both DSTM and TESQEL artifacts, SableCC for parsing and translating data files and transition decorations, ATL (Atlas Transformation Language) for implementing Model-to-Model transformations, and Acceleo as a Model-to-Text transformation. Data exchange with other components has been realized by using JSON.

6.5. Matching user's requirements

Before describing the application of the testing environment to the Ansaldo STS case study, let us discuss how the initial user's requirement, introduced in Section 3.3, are indeed satisfied by the implementation described.

[16] http://www.sablecc.org/

R1 *Limited impact and effort*. This requirement has been met by realizing the testing environment so that a smooth transition to effective tool integration strategies is possible. The joint adoption of OSLC and REST web services enables the instantiation and the inter-operation of the testing environment with a number of commonly adopted ALM/PLM tools (DOORS and RQM are an example of OSLC Service Providers compliant with the Ansaldo STS processes). The adoption of widespread technologies for the development of the framework, based on standard and interoperable formats, also supports the efficient and seamless integration of the testing environment with other life cycle tools used in Ansaldo STS, thereby minimizing the impact on other phases of the production process.

R2 *Automation*. The steps from the specification model to the test reports, as well as the interface with the OSLC Service Providers, are fully automated. As OSLC is a set of standard specifications, the implementation of the Adapters is independent from the specific tools providing access to the OSLC services.

R3 *Inter-operation*. This requirement has been satisfied by using OSLC and other open specifications and programming standards. OSLC allows to access shared resources available over a network from different management tools and build tool chains, as each component providing a management service can be replaced by a different one, as long as it provides an equivalent service.

R4 *Separation of domains*. As a consequence of the adoption of OSLC, different conceptual domains (e.g., requirements, models, test cases, etc.) are clearly distinguished in the proposed environment.

R5 *Traceability*. Backward traceability is guaranteed from Test Results to Requirements across the chain of different tools: OSLC supports the specification of traceability links and this mechanism has been integrated with the features of the modeling languages and notations used to implement the testing process. Moreover, the developed components *RailModel GUI, IOP Test Writer* and *RTP Adapter*, implement the software code needed to manage automatically OSLC traceability links, updating them after changes in the artifacts (e.g., after a re-generation of test cases and re-execution of test scripts) as described before.

R6 *Customization*. This requirement has been taken into account at different levels, by exploiting the advantages of decoupling the input/output formats in the tool chain, by means of model-to-text and text-to-text transformations. In particular, the specific notation in which Test Scripts are written does not depend on the notation used to write the abstract Test Cases, allowing the extension of the environment to support several script languages. The customization of the final Reports in our instantiation is provided by the Quality Management ALM/PLM tool.

7. Discussion

In this section a brief discussion about the work described in the paper is provided. Regardless of the specific implementation of the interoperable testing framework, two main issues need to be addressed: a) the quality aspects of the resulting framework that are considered relevant by the target company/industry that is going to use it; and b) the level of maturity of the adopted standards, technologies and tools.

As to the first point, automation, traceability and ease of integration are crucial requirements, as discussed in Section 3.3. This leads to the second point, as seamless tools interoperability is hard to obtain. A *vertical approach*, in which a single leading technology is adopted throughout the tool chain, is the simplest solution, but this is far from the aims of this work, as explained in Section 2. In turn, an enabling and “above the fray” interoperability framework is needed to implement a *horizontal approach*, thus leveraging the heterogeneity of software systems and tools. OSLC allows independence from different vendors, technologies and standards. OSLC is a good solution to meet integration and traceability requirements, and it is the only way, at the time of writing, to achieve the goal following standard specifications. The OSLC standard is powerful enough to allow for modeling the main aspects of a cross-domain product life-cycle, and it also provides customization facilities to implement specific needs, despite the non-normative management of attachments, introduced in the current version 3.0.

On the other hand, based on the authors’ experience, the widespread adoption of OSLC could require further effort to fill the gap between the concrete industrial setting and the vision that OSLC envisages. In this respect, some interesting remarks are reported in Leitner et al. (2016). The implementation proposed in this paper suggests that an hybrid solution, partially based on assessed and commonly used technologies, may have the advantage of supporting a gradual migration of the industry processes towards life-cycle collaboration. Indeed, when adopting an horizontal approach two complementary aspects have to be considered, that is the support provided by tool vendors to the enabling framework (i.e., OSLC in this case) and the effort to develop suitable adapters, to make the components of the architecture interoperable through the mechanisms provided by the enabling framework. These aspects are strictly related, since the greater the support, the lower the effort, so that the success of the solutions based on interoperability specifications depends on the real interest that tools and technologies vendors have and on the confidence they have in the impartiality of the specific enabling framework. Hence, one of the results achieved with this work, beside the definition and the instantiation of a general architecture for automating the testing process of ERTMS controllers, is precisely the experience gained in using OSLC to integrate life-cycle tools and to enable cross team traceability and collaboration. This is an important result in the direction of stimulating customers to ask for technologies enabling the realization of life-cycle integration strategies.

8. The RBC’s COMM functionality

In order to illustrate the applicability of the proposed approach and its concrete instantiation, here a real implementation of RBC is considered. Specifically, this section refers to the Communication Management functionality (*COMM*), selected by Ansaldo STS as use case in the CRYSTAL project. COMM is in charge of establishing and managing the communication with trains under the RBC control. The communication is organized in sessions, which have to be managed so as to guarantee a proper safety level, even in case of temporary or permanent failures. Three phases are performed, starting from the moment a train enters in the area under the supervision of RBC:

- *Communication establishment*: when a train wants to establish a safe connection with RBC, it sends a proper *connection request* message. RBC may accept or refuse this request according to the number of trains already connected, which is limited for safety reasons. RBC must manage the communication with the trains under its control, while waiting for further connection requests from other trains.
- *Session establishment*: once a connection request has been accepted, RBC must follow a specific protocol with the train, before granting it a suitable Movement Authority. This protocol mainly depends on the state of the train (i.e., if it was previously stopped or if it is coming from a non-ERTMS area).
- *Management of the train movement*: when a session is safely established, RBC periodically sends the Movement Authority to the train, checking the presence of possible human-raised

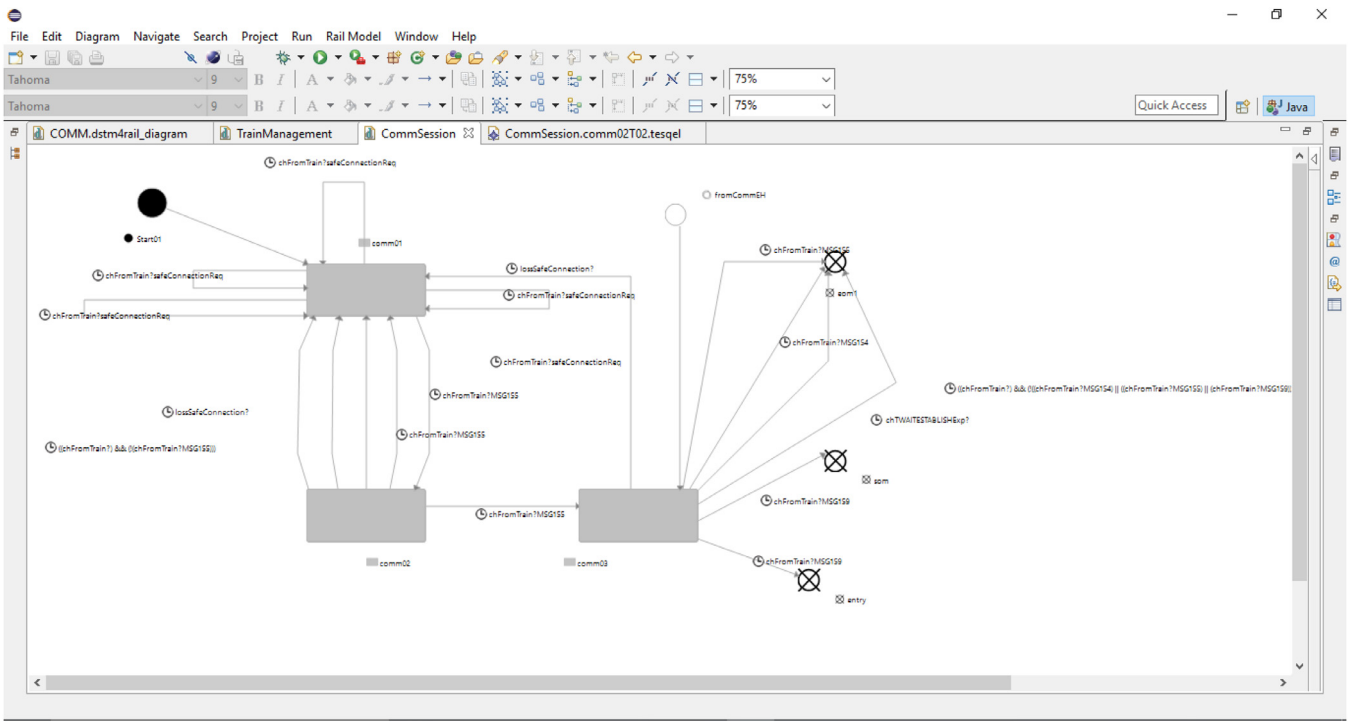

Fig. 6. Rail Model GUI - CommSession machine.

alarms at the same time. The Movement Authority contains information about the distance the train can safely cover (i.e., the following area free from trains). This is crucial to guarantee a safe distance between trains.

A reliable communication channel between RBC and EVC is of vital importance for the safety of the train march. Possible hazards are due to the loss of communication, as well as to the absence of related diagnostic mechanisms, and the consequent impossibility of the train to receive speed restriction and emergency braking messages from RBC. In case of loss of communication, after a given lapse of time EVC shall start proper braking procedures and RBC shall consider that the train is no longer under its supervision. Additional details on this functionality are in Benerecetti et al. (2017) and UIC (2008).

In the following part of the Ansaldo STS Use Case is used as a case study to validate the proposed environment addressing safety-related requirements. The generation of test cases and their execution is out of the scope of this paper (interested readers can found additional details on the DSTM model of the COMM and on the automatic test case generation in Benerecetti et al. (2017); Nardone et al. (2015)). Here the focus is on the usage of the proposed environment. Specifically, some usage scenarios are addressed to show how the environment supports:

1. the annotation of requirements on the DSTM test model;
2. the automatic generation of test cases;
3. the automatic translation of test cases into (human-readable) test scripts to enable their execution;
4. the computer-aided analysis of the test logs;
5. the production of a synthetic test report.

Test model and annotation of requirements A DSTM model of the *COMM* procedure has been defined on the basis of the behavior described by the system-level functional requirements. The entire DSTM model consists of six state machines.

Figs. 6 and 7 show the *CommSession* machine and models the *session establishment* phase. The model was specified by means of *RailModel GUI*: in particular, Fig. 6 represents the DSTM model (made of state machines) and has been created by the graphical editor, while Fig. 7 represents the data model specifying the data structures and the variables used in the model to specify triggers, conditions and actions of transitions between states.

This machine has three nodes (the gray rounded rectangles) that represent the three steps needed to instantiate the communication session. An initial node (the black circle) corresponds to the default starting point of the machine, an entering node (the white circle) represents an additional entry point that can be explicitly specified when calling the machine. Three exit nodes (the crossed circles) model three different termination modes. Each transition can be annotated with a trigger, a condition and actions, which have to be written as strings following the provided DSTM formal syntax. Once the model is specified, the V&V engineer calls the *DSTM Verifier* to check the compliance of the defined model with the formal syntax and semantics of DSTM, including the syntactical correctness of triggers, conditions and actions annotated over transitions.

As explained in Section 6.1, both states and transtion of the DSTM model can be further annotated with requirements that are stored and managed by DOORS. *RailModel GUI* inter-operates with DOORS through the OSLC *RM Adapter*, allowing the modeler to retrieve, preview and select the requirements from a graphical view.

Fig. 8 reports a screenshot of DOORS, showing the list of requirements associated with the *COMM* procedure and details about the requirement with identifier 132, which states the actions to be performed when the safe connection between RBC and the train is lost. Transition *comm02T02* of *CommSession* in Fig. 6 models the situation when of the safe connection is lost. By clicking on this transition, the list of requirements retrieved from DOORS is shown on a window of *RailModel GUI*, so that the modeler can select and add requirement 132 to the list of requirements that annotates transition *comm02T02* (see the left-hand side of Fig. 9). The modeler can view the details of a requirement by double clicking over a single item in the lists. In this case, a delegated User Interface is

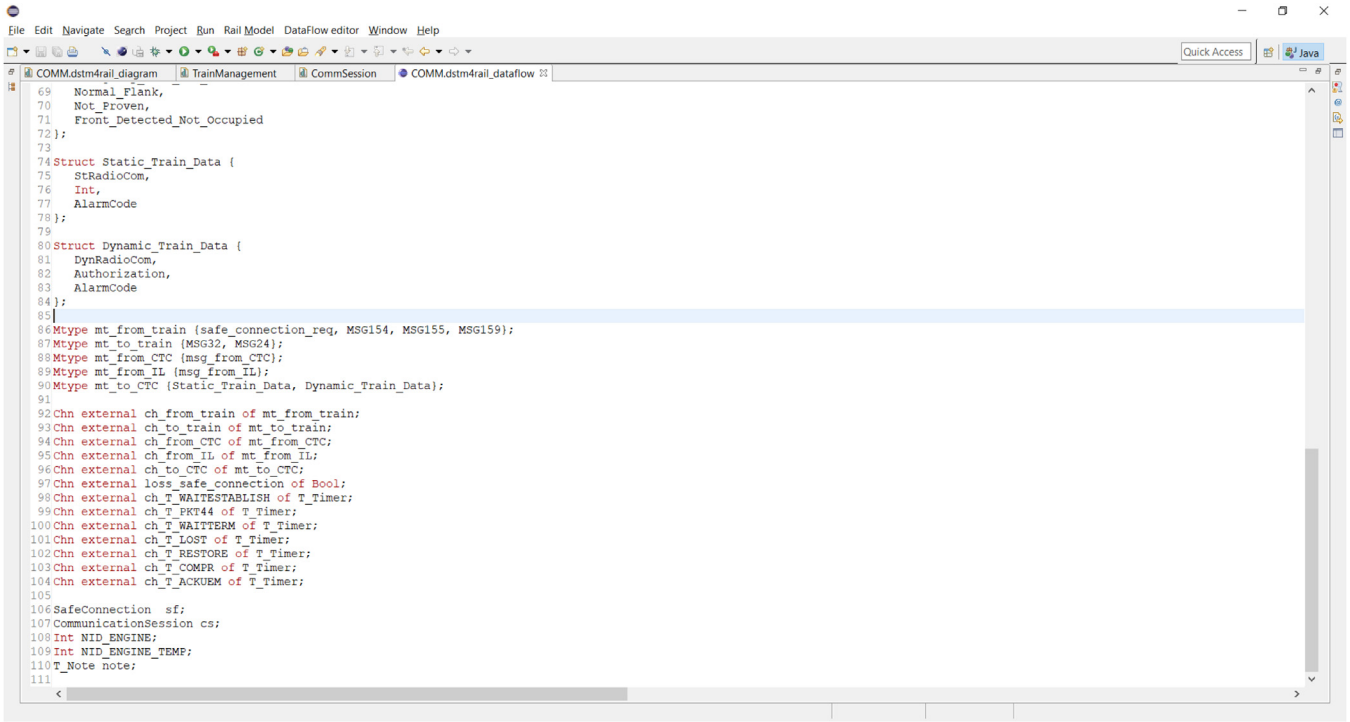

Fig. 7. Rail Model GUI - Data model.

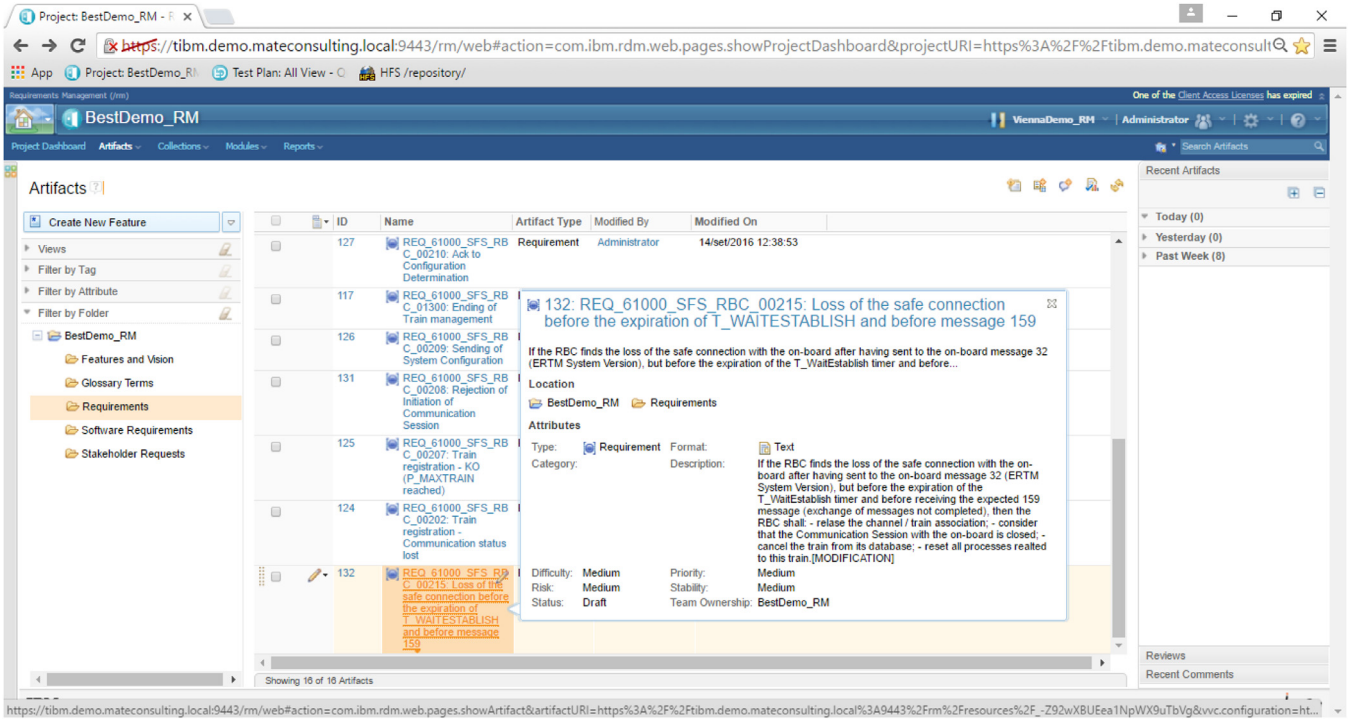

Fig. 8. Requirement view and REQ 132.

realized to get a preview of the Resource (right side of Fig. 9), as explained in Section 6.1.

Generation of test cases. Once a DSTM model has been annotated, it is ready to be fed to the test case generation process. *TCG* is invoked by *RailModel GUI*, through a suitable menu entry. When the user starts the test case generation, *RailModel GUI* retrieves the list of the existing Test Plans from RQM and asks the user to choose if he/she wants to add the new Test Cases to one of them or create a new Test Plan (Fig. 10). In the latter case, *RailModel GUI* creates the new Test Plan in RQM via the OSLC *QM Adapter*. In both cases, for each test case automatically generated, *RailModel GUI* also creates a corresponding Test Case Resource in RQM via the OSLC *QM Adapter* and inserts in it the URI of the associated test case file, which is maintained by a server, as explained in Section 5.2.

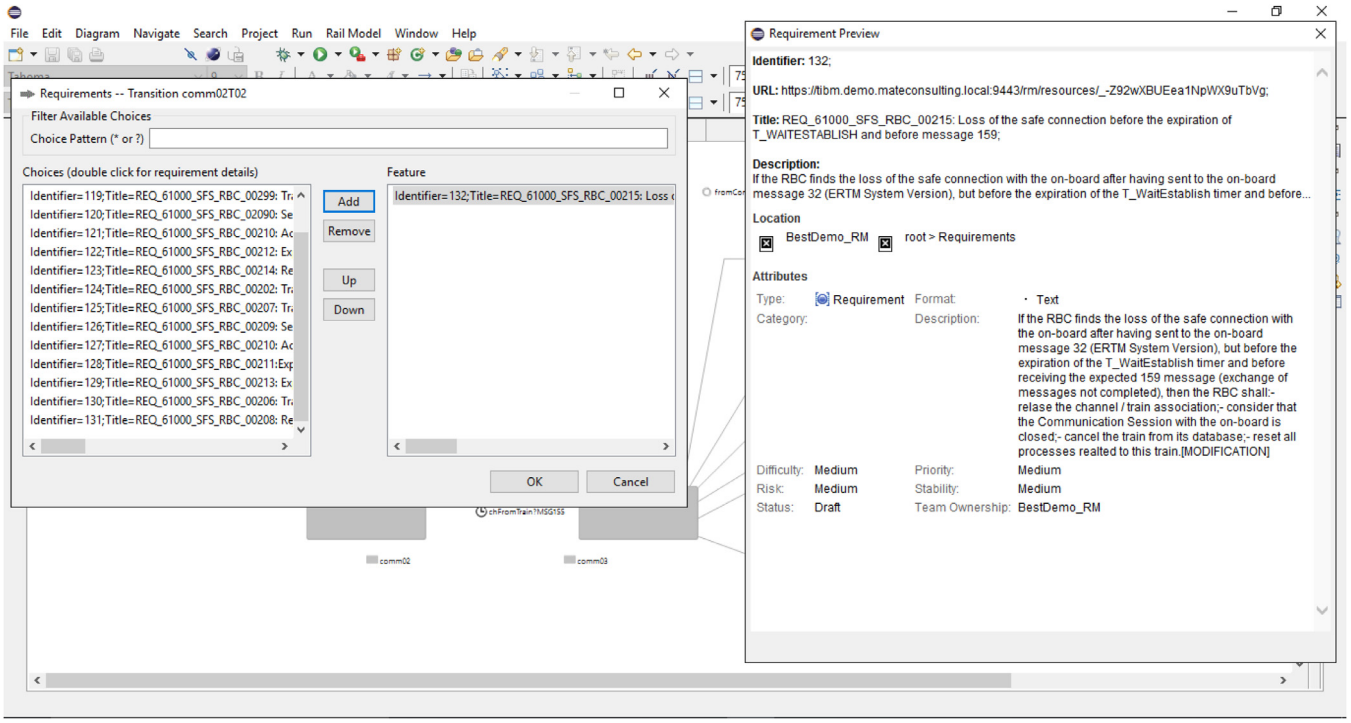

Fig. 9. Annotation of REQ 132 over transition *comm03T02*.

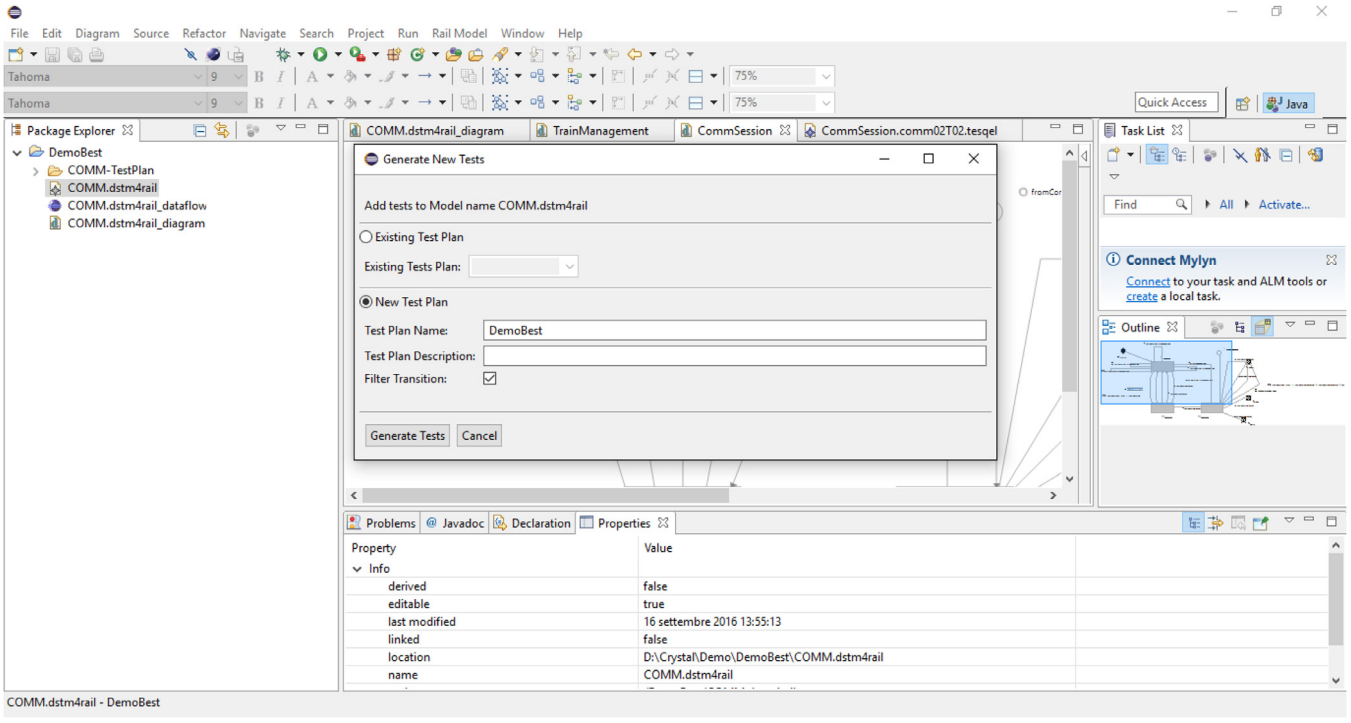

Fig. 10. Generate test cases in a new test plan.

In the current implementation, the test generation criterion focuses on transitions coverage: if a transition is annotated with at least one requirement, a corresponding test case is generated, which describes one behavior of the modeled system that starts from an initial state and leads to the execution of that transition. The set of test cases covering all the transitions of the *CommSession* machine (18 test cases) have been generated in about 3 min on a server equipped with a Intel Core 2 Duo processor and 4GB of RAM.

Fig. 11 shows a screenshot of *RailModel GUI* after the generation of the test cases. The generated test cases are automatically grouped in a folder of the project. The generated folders of test cases are shown on the left of the editor, in the view *Package Explorer*. In this folder, each test case takes the name of the last transition reached by the test itself. A single test case can be viewed in the main view of the editor, showing the details of its internal steps. In Fig. 11 the test case named *CommSession.comm02T02* (reaching the transition *comm02T02* of the machine *CommSession*

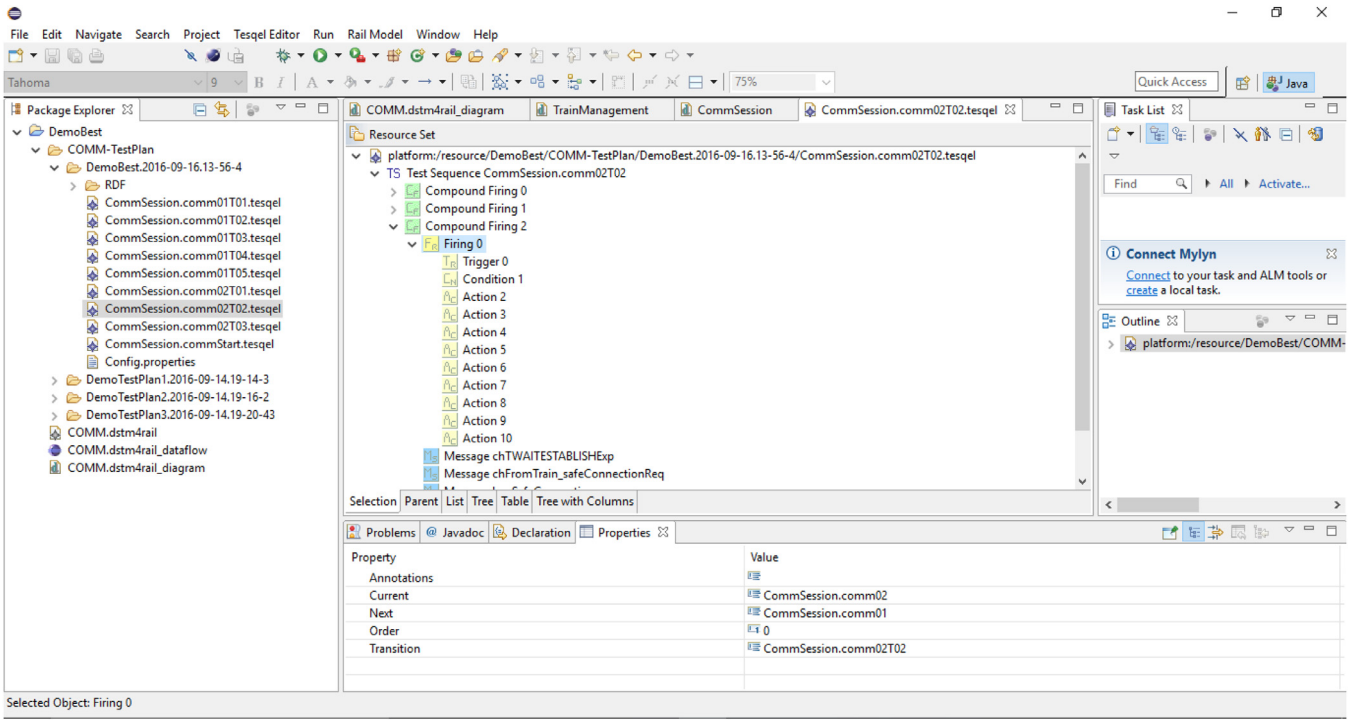

Fig. 11. Test case and details.

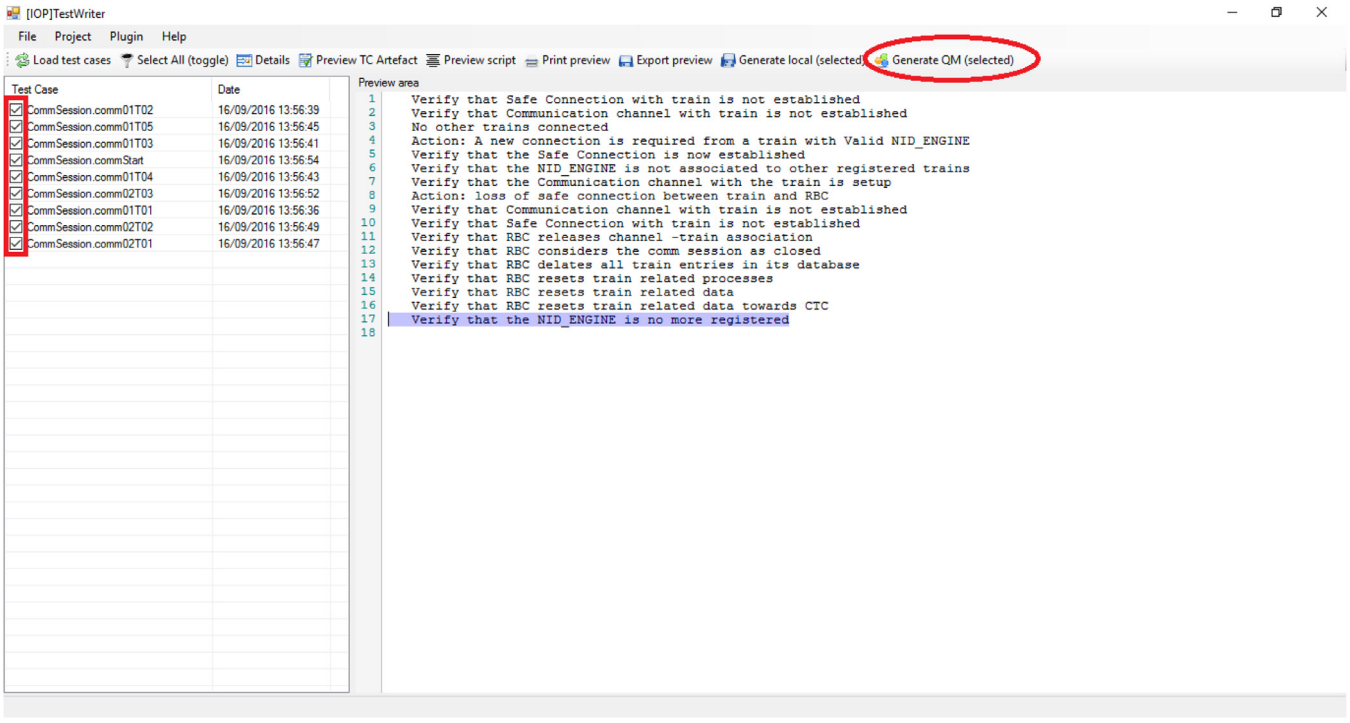

Fig. 12. IOP Test Writer - test script generation.

as last transition) is shown in details. We recall here that DSTM is a state-based formalism, allowing for concurrent execution of machines (for details on the formalism refer to the Appendix B). Hence, the internal steps of test cases, named *Compound Firings*, are group of *Firings* of a single machine transition. For each firing, the test case shows the name of the transition fired, the related current and next states. Moreover, the test data enabling the firing are given for each firing, also specifying which shall be the events and the values of variables to make true the trigger and the condition of the transition.

Generation of test scripts. Fig. 12 shows a screenshot of *IOP Test Writer* during the test script generation. For each test case, the main view shows a preview of the translation into (human-readable) test scripts. Each step of the test case is transformed into either an assert to be checked or an action to be performed on the system in order to execute the test script. The V&V engineer can

```
TEST "CommSession.COMM02T02"
EXECUTE COMM_START_TEST_PROCEDURE.prv
...
SEND_MSG $EVC KIND=ESTABLISH MSG_ID=55 DEST=RBC DATA=[(NID_ENGINE,1),...]
WAIT = 1
...
WAIT-UNTIL = 30 MSG $RBC MSG_ID=64 KIND=ACK_ESTABLISH
...
EXECUTE COMM_END_TEST_PROCEDURE.prv
ENDTEST "CommSession.COMM02T02"
```

Listing 3. Excerpt of generated PRV files.

choose to select all the test cases and generate the corresponding test scripts by clicking on the button *Generate QM (selected)*. IOP Test Writer performs the translation of the entire set of test cases and, for each test script, also creates the corresponding Test Script Resource in RQM through its OSLC *Adapter* and inserts in it the URI of the associated test script file maintained by a server. After this step, test scripts can be executed, generating the relative test logs.

Even if the generation of the concrete scripts is not the core of this paper, it is important to give some technical details to demonstrate the approach is feasible. In this concrete case, the IOP Test Writer generates the script according to the "PRV" language originally used by Ansaldo STS's engineers to manually generate test scripts. These scripts are than interpreted by a proprietary test engine — which stands a the Test Execution Environment of Fig. 3 — which interfaces with hardware-in-the-loop and/or simulted ERTMS/ETCS subsystems (e.g., RBC, EVCs, the IXLs). This interfaces deal both with stimulations (sending of test messages, changing of the values of internal variables) and verifications (waiting until some messages are not sent or until some internal variables get some values). An excerpt of the concrete test script generated by the IOP Test Writer is in Listing 3.

The generation is enabled by a configuration repository where specific strings translating the high level messages coming from the TESQUEL model are resolved allowing the IOP Test Writer to determine the correct configuration of messages and message variables according to the specific system. It is important to underline that a meaningful part of this repository can be shared among similar projects (e.g., ERTMS/ETCS messages are, bu definitions, standard). The PRV language has importing primitives (i.e., the EXECUTE command) by which common (predefined) procedures can be recalled and executed. This mechanism also allows pre-test and post-test procedures. In the practice of Ansaldo STS, such procedures are in general shared among all the tests of a functional scenario (e.g., COMM, braking, emergency management, etc.).

Analysis of test logs Test logs are properly managed by the *RTP Adapter*. Fig. 13 reports a screenshot, showing that three test cases are marked as passed (green tick), while the remaining test cases are tagged with a red "x", since no test logs have been found in the selected log folder with reference to them. Again, *RTP Adapter* creates on RQM the related Test Result Resources and inserts in each Test Result the URI of its associated log file stored in a separate server.

Production of test reports. At the end, the V&V engineer can generate synthetic tables, reporting the results of the test campaign and the traceability information between the outcomes of the test execution and the source requirements. RQM provides the users with specific functions to generate custom summary tables.

Two templates have been defined, according to which these information can be reported, as shown in Figs. 14 and 15. Fig. 14 summarizes the results of a the test campaign reporting, for each requirement, the test cases that have been completed and the corresponding results. Instead, Fig. 15 reports the requirements which are impacted by each test case. By analyzing these reports, the V&V engineer can easily obtain information on the test cases that did not pass and/or the requirements that still must be verified.

Traceability among artifacts In this paragraph we highlight how the traceability links (depicted in Fig. 5) are enforced by the realized environment among the different artifacts involved in the described case study. As previously described, the greater part of traceability links is managed by RailModel GUI and by its Adapters. In fact, RailModel GUI offers the possibility to retrieve requirements from DOORS and annotate them over each model element (i.e., over each state or transition) of the *DSTM model*. As shown in Fig. 9, the requirement REQ 132 (which details are stored in DOORS) has been annotated over the transition *comm03T02* of the CommSession machine in the case study. Moreover, RailModel GUI handles also the traceability links related to the test cases. In the described case study, when the user asks to create a new test plan during the generation of the test cases, RailModel GUI automatically creates the proper QM-Resources (i.e., *Test Plan* and a set of *Test Cases*) in RQM and links them with the related attachments (i.e., *TESQEL test cases*), also uploading these last in a central server. At last, RailModel GUI also handles the traceability between *Test Case*s and *Requirement*s and between *TESQEL test case*s and the source *DSTM model*. In fact, after the generation of test cases, RQM is populated with the resources representing test cases, which are linked properly to requirements (as demonstrated by Figs. 14 and 15).

The traceability links between *Test Case*s and *Test Script*s, and between *Test Script*s and the *IOP Script*s are instead managed by IOP Test Writer. In fact, after the generation of test scripts (depicted in Fig. 12), the tool automatically creates the related resources in RQM. Similarly, RTP Adapter handles the traceability links between *Test Result*s and *Test Script*s and between *Test Result*s and *Test Log*s. Resources are created once again in RQM, after the analysis of logs. In fact, at the end of the entire process, RQM contains all the data and the information needed to produce reports, as shown in Figs. 14 and 15.

9. Related work

The advent of Industry 4.0 and Internet of Things (IoT) has introduced new challenges in the products life-cycle management. The increasing complexity of requirements related to the demand of integrated software and services raises the need to meet both customers demand and industry strategies (Song, 2017), so asking for the integration of PLM and ALM features. Integrated PLM-ALM frameworks allow to link the life-cycles of hardware products, software applications and data modules; this leads to better planning the development, testing and release activities, to reduce specification errors and to save time and effort for development, especially in case of late changes of the initial requirements (Ebert, 2013). Big companies as well as small and medium enterprises can benefit from the integration of ALM-PLM frameworks in terms of automation of the business processes, communication, reduction of failure costs, enhancement of traceability. This last aspect is es-

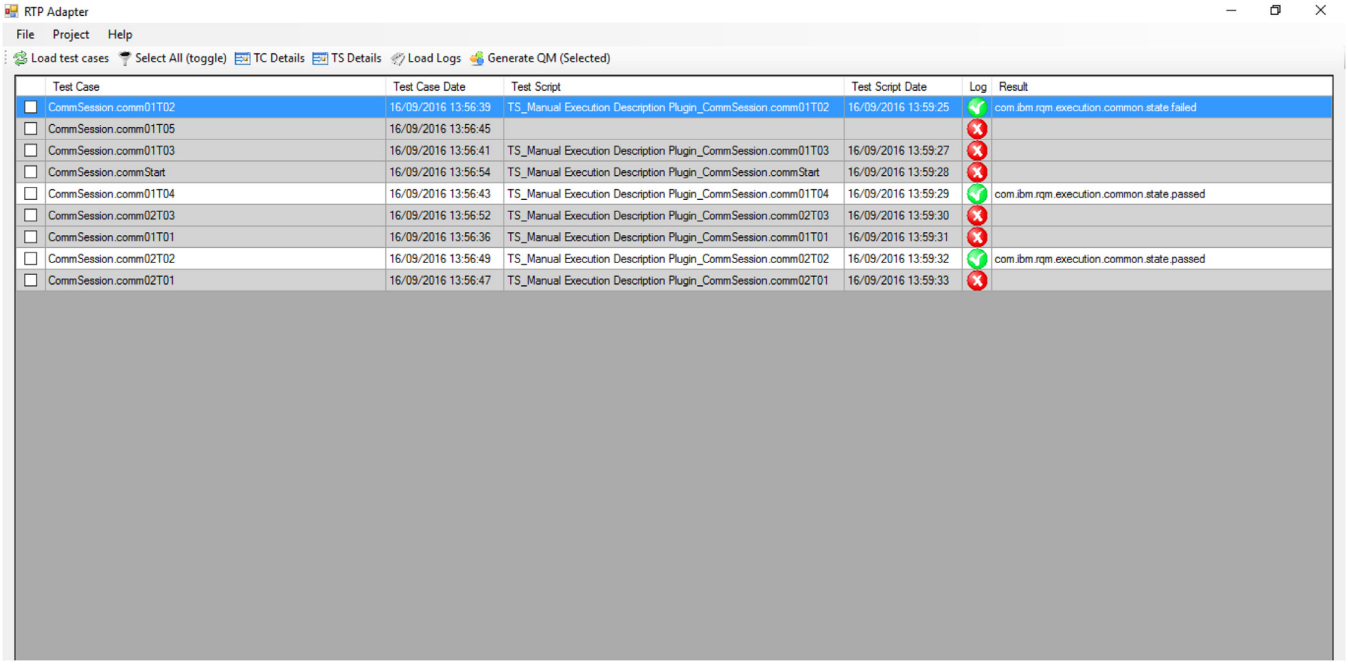

Fig. 13. RTP Adapter - test log analysis.

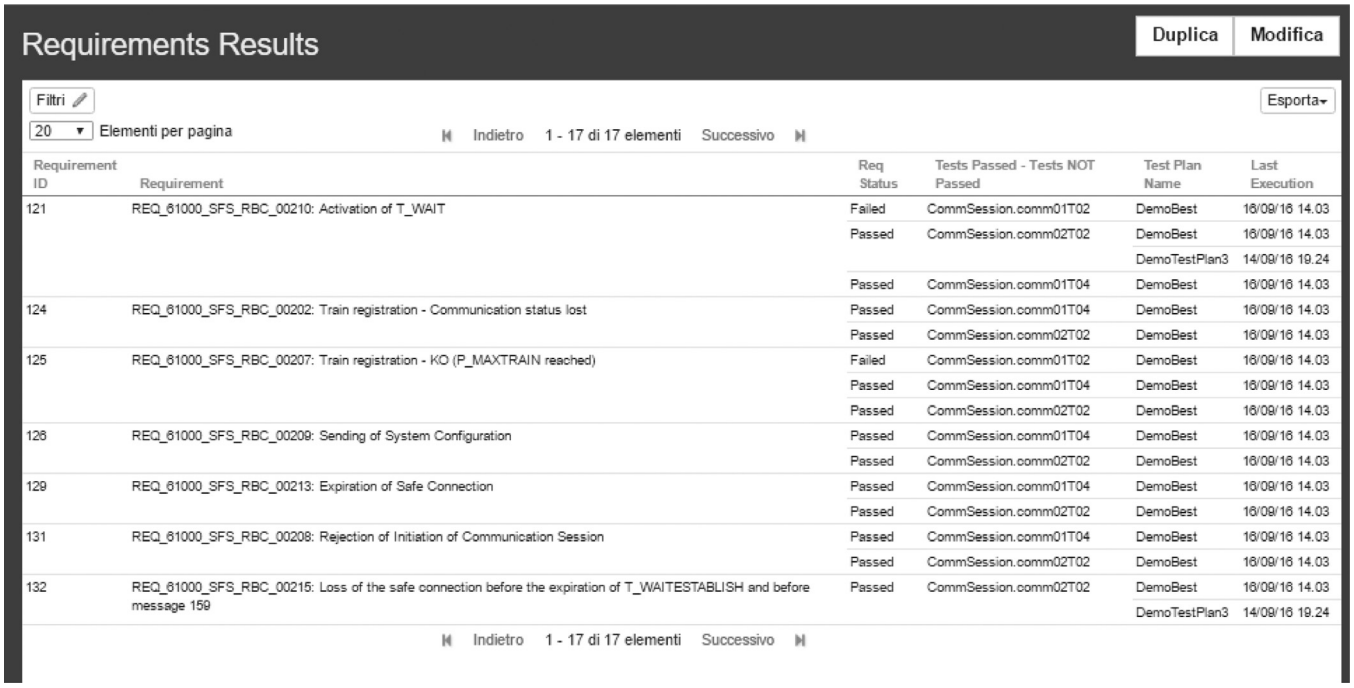

Fig. 14. IBM RQM - requirement report.

sential to evaluate the satisfaction of the requirements and the quality of the final product. Recent academic and industrial research trends, demonstrate that there is a wide interest in the PLM-ALM integration. The work in Dekhtiar et al. (2018) defines an approach to integrate deep learning algorithms into an existing PLM tool in the manufacturing domain. The goal is to develop a decision support system able to support the visual inspection of mechanical parts, based on the previous knowledge collected in thousand item CADs and operational datasets. The approach described in Essamlali et al. (2017) integrates concurrent engineering principles into PLMs to improve the efficiency of the life-cycle management for the development of wearable smart products, with the aim to reduce the development effort and place the product on the market as soon as possible. The work in Deuter et al. (2018), instead, focuses on the requirements to be satisfied to integrate an ALM tool and the existing PLM of an electrical manufacturing company. Finally, Siemens acquired the Polarion ALM tool[17] to integrate ALM concepts in existing processes of the automotive domain.

A common principle emerging from the literature review is that the integration between PLM and ALM can be facilitated by the definition of a shared model of the system. Model-based ap-

[17] https://polarion.plm.automation.siemens.com/products/polarion-alm

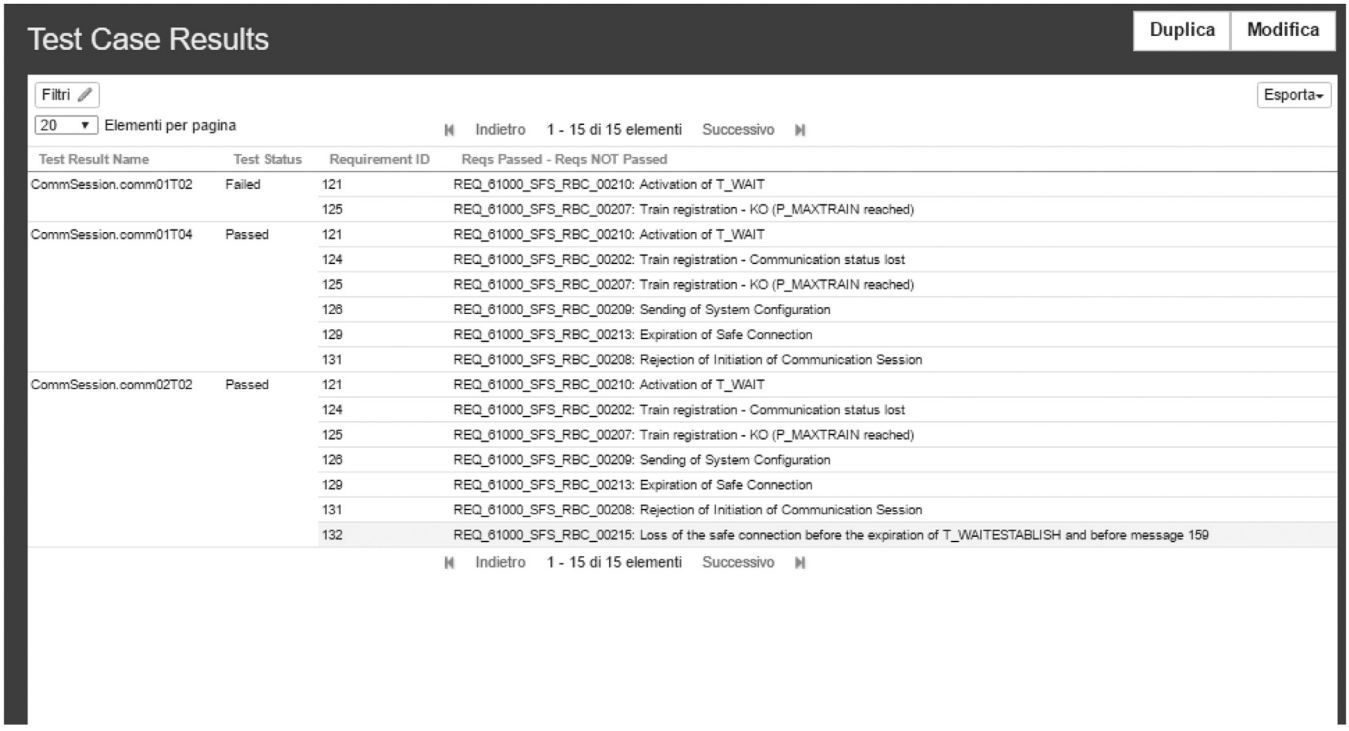

Fig. 15. IBM RQM - test report.

proaches cope with the system complexity by allowing to work at an high level of abstraction and by facilitating the system development, with an overall enhancement of efficiency and quality (Ebert, 2013). In this context, PLM-ALM environments supporting Model-Based testing (MBT) are highly desirable.

The effort spent on testing *critical systems* usually accounts for more than fifty percent of the total development costs (Sommerville, 2006) since a missing or poor testing activity may impact on a quota close to eighty percent of the overall system costs (Tassey, 2002). MBT can reduce such effort by deriving test cases from a specification of the System Under Test (SUT) (Utting and Legeard, 2007). Several MBT approaches have been assessed and implemented to support automated test generation (Zander et al., 2011; de Niz et al., 2006; Dias Neto et al., 2007; Anand et al., 2013) and a number of commercial tools are currently available from different vendors (e.g., Ansys,[18] Conformiq,[19] MathWorks,[20] Systemite[21]). In PLM-ALM environment the integration of MBT can be achieved in different ways: through common data representations (e.g., using standard languages such as UML and standard UML formats, such as XMI) (Thomas and Nejmeh, 1992), by providing a common middleware for data and control integration (e.g., ModelBus Hein et al., 2009) or by designing for extensibility (e.g., the Eclipse platform). Two complementary strategies can be adopted: the horizontal and the vertical. The horizontal integration pursues tools inter-operation on one among the three main aspects of ALM (governance, development and maintenance), by sharing platform guidelines and common components (e.g., Eclipse and Microsoft Visual Studio). Vertical tools integration allows for cooperation across the life-cycle and provides solutions specific to a domain, an industry or a group of stakeholders, by sharing data, control and presentation layers (e.g., Microsoft Visual Studio, IBM Rational and IBM's Jazz initiative[22]). Public APIs or plug-ins may be provided in order to support the development of extensions. However, the increasing complexity of modern critical systems requires many specialized tools, which have to inter-operate to manage and perform the activities throughout the life-cycle, beyond the development of point-to-point integrations and proprietary bridges or platforms. The need for more efficient and effective approaches to seamless tools interoperability is leading to the definition and the adoption of new standards and protocols. The Open Services for Life-cycle Collaboration (OSLC) initiative[23] is a joint effort between the academia and the industry to develop specifications for tools inter-operation, based on assessed and standardized web technologies and W3C Linked Data. Several works describe the use of OSLC (Aichernig et al., 2014; Seceleanu and Sapienza, 2013; Marko et al., 2015; Kaiser and Herbst, 2015; El Salloum, 2015). However, only a few of them focus on test automation and test case generation. In Marinescu et al. (2013), the design and implementation of an OSLC adapter is proposed for achieving tool interoperability in deriving executable tests from East-ADL architectural models. Requirements engineering, analysis and test case generation are combined in the methodology presented in Aichernig et al. (2014) and its supporting tool MoMuT: the integration of MoMuT and a requirement management tool (e.g., IBM Rational DOORS) is realized via OSLC. Both these works were developed within the ARTEMIS JU project MBAT (Nielsen, 2014), which, in turn, partially relied on the results of CESAR (Jolliffe, 2010), whose objective was the definition of a framework to facilitate tighter integration among different tools and enhanced traceability. In this direction, CRYSTAL defined a Reference Technology Platform (RTP) and a harmonized interoperability specification (IOS), incorporating various open specifications and standards, such as OSLC. Concerning these works, this paper focuses on the definition and the realization of a testing environment supporting all the phases of a testing process, from the

[18] http://www.esterel-technologies.com/products/scade-suite
[19] https://www.conformiq.com/products
[20] https://it.mathworks.com/discovery/model-based-testing
[21] https://www.systemweaver.se
[22] https://jazz.net/products/clm
[23] http://open-services.net/specifications/core-2.0

generation of test cases to the production of test reports. The reference architecture introduced in Section 5 proposes a solution, whose implementation can be compliant with the OSLC specifications.

10. Conclusion and future work

This paper has presented a reference architecture and a specific implementation of an OSLC-based interoperable environment for system-level functional testing of ERTMS/ETCS controllers. The work described in the paper has been conducted within the ARTEMIS project CRYSTAL. An application of the realized implementation to concrete activities, daily performed in an industrial setting, has been provided to exemplify the usage of the environment, demonstrate the feasibility of the approach and give evidence of the readiness level reached by the software framework. As for the testing automation, it relies on a model-driven approach and model checking. A chain of transformations hides to the user all the steps needed to derive the test cases from an initial description of the SUT behavior. As to the proposed architecture, data sharing and tools inter-operation is obtained by the usage of web technologies and OSLC, a recent standard for enabling interoperability and integration of life-cycle tools, so illustrating how to exploit linked data and the specifications provided by OSLC to cope with a concrete case in the railway domain.

Future work will address two main points of improvement with respect to the current implementation.

The first pertains to the management of the attachments, in fact the solution here adopted is functionally effective but it is not fully compliant with the philosophy underlying OSLC. This limit has been mentioned in the discussion and in Section 5 where the actual solution is described.

The second improvement is the integration in the testing framework of tools for data analysis on test failures. Providing the framework with the capability of performing mining on logs and historical data series to analyze the trends of testing failures was not taken into account at first. This can be now considered with the aim of driving the generation and the execution of specific test cases, so improving the quality of the process while shortening the validation time.

Acknowledgments

This paper has been partially supported by research project CRYSTAL (Critical System Engineering Acceleration), funded from the ARTEMIS Joint Undertaking under grant agreement no 332830 and from ARTEMIS member states Austria, Belgium, Czech Republic, France, Germany, Italy, Netherlands, Spain, Sweden, United Kingdom. The work of some authors is also supported by Department of Electrical Engineering and Information Technology (DIETI) of University of Naples Federico II under the project MODAL (*MOdel-Driven AnaLysis of Critical Industrial Systems*).

Appendix A. A Glossary of the enabling technologies

A strength of the proposed architecture is that it takes the best of breed of technologies adopted in academic and industrial contexts. This section lists and defines languages, frameworks and tools referred throughout the paper. Fig. A.16 provides a conceptual map of technologies and tools according to the context in which they have been used to implement the testing framework: specification, programming, tool integration and analysis. In addition, the map also includes third party tools.

1. *Specification*
   - *EBNF* (Extended Backus–Naur form) is used in the definition of Dynamic STate Machines (DSTMs).
   - *Ecore* (Steinberg et al., 2008) is a meta-model included in the Eclipse Modeling Framework (EMF) for describing models and run-time support for the models, used in the development of all the languages involved in the test automation process.
   - *JSON* (JavaScript Object Notation) is an open-standard file format to specify the format of the data exchanged between the web services implementing the proposed architecture.
   - *Promela* (Holzmann, 2004) is the specification language of the Spin model checker used to automate the test cases generation process.
   - *XMI* (XML Metadata Interchange) is the well-know OMG standard for exchanging meta-data information in XML format used to exchange *models* between different frameworks and tools.
2. *Programming*
   - *ATL* (Atlas Transformation Language) (Jouault and Kurtev, 2005) is a model transformation language developed on top of the Eclipse platform to derive a set of target models from a set of source models. It is used to realize the transformations chain from DSTM test models to test cases.
   - *Eclipse* is a powerful Integrated Development Environment (IDE), used both as a development environment and as an integration platform to develop the testing framework.
   - *EMF* (Eclipse Modeling Framework) (Steinberg et al., 2008) is a stable Eclipse framework which provides code generation facilities for building toolsets and Java applications from model specifications described in XMI.
   - *GMF* (Graphical Modeling Framework) is an eclipse-based framework which provides facilities to generate graphical editors from Ecore-based model specifications.
   - *LINQ* (Language-Integrated Query) is a component of the Microsoft.NET framework, used to develop *IOP Test Writer* and *RTP Adapter*.
   - *Log4Net* is part of the Apache Logging Services project, used to perform logging activities for *IOP Test Writer* and *RTP Adapter*.
   - *SableCC* (Gagnon and Hendren, 1998) is a parser generator used in the development of *DSTM Verifier*.
3. *Integration*
   - *Eclipse Lyo* is an Eclipse SDK introduced to support the development of OSLC-compliant tools.
   - *OSLC* (Open Services for Lifecycle Collaboration) is a set of open specifications for tools integration in support of end-to-end life-cycle processes through the integration of data and workflows.
   - *OSLC4Net* is an SDK used in.NET development to support the development of OSLC-compliant tools used to implement the OSLC integration with IBM Rational Doors and Quality Manager.
   - *REST* is an architectural style for the development of distributed systems adopted to develop the part of the framework based on web services.
4. *Analysis*
   - *Spin* Holzmann (2004) is a powerful model checking tool, widely used for formal verification of concurrent and distributed systems and adopted in the proposed architecture to generate the test sequences from Promela models.
5. *External tools*
   - *IBM Rational Doors* is a client-server framework for the management of requirements usually adopted to increase collaboration and traceability within an enterprise. DOORS was used as a OSLC Requirement Management Provider.
   - *IBM Rational Quality Manager* is a test management tool. It allows to store test cases and the outcomes of their execu-

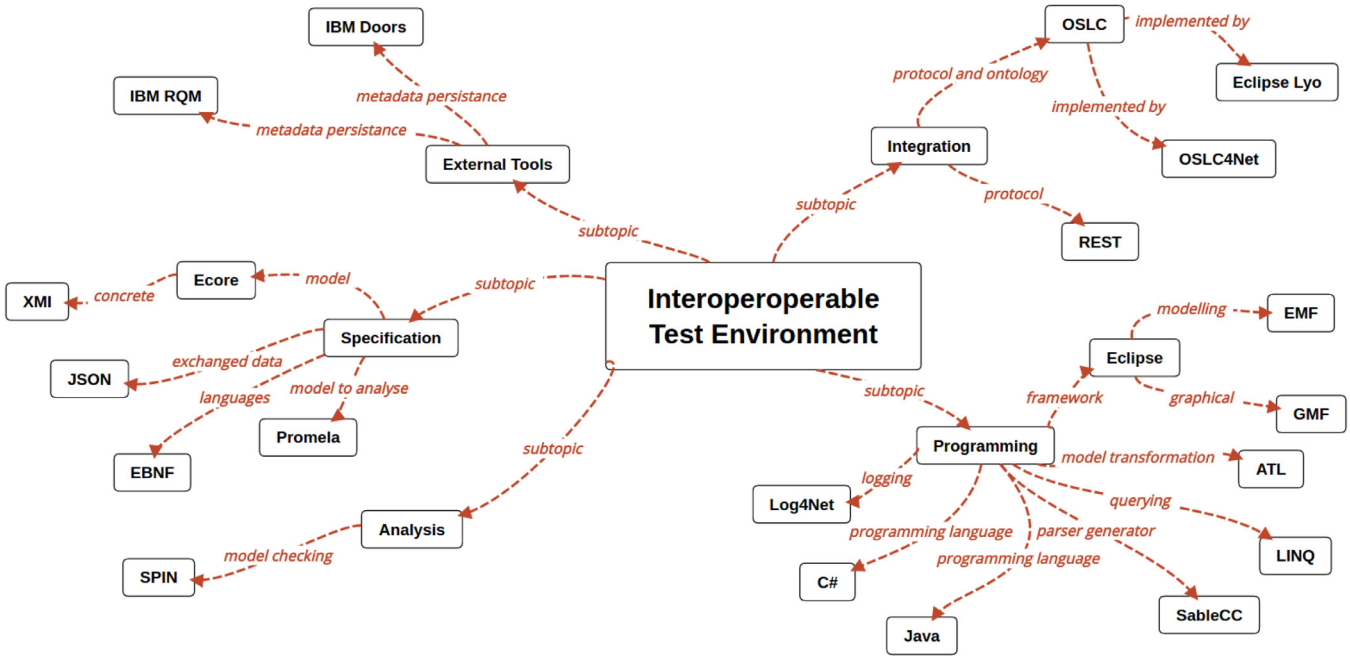

Fig. A.16. Conceptual map of the adopted technologies.

tion, and provides facilities to trace results on the system requirements. RQM has been used as OSLC Quality Management Provider.

Appendix B. The DSTM formalism

This section provides an overview of the DSTM formalism, adopted in the *RailModelGUI* to produce the system model. For a complete account of the formal syntax and semantics of DSTM we refer to Benerecetti et al. (2017).

A Dynamic STate Machine (DSTM) model is a sequence of machines communicating over a set of global variables and a set of (typed and buffered) global communication channels. The communication channels are grouped into internal and external channels. The former are channels where machines can send and read (consuming or non consuming) the message. The latter have a similar behavior but, if it is empty for a step, a message can be non-deterministically chosen from the set of possible messages compliant with the channel data structure. This mechanism simulates the presence of an external environment that can send messages to the modeled system with any value in any sequence.

One machine in a DSTM model is the *initial* (i.e., main) machine, namely the initial process of the model. Each machine, with except of the initial one, may be parametric. Parameters are aliases for channels and variables names and are actualized when the machine is instantiated, allowing also for multiple instantiations of the same machine with different parameter values. Each machine is defined as a state-transition diagram. With respect to traditional formalisms, DSTM defines a novel semantics for the fork and join operators. In fact, a fork allows for the instantiation of machines either in a *synchronous* of in an *asynchronous* way. In former case, the forking machine is suspended and waits for the activated processes to terminate, while in the latter the forking machine continues its activity, executing concurrently to the newly-activated processes. Consequently, a join closes an instantiation of machines, and synchronizes the control flows. A transition connects a source and a destination vertex in a machine; it is decorated with a *trigger* (an input event originating from the external environment or from other machines, e.g. the presence of messages on a given channel), a *guard* (a Boolean condition on the current contents of variables and channels) and an *action* (one or more statements on variables and channels). A transition fires if its trigger is fulfilled and its guard satisfied by the current status of channels and variables. When a transition fires, its action is executed with possible side-effects.

DSTM defines a complete type system, which is used to define types of variables and of channel messages. Three types are defined: *basic types, compound types* and *multi-types*. The *basic types* includes the Boolean and the integer types, the channel type (i.e., *Chn* for channel names, and a set of user-defined enumeration types. *Compound types* are tuples of *basic types*, so defining data structures. A *multi-type* is a type defined as composition of both basic and compound types. Variables can be typed as basic or compound types; similarly, internal channels can convey only messages of basic or compound types, while external channels can convey also multi-type messages. This last mechanism is used to model for the presence of an external environment that can send messages to the modeled system of a different type.

The evolution of a DSTM consists in a sequence of instantaneous reactions called *steps*. A step is a maximal set of transitions that are triggered by the current system state and by the current value of channels. The firing of a transition can have side effects on the available channels and variables. The content sent during a step on an external channel, unlike for internal ones, can only be observed in the next step. As previously anticipated, if an external channel is empty at the end of a step, the semantics allows for the generation of a random message, non-deterministically chosen among the possible set of messages compliant with the channel type (possibly, multi-type).

References

Aichernig, B.K., Hörmaier, K., Lorber, F., Nickovic, D., Schlick, R., Simoneau, D., Tiran, S., 2014. Integration of requirements engineering and test-case generation via OSLC. In: Quality Software (QSIC), 2014 14th International Conference on. IEEE, pp. 117–126.

Anand, S., Burke, E.K., Chen, T.Y., Clark, J., Cohen, M.B., Grieskamp, W., Harman, M., Harrold, M.J., Mcminn, P., Bertolino, A., et al., 2013. An orchestrated survey of

methodologies for automated software test case generation. J. Syst. Softw. 86 (8), 1978–2001.
Baier, C., Katoen, J., 2008. Principles of Model Checking. MIT Press.
Barberio, G., Martino, B.D., Mazzocca, N., Velardi, L., Amato, A., Guglielmo, R.D., Gentile, U., Marrone, S., Nardone, R., Peron, A., Vittorini, V., 2014. An interoperable testing environment for ERTMS/ETCS control systems. In: Computer Safety, Reliability, and Security - SAFECOMP 2014 Workshops: ASCoMS, DECSoS, DEVVARTS, ISSE, ReSA4CI, SASSUR. Florence, Italy, September 8–9, 2014. Proceedings, pp. 147–156.
Baybutt, P., 2014. The alarp principle in process safety. Process Saf. Progress 33 (1), 36–40.
Belkadi, F., Troussier, N., Eynard, B., Bonjour, E., 2010. Collaboration based on product lifecycles interoperability for extended enterprise. Int. J. Interact. Des.Manuf. 4 (3), 169–179.
Benerecetti, M., De Guglielmo, R., Gentile, U., Marrone, S., Mazzocca, N., Nardone, R., Peron, A., Velardi, L., Vittorini, V., 2017. Dynamic state machines for modelling railway control systems. Sci. Comput. Programm. 133, 116–153. doi:10.1016/j.scico.2016.09.002.
Benerecetti, M., Gentile, U., Marrone, S., Nardone, R., Peron, A., Starace, L.L.L., Vittorini, V., 2019. Dynamic state machines for modelling railway control systems. In: Proc. of 26th International SPIN Symposium on Model Checking of Software (SPIN), 11636.
Boroday, S., Petrenko, A., Groz, R., 2007. Can a model checker generate tests for non-deterministic systems? Electron. Not. Theor. Comput. Sci. 190 (2), 3–19.
CENELEC EN 50128, 2012. Railway applications - communication, signalling and processing systems - software for railway control and protection systems. Book EN 50128.
Dekhtiar, J., Durupt, A., Bricogne, M., Eynard, B., Rowson, H., Kiritsis, D., 2018. Deep learning for big data applications in cad and plm, research review, opportunities and case study. Comput. Ind. 100, 227–243. doi:10.1016/j.compind.2018.04.005.
Deuter, A., Otte, A., Ebert, M., Possel-Dölken, F., 2018. Developing the requirements of a PLM/ALM integration: an industrial case study. In: Intelligent, Flexible and Connected Systems in Products and Production - 4th International Conference on System-Integrated Intelligence, 24, pp. 107–113. doi:10.1016/j.promfg.2018.06.020.
Dias Neto, A.C., Subramanyan, R., Vieira, M., Travassos, G.H., 2007. A survey on model-based testing approaches: a systematic review. In: Proceedings of the 1st ACM international workshop on Empirical assessment of software engineering languages and technologies: held in conjunction with the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE) 2007. ACM, pp. 31–36.
Ebert, C., 2013. Improving engineering efficiency with plm/alm. Softw. Syst. Model. 12 (3), 443–449. doi:10.1007/s10270-013-0347-3.
El Salloum, C., 2015. Seamless integration of test information management and calibration data management in the overall automotive development process. In: 2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST), pp. 1–3. doi:10.1109/ICST.2015.7102629.
Essamlali, M.T.E., Sekhari, A., Bouras, A., 2017. Product lifecycle management solution for collaborative development of wearable meta-products using set-based concurrent engineering. Concurr. Eng. 25 (1), 41–52. doi:10.1177/1063293X16671386.
Fielding, R.T., 2000. Architectural Styles and the Design of Network-based Software Architectures. AAI9980887
Flammini, F., Marrone, S., Mazzocca, N., Nardone, R., Vittorini, V., 2012. Model-driven V&V processes for computer based control systems: A unifying perspective. In: Leveraging Applications of Formal Methods, Verification and Validation. Applications and Case Studies - 5th International Symposium, ISoLA 2012, Heraklion, Crete, Greece, October 15–18, 2012, Proceedings, Part II, pp. 190–204.
Gagnon, E.M., Hendren, L.J., 1998. Sablecc, an object-oriented compiler framework. In: TOOLS 1998: 26th International Conference on Technology of Object-Oriented Languages and Systems, 3–7 August 1998, Santa Barbara, CA, USA, pp. 140–154.
Gargantini, A., Heitmeyer, C., 1999. Using model checking to generate tests from requirements specifications. SIGSOFT Softw. Eng. Not. 24 (6), 146–162. doi:10.1145/318774.318939.
Gentile, U., Marrone, S., Mele, G., Nardone, R., Peron, A., 2014. Test specification patterns for automatic generation of test sequences. In: Formal Methods for Industrial Critical Systems - 19th International Conference, FMICS 2014, Florence, Italy, September 11–12, 2014. Proceedings, pp. 170–184.
Hein, C., Ritter, T., Wagner, M., 2009. Model-driven tool integration with modelbus. In: Workshop Future Trends of Model-Driven Development, pp. 50–52.
Holzmann, G.J., 2004. The SPIN Model Checker - Primer and Reference Manual. Addison-Wesley.
Jolliffe, G., 2010. Cost-efficient methods and processes for safety relevant embedded systems (CESAR) - an objective overview. In: Making Systems Safer - Proceedings of the Eighteenth Safety-Critical Systems Symposium, Bristol, UK, February 9–11, 2010, pp. 37–50.
Jouault, F., Kurtev, I., 2005. Transforming models with atl. In: International Conference on Model Driven Engineering Languages and Systems, pp. 128–138.
Kaiser, C., Herbst, B., 2015. Smart engineering for smart factories: How OSLC could enable plug & play tool integration. In: Mensch und Computer 2015 - Workshopband, Stuttgart, Germany, September 6–9, 2015, pp. 269–278.
Lacheiner, H., Ramler, R., 2011. Application lifecycle management as infrastructure for software process improvement and evolution: experience and insights from industry. In: Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on. IEEE, pp. 286–293.
Leitner, A., Herbst, B., Mathijssen, R., 2016. Lessons learned from tool integration with OSLC. In: Information and Software Technologies - 22nd International Conference, ICIST 2016, Druskininkai, Lithuania, October 13–15, 2016, Proceedings, pp. 242–254.
Marinescu, R., Saadatmand, M., Bucaioni, A., Seceleanu, C., Pettersson, P., 2013. East-adl Tailored Testing: From System Models to Executable Test Csases. Technical Report. Mälardalen University, Technical Report ISSN 1404-3041 ISRN MDH-MRTC-278/2013-1-SE.
Marko, N., Leitner, A., Herbst, B., Wallner, A., 2015. Combining xtext and oslc for integrated model-based requirements engineering. In: 2015 41st Euromicro Conference on Software Engineering and Advanced Applications, pp. 143–150.
Nardone, R., Gentile, U., Benerecetti, M., Peron, A., Vittorini, V., Marrone, S., Mazzocca, N., 2015. Modeling railway control systems in Promela. In: Formal Techniques for Safety-Critical Systems - Fourth International Workshop, FTSCS 2015, Paris, France, November 6–7, 2015. Revised Selected Papers, pp. 121–136.
Nardone, R., Gentile, U., Peron, A., Benerecetti, M., Vittorini, V., Marrone, S., De Guglielmo, R., Mazzocca, N., Velardi, L., 2014. Dynamic state machines for formalizing railway control system specifications. In: International Workshop on Formal Techniques for Safety-Critical Systems. Springer, pp. 93–109.
Nielsen, B., 2014. Towards a method for combined model-based testing and analysis. In: 2014 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD), pp. 609–618.
de Niz, D., Bhatia, G., Rajkumar, R., 2006. Model-based development of embedded systems: The sysweaver approach. In: 12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2006), 4–7 April 2006, San Jose, California, USA, pp. 231–242.
OASIS, 2013. Last access: 2017-11-17. Open Services for Lifecycle Collaboration core specification version 2.0.
Offutt, J., Liu, S., Abdurazik, A., Ammann, P., 2003. Generating test data from state-based specifications. Softw. Test. Verif. Reliab. 13 (1), 25–53. doi:10.1002/stvr.264.
Ray, M., Patnaik, S., Pradhan, S., 2019. Coverage criteria for state-based testing: a systematic review. Int. J. Inf. Technol. Proj. Manag. 10 (1), 1–20. doi:10.4018/IJITPM.2019010101.
Seceleanu, T., Sapienza, G., 2013. A tool integration framework for sustainable embedded systems development. IEEE Comput. 46 (11), 68–71.
Sommerville, I., 2006. Software Engineering: (Update) (8th Edition) (International Computer Science). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
Song, W., 2017. Requirement management for product-service systems: status review and future trends. Comput. Ind. 85, 11–22. doi:10.1016/j.compind.2016.11.005.
Stark, J., 2015. Product Lifecycle Management. In: Product Lifecycle Management. Springer, pp. 1–29.
Steinberg, D., Budinsky, F., Merks, E., Paternostro, M., 2008. EMF: Eclipse Modeling Framework. Pearson Education.
Summers, A.E., 1998. Techniques for assigning a target safety integrity level. ISA Trans. 37 (2), 95–104.
Tassey, G., 2002. The economic impacts of inadequate infrastructure for software testing. TheEconomic Impactsof Inadequate Infrastructure for Software Testing. Cited By 642
Thomas, I., Nejmeh, B.A., 1992. Definitions of tool integration for environments. IEEE Softw. 9 (2), 29–35.
UIC, 2008. ERTMS/ETCS Application Level 2 - Safety Analysis. Part 1 - Functional Fault Tree. Ref. SUBSET-088 part 1 issue 2.3.0.
Utting, M., Legeard, B., 2007. Practical Model-Based Testing: A Tools Approach. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
Zander, J., Schieferdecker, I., Mosterman, P.J., 2011. Model-Based Testing for Embedded Systems, 1st ed. CRC Press, Inc., Boca Raton, FL, USA.

Roberto Nardone received his M.D. and Ph.D. in Computer Engineering from the University of Naples Federico II, Naples, Italy, in 2009 and 2013, respectively. From 2018, he is an Assistant Professor at Mediterranean University of Reggio Calabria. He was a post-doctoral researcher at University of Naples Federico II. His research interests include quantitative evaluation of non-functional properties, with a particular focus on dependability and performability assessment and threat propagation analysis, by means of model-based and model-driven techniques. He is involved in research projects with both academic and industrial partners.

Stefano Marrone is an assistant professor in Computer Engineering at Seconda Università di Napoli, Italy. His interests include the definition of model driven processes for the design and the analysis of transportation control systems, complex communication networks and critical infrastructures. He is involved in research projects with both academic and industrial partners.

Ugo Gentile is a Post-Doctoral researcher in the Engineering Department at CERN (Geneva, Switzerland). He got is PhD in Computer and Automation Engineer at the University of Naples Federico II focusing his research on verification and validation of safety-critical systems. His main research topics include the application of machine and deep learning for the data analysis of complex industrial systems and the application of model-driven principles to support the life cycle of ICT systems and infrastructures. He has authored different publications in international peer-reviewed journals and he has been actively involved in different FP7 international projects as MASSIF, CRYSTAL and SVEVIA.

Aniello Amato is a software engineer in Mate Consulting, a digital technology company. After graduating in computer science at University of Salerno, he was chosen for a research position for the Modern project (Architectural Models for the Definition, Execution and Reconfiguration of User-Centric Processes in Enterprise 2.0). One of the most challenging work experience is the CRYSTAL project (CRitical sYSTem engineering AcceLeration). Currently, he is working on different projects for the development of web-based applications.

Gregorio Barberio is the head of technology for Business Application department at Mate Consulting, a digital and technology consultancy company. He has more than 20 years of experience in IT industry and a huge experience in leadership. Currently, he is coordinating and working on different research and development projects.

Massimo Benerecetti is currently serving as Associate Professor in Computer Science at the Department of Electrical Engineering and Information Technologies of the Università di Napoli Federico II. His main research topics include Formal Verification and Synthesis of reactive, real-time and cyber-physical systems, Logics for Computer Science, Artificial Intelligence and Multi-Agent Systems.

Renato De Guglielmo is the Head of the RAMS Tools Unit in Hitachi Rail STS. He has more than 10 years of experience in software development for railway controllers. He got his degree in Computer Engineering from the University of Naples Federico II in 2006. He has been involved in different European research projects aiming at improving the efficiency of software tools in the railway lifecycle.

Beniamino Di Martino is Full Professor at University of Campania, Engineering Dept. He is author of 14 international books and more than 300 publications in international journals and conferences. He has been Coordinator of EU funded FP7-ICT Project mOSAIC, and participates to various international research projects. He is Editor / Associate Editor of seven international journals and EB Member of several international journals. He is vice Chair of the Executive Board of the IEEE CS Technical Committee on Scalable Computing. He is member of: IEEE WG for the IEEE P3203 Standard on Cloud Interoperability, IEEE Intercloud Testbed Initiative, IEEE Technical Committees on Scalable Computing (TCSC) and on Big Data(TCBD), Cloud Standards Customer Council, Cloud Computing Experts' Group of the European Commission.

Nicola Mazzocca is a full professor of High-Performance and Reliable Computing at University of Naples Federico II, Italy. He was the head of the Department of Electrical Engineering and Information Technologies. He served as coordinator in several national and international research projects. His research activities include methodologies and tools for design/analysis of distributed systems, secure and reliable systems and dedicated parallel architectures.

Adriano Peron is full professor in Computer Science at the Department of Electrical Engineering and Information Technologies of the Università degli Studi di Napoli Federico II. He got a Ph.D. in Computer Science from University of Pisa. His research activity mainly focus on the development and application of techniques for the specification and verification of inherent properties of concurrent and distributed systems with a particular attention to their real time aspects. A keyword of interest is model checking of finite and infinite state systems with respect to pointbased and interval based temporal logics.

Gaetano Pisani is Business Application Team Leader for Mate Consulting. He takes care of requirement analysis and development of web-oriented applications. He has more than 10 years of experience in software development and currently he is working on different projects for the development of web-based applications.

Luigi Velardi got his Master's degree in Computer engineering in 2002. In 2004 he gained a Post Master's degree on "Computer-based Systems in safety critical industrial application". He is working in Hitachi Rail STS since 2004, first as RAMS engineer, performing validation and verification activities on Italian national STM (SCMT), then, from 2009, he moved to the Product Development department to manage the System Integration and the Testing for Wayside Signalling Systems, such as ERTMS RBC, Railway and Metro Interlocking and CBTC Metro Systems. In Product Development, he worked also on project to increase the Standardization and the Modularization of the developed systems, and on several European funded R&D projects to improve the V&V and testing tools and methodologies used in Hitachi Rail STS.

Valeria Vittorini is Associate Professor at University of Naples Federico II since 2005. She received her M.D. in Mathematics and her Ph.D. in Computer Engineering from the University of Naples Federico II in 1991 and 1995, respectively. She teaches computer programming, formal modeling, workflow and process automation. Her current research interests include dependability and performance evaluation of computer systems, validation and verification of critical systems, critical infrastructures protection with a special focus on railway transportation systems.