 THE RESEARCH LEADING TO THESE RESULTS HAS RECEIVED FUNDING FROM THE EUROPEAN UNION'S SEVENTH FRAMEWORK PROGRAM (FP7/2007-2013) FOR CRYSTAL – CRITICAL SYSTEM ENGINEERING ACCELERATION JOINT UNDERTAKING UNDER GRANT AGREEMENT N° 332830 AND FROM SPECIFIC NATIONAL PROGRAMS AND / OR FUNDING AUTHORITIES.

CRitical S**YST**em Engineering **A**cce**L**eration

Data and Methodologies report

D501.010

DOCUMENT INFORMATION

Project	CRYSTAL
Grant Agreement No.	ARTEMIS-2012-1-332830
Deliverable Title	Data and Methodologies report
Deliverable No.	D501.010
Dissemination Level	CO
Nature	R
Document Version	V1-0
Date	2014-02-07
Contact	Renato De Guglielmo
Organization	Ansaldo STS (ASTS)
Phone	+39 (0)81 243 7608
E-Mail	Renato.DeGugliemo@ansaldo-sts.com

Authors

Name	Company	E-Mail
DE GUGLIELMO Renato	Ansaldo STS	Renato.DeGugliemo@ansaldo-sts.com
VELARDI Luigi	Ansaldo STS	Luigi.Velardi@ansaldo-sts.com
NARDONE Roberto	Ansaldo STS	Roberto.Nardone.Prof608@ansaldo-sts.com
MASSAROLI Gianpaolo	Ansaldo STS	Gianpaolo.Massaroli.Prof644@ansaldo-sts.com
MARRONE Stefano	Seconda Università di Napoli	stefano.marrone@unina2.it
VITTORINI Valeria	Università “Federico II” di Napoli	valeria.vittorini@unina.it
GENTILE Ugo	Università “Federico II” di Napoli	ugo.gentile@unina.it

Reviewers

Name	Company	E-Mail
BARBERIO Gregorio	Mate Consulting	g.barberio@mateconsulting.it
POISSON Pascal	Alstom Transport	Pascal.poisson@transport.alstom.com
KRENN Willibald	AIT Austrian Institute of Technology	Willibald.Krenn@ait.ac.at

Change History

Version	Date	Reason for Change	Pages Affected
0.1	10/01/2014	Create document	All
0.2	21/01/2014	Introduction of internal review comments and updates	All
1-0	07/02/2014	Introduction of external review comments and updates	All

CONTENT

Content of Figures

Content of Tables

1 Introduction

1.1 Role of Deliverable

This document has the following main purposes:

• specifying the main methodological requirements for Radio Block Centre, selected as ASTS use case;

• describing the state-of-art of modelling methodologies;

• showing the modelling methodology to be adopted in ASTS use case.

1.2 Relationship with Other CRYSTAL Documents

This document is strictly connected with the deliverable “CRYSTAL_D_D501.020 – Use Case Requirements Specifications”, where the details of ASTS workflow, based on the methodology chosen to be adopted in ASTS use case (here described), are shown. Indeed, in D501.020 the main methodological requirements presented in Section 2 of this document are translated in technological requirements which are necessary for the ASTS needs.

1.3 Structure of This Document

This document is structured as follows:

- Section 1 (this Section) introduces the contents and the structure of the document, clarifying also the relationships with other documents related to the CRYSTAL project;
- Section 2 provides a short description of RBC (Radio Block Centre), selected as ASTS use case, the context in which it is located and the fundamental requirements of the CRYSTAL methodology;
- Section 3 describes the state-of-art of testing automation, describing the adopted methodology and explaining the reasons which conduct us to this choice;
- Section 4 reports the list of acronyms used in this document;
- Section 5 reports the list of references.

2 ASTS use case: the Radio Block Centre

2.1 Overview of Use Case

The use case chosen by ASTS, in which the new methodology provided by CRYSTAL project is applied, is the RBC (Radio Block Centre) system, the main component of Level 2 of European Rail Traffic Management System / European Train Control System (ERTMS/ETCS).

The ERTMS is an initiative backed by the European Union to enhance cross-border interoperability and the procurement of signalling equipment by creating a single Europe-wide standard for train control and command systems. The main component of ERTMS is the ETCS, a signalling, control and train protection system designed to replace the many incompatible safety systems currently used by European railways, especially on high-speed lines: it allows a train equipped with ERTMS/ETCS to travel without signal system boundaries within the ERTMS/ETCS fitted infrastructure network, regardless of the country the train is travelling in, the legal nature of the infrastructure manager or the supplier providing the ERTMS/ETCS system.

The ERTMS concept has been developed in 4 different functional levels, depending on the system architecture. Level 0, level 1 and level 2 of ERTMS/ETCS are already implemented (while Level 3 is currently under development) and in revenue service in most European countries and beyond, with Ansaldo STS acts as global leader in ERTMS/ETCS components and systems for conventional and high-speed lines.

The definition of the ERTMS level depends on how the route is equipped and the way in which the information is transmitted to the train:

- It's possible to talk of ERTMS/ETCS Level 0 when ERTMS/ETCS-compliant locomotives or rolling stock interact with line-side equipment that is non-ERTMS/ETCS compliant. A driver shall observe the physical signals encountered along the route, knowing the specific meaning of those signals on the railway;
- ERTMS/ETCS Level 1, instead, consists of a cab signalling system that can be superimposed to the existing conventional signalling system leaving the fixed signal system (national signalling and track-release system) in place. The on-board equipment monitors and calculates the maximum speed and the braking curve relying on the data received from the beacons at fixed points;
- ERTMS/ETCS Level 2 is a train protection system based on continuous communication of variable data between the RBC and the trains via a radio system (some additional information is received on board via fixed beacons);
- Finally, ERTMS/ETCS Level 3 implements a full radio based train control and spacing hence fixed block track equipment is no longer required. This technology allows for detecting the current position of each train always in time, hence it is possible to send continuously line-clear authorization to each train.

As described in Figure 2-1, in ERTMS/ETCS Level 2 the RBC continuously monitors the train movements because it automatically finds out from trains their exact position, it receives train detection and it routes status information from the interlocking and automatic block system as applicable, making all information available to each train continuously via GSM-R in form of movement authorities. The RBC interacts with the on-board unit by managing a communication session using the EURORADIO protocol and the GSM-R

network. A single RBC can contemporary communicate with a different trains depending on the maximum number allowed by physical characteristics of the GSM-R network.

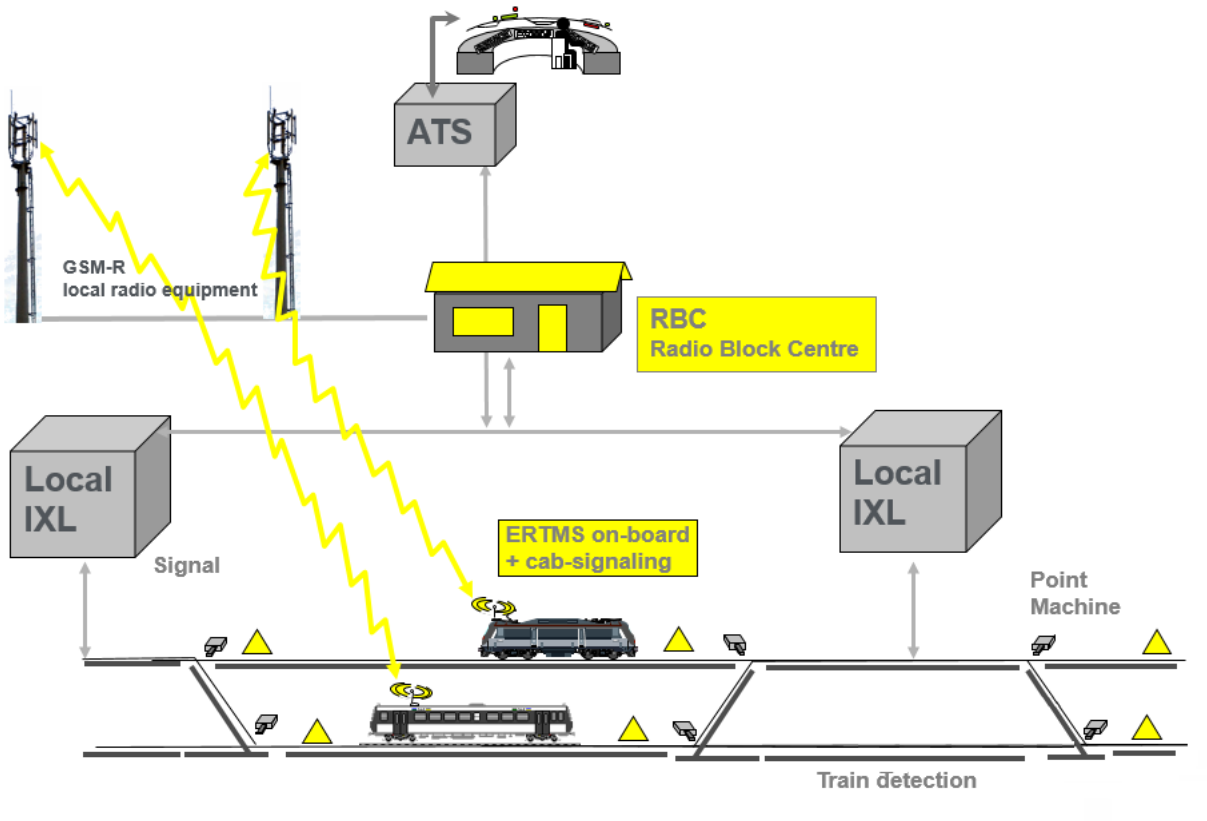

Figure 2-1: ERTMS/ETCS – Level 2

Different RBCs are implemented for different railway projects; for this reason implementation and validation activities shall be performed again.

For what concerning the implementation, RBC is commonly structured in a portion common to different implementations (the **RBC core**) and a portion which is coherently updated in each project (the **RBC specific**). Requirements and test cases related to the RBC core are commonly similar between different projects.

The V&V activities should benefit from this situation in order to reduce time and costs of the RBC core validation. Furthermore the detection of a bug on the RCB core of a particular implementation should generate an alert which requires a further verification on the other implementations.

2.2 Context of Use Case

As explained in Section 2.1 ERMS/ETCS allows the interoperability of European railway signalling systems specifying the traffic management interfaces which shall be implemented both at technological and at procedural levels. This system provides the specification of the traffic management and the train control

system that enables the transit of high speed trains through national borders. In this way all the real implementations of the ERTMS/ETCS shall comply with the Union of Signalling Industry (UNISIG) System Requirement Specification (UNISIG SUBSET-026).

The ERTMS/ETCS provides the safe movement of trains and the optimal regulation of traffic of high speed trains; for this reason the entire system (and specifically the RBC) shall be classified as safety critical although complex system: it shall guarantee the safety of the train movement, preventing collisions in any case, also in situations of breakdowns and human errors. In this context several international standards are applicable since the reliability and safety evaluation and management are mandatory for this kind of systems. The basic one is the IEC EN 61508 [Nordland, 2003], an international standard for the functional safety of programmable electronic safety-related systems (PES) produced by International Electrotechnical Commission (IEC). This standard defines the SIL (an indicator -whose acronym stands for Safety Integrity Levels- which represents the integrity level of safety functions) of a system as measurement of safety required/obtained by itself; this definition is applicable to all kind of industries for both hardware and software components. It is important to remember that the SIL is related to a single safety function and not the entire system or individual components: within a given system a lot of safety features will exist, each of them related to a particular hazard to which an appropriate SIL will be associated. The whole set of components of each security system must be such as to respect the SIL class to respect.

This standard does not define the SIL to achieve for the specific application domain but it lists all the operations that must be carried out, i.e. a risk analysis of the specific system and an assessment of acceptable risk as a combination of the probability and the hazard level. This standard covers the complete safety lifecycle decomposed into 16 phases addressing analysis, realisation and operation of a safety critical system. The main concepts defined by this standard lie in the fact that safety must be considered from the beginning of the lifecycle, non-tolerable risks must be reduced and zero risk can never be reached.

Safety related concepts are interpreted in specific application domains and appear under different other standards; in the railway field three of them are nowadays applicable: EN 50126, EN 50128 and EN 50129 [Nordland, 2003]. These standards have been produced by CENELEC, the European Committee for Electrotechnical Standardization, which is responsible for standardization in the electrotechnical engineering field. CENELEC produces standards and reference documents. CENELEC applies international standards, wherever possible, through its collaboration with IEC.

The three aforementioned CENELEC standards represent the backbone of the RAMS demonstration process of a railway system: possible failures and hazards are identified during the overall lifecycle, they are properly corrected or mitigated considering their occurrence rate and the effort to spend; finally the risk is evaluated. In detail EN 50126 describes the processes and methods that are used to specify the most essential and important aspects for operability and safety in the rail domain; the EN50128 and the EN 50129 give a set of requirements which have to be satisfied during the safety-critical software (the former) / hardware (the latter) development, deployment and maintenance phases. The lifecycle suggested for these systems is a common ‘V’ lifecycle where the design is implemented during the descendent activities, which correspond to the verification and validation activities performed during the ascending branch.

The same standards also define the list and the minimal contents of the documents and deliverables that have to be produced during the lifecycle of a new railway system: in particular it is clear that the entire set of requirements, test cases and reports have to be traced in appropriate documents. It also imposes the clarification of the traceability between requirements and test reports in order to demonstrate how each requirement has been satisfied by outcomes of test cases.

The ERTMS/ETCS, and RBC specifically, implement also functions classified as SIL 4 (the most dependable one). This is a key factor in system verification and validation: in fact some techniques listed in standards are highly recommended for these systems. In particular, among others, the adoption of a structured methodology is required for SIL 4 systems; formal methods and a modelling approach are needed during the requirements specification as well as during design and implementation. Formal proof, static and dynamic analysis and testing are highly recommended during component and integration testing; modelling is required at system level to have a clear and complete understand of the system behaviour. Furthermore the evaluation of some coverage metrics is necessary: specific test cases which stress a predefined portion of the system shall be defined in order to individuate which portions are active and which ones are unreachable.

2.3 ASTS Requirements Management Approach

Actually ASTS implements a well-defined documental process to manage requirements related to RBC during each railway project. All the requirements of a specific signalling system are traced in a document, which is properly updated during the entire system lifecycle. This document, known as “Technical Specifications”, lists all the requirements of the system including specific requirements indicated by the client. These requirements are reported in natural language and then, after their complete understanding in collaboration with the client, they are properly translated into semi-formal specification (the “System Requirements”). After these steps all the requirements are apportioned and assigned to all the sub-systems involved, and specifically to the RBC.

The list of RBC requirements is hence reported in a specific technical document, commonly known as “RBC System Requirements”. This document reports all the functions implemented by the RBC specifying also their interfaces. The document flow is depicted in Figure 2-2.

The “RBC System Requirements” document contains all the information necessary to implement a new RBC system and to produce its Test Specification. In particular this document contains the complete description of each function that shall be implemented by the RBC and its specific requirements. Most of the requirements are reported in a “state / condition / action” format: the “state” specifies the state in which the RBC is in a specific instant of time, the “condition” expresses the set of conditions to which eventually the RBC shall react, the “action” defines which are the action that shall be performed. Since this document contains the entire description of the RBC behaviour, the implementation can be based on this information. The document also represents the basis of the validation activities. In fact, in the current engineering lifecycle, V&V engineers rely on this information to define the RBC Test Specification.

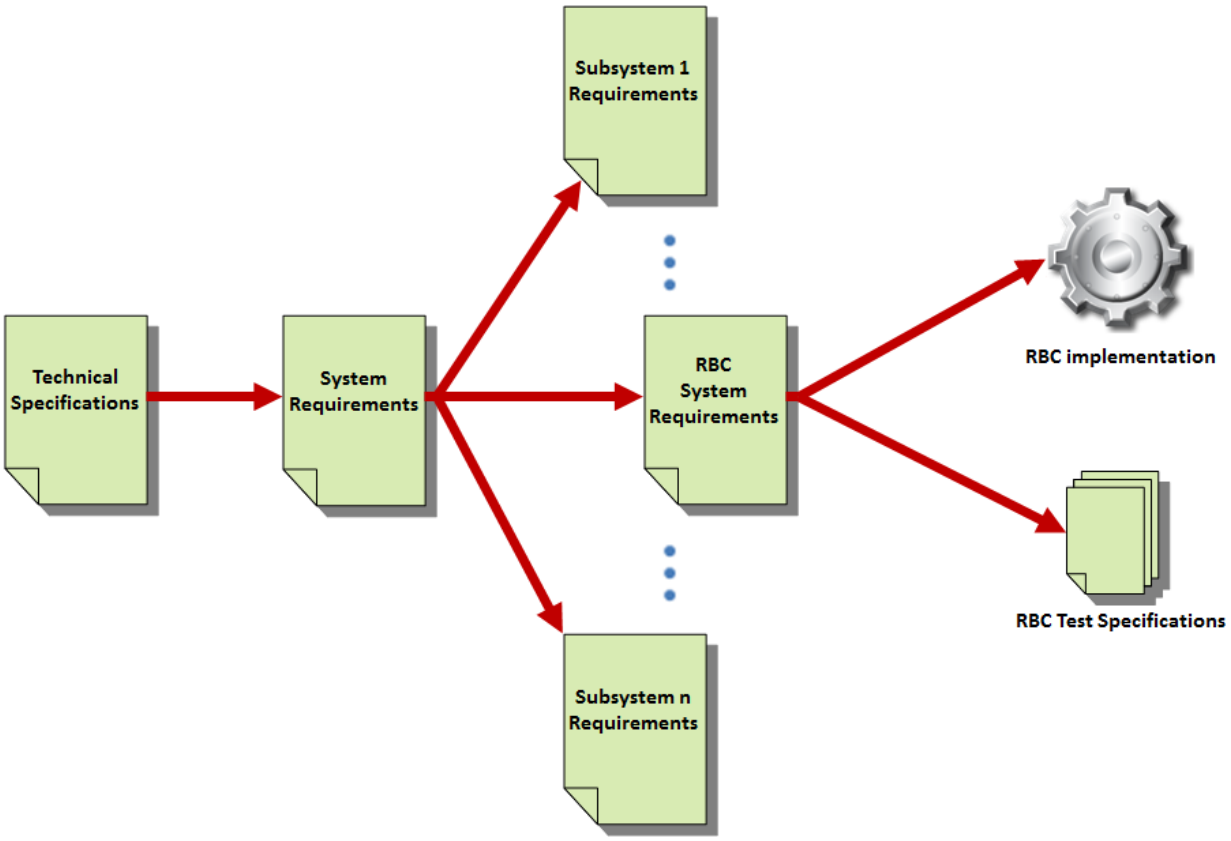

Figure 2-2: Document flow of the ASTS requirements management approach

2.4 ASTS Validation Approach

The validation approach actually implemented in ASTS is structured as described in this section. As described previously, the Validation approach is centred on the information contained in the “RBC System Requirements” document. Specifically, this document acts as input for the implementation and the test specification activities. Appropriate Test Cases are obtained from the set of Test Specifications, and are then translated into executable tests (the Test Scripts). Test Scripts are executed on the implemented system, both in simulated and in real environments. Each test execution is logged on complete and complex Test Log files, which are then analysed in order to verify the correctness referring to the expected outcomes. The test outcomes, in a more essential and comprehensible format, are also reported in the “RBC Test Report” document, where also the mapping to the initial requirements is explicated.

Figure 2-3 depicts the validation approach implemented by ASTS as described. The two backward arrows indicate the feedbacks of the test execution on the implementations and/or on the Test Specification (if some tests discover bugs) of the RBC.

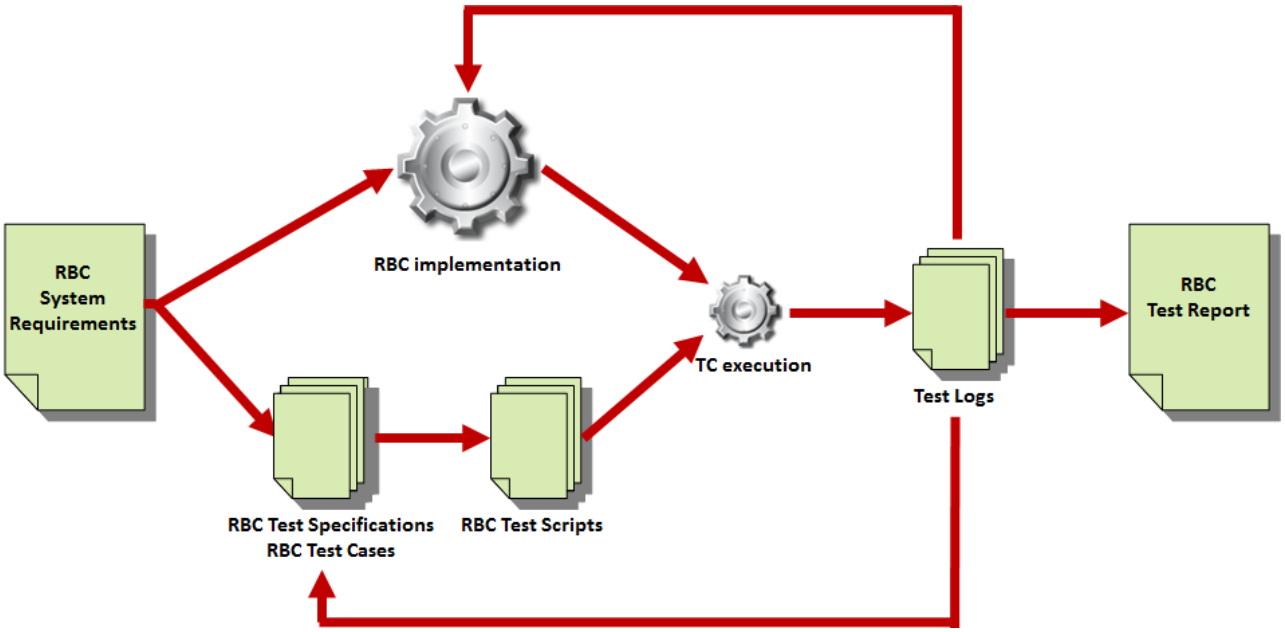

Figure 2-3: ASTS validation approach

2.5 Requirements of the Crystal methodology

This paragraph reports the main methodological requirements that have been defined after the context analysis. These requirements are here listed, their mapping with the Crystal methodology are indicated in the following Section 3.

ASTS verification and validation activities are conducted by engineers that need to comply with the applicable norms and standards, both in terms of system engineering lifecycle and of required formalisms/techniques. In particular it is necessary that the Crystal methodology shall be able to support the test case generation at system level and, at the same time, it shall be extensible in order to support future extensions.

The input of the approach shall be the system model that is defined using a proper state-based formalism; this language shall support the definition of graphical interfaces and it shall support future specializations and extensions.

The entire methodology shall be verifiable since the target systems are safety critical. It shall have a high level of automation, where possible, in order to reduce the influence of human errors during the generation of test case. In this way the methodology shall be able to generate eventually a test case if it is effectively practicable, otherwise it shall be certain that, if a test can't be generated, it is infeasible. Human influences shall be reduced to the modelling and updating of the source model; in fact the entire methodology shall be used by experts of the rail domain who may not know which is the adopted approach of test generation. Furthermore, since the signalling system is classified as complex systems, the entire methodology shall be usable by different users who shall conduct the validation of the same system at the same time. The multi-user requirement in conjunction with the necessity to trace requirements, test cases and test reports (as

imposed by applicable standards) requires the implementation of a Concurrent Versions System, which includes the trace of the users which update documents and/or artefacts.

At last regression testing shall be supported: test cases that impact on a portion of the source model shouldn't be re-generated after the updating of the source model in other portions.

In the following the list of requirements to which the Crystal methodology shall comply is summarized; some of them (i.e. REQ.501.010_02, REQ.501.010_07, and REQ.501.010_09) are qualitative, their effective implementation in the Crystal project will be verified by ASTS after the implementation of the Crystal workflow:

- **(REQ.501.010_01)** the methodology shall be compliant with the lifecycle introduced by applicable norms, in particular it shall be applied at the system testing level (i.e. generated tests shall be used to perform the final validation against system requirements);
- **(REQ.501.010_02)** the methodology shall have an high level of automation, where possible;
- **(REQ.501.010_03)** the methodology shall trace the coverage between test cases, test reports and requirements;
- **(REQ.501_010_04)** the methodology shall support the consistency when different users work on the same system;
- **(REQ.501_010_05)** the methodology shall support versioning of the source models, of the test cases and of the test reports;
- **(REQ.501.010_06)** all the steps of the proposed methodology shall be verifiable since it shall be adopted in the lifecycle of critical systems;
- **(REQ.501.010_07)** the methodology shall be, whenever applicable, extensible in order to allow future extensions of the modelling language and of the final scope**;**
- **(REQ.501.010_08)** the modelling approach shall rely on state-based formalism as source modelling language;
- **(REQ.501.010_09)** the test case generation method shall be usable by experts of the rail domain which may not know how tests are generated;
- **(REQ.501.010_10)** the test case generation method shall not generate a test case from a test specification if the test specification is effectively infeasible, otherwise a test shall be eventually generated;
- **(REQ.501.010_11)** the test case generation method shall not re-generate test cases when no updates are performed on model portions.

3 Modelling Methodologies

Testing automation is an objective to pursue in order to achieve high process efficiency and product quality. This objective can be accomplished by creating a software suite, distinct from the System Under Test (SUT).

It is used to control the execution of tests and to analyze the results comparing them to what expected. This process is composed by four different stages as depicted in Figure 3-1:

- Test Generation a high-level description of the test suite (the abstract test suite) is generated from the SUT specification. This suite generally is not executable;
- Test Implementation: the abstract test suite is rewritten to obtain a test suite written in a machine-readable language;
- Test Execution: the test suite is executed on the SUT
- Test Result Analysis: outcomes from the tests running are analyzed and compared with predicted outcomes generated by oracles.

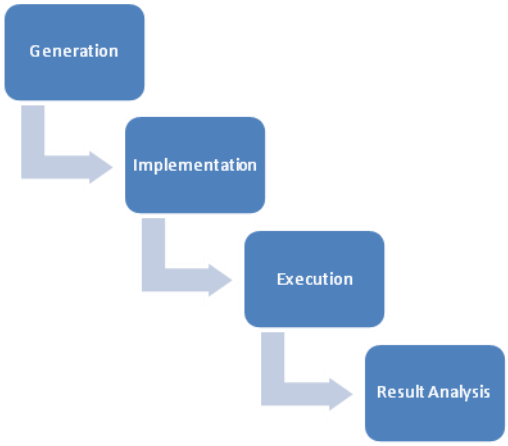

Figure 3-1: General testing process schema

Testing automation activities can be applied at three different levels:

- Unit testing, in which each single unit (as functions or modules) of SUT is tested separately;
- Component testing, where some units are aggregated and their interactions are tested as well as their interfaces;
- System testing, where all the subparts of the SUT are integrated in their environments concurring to provide system functionalities (that are the objective of this testing phase).

This section addresses test case generation when applied at system level by using model-based approaches.

Automated Model Based Testing (MBT) is considered a leading-edge technology in industrial settings since it has proven to increase both quality and efficiency of V&V processes. According to MBT the focus of the testing activities is shifted from test data elaboration and procedure programming to modeling.

Models can be used to represent the behavior of a SUT, as well as testing strategies and test environments. One of the reasons why MBT has received growing attention in recent years was the success of model-driven development and the availability of efficient tools support.

A wide literature on MBT is available: a brief review of meaningful surveys and books on MBT is provided in D_603.011. Here, the focus is on model based test case generation for system testing of critical embedded system and specifically on the modeling approach underlying the generation process. In the following the assumption is made, that the modeling notation adopted to describe the behavior of the SUT is state-based. This assumption is motivated in this section.

3.1 Automated Test Case Generation

In the scientific literature, there are several approaches to support automatic generation of test cases according to the adopted testing paradigm.

Black Box Testing

In the black box approach test are generated starting from system specifications: the internal structure of the system may be unknown. Most of the black-box testing automation techniques support conventional techniques like:

- Equivalence partitioning: a technique whose objective is to reduce the total number of test cases necessary by partitioning the input conditions into a finite number of equivalence classes [Myers, 2011]. There are two main types of classes: the set of valid inputs to the program is a valid equivalence class, and all other inputs are grouped in invalid equivalence class.
- Boundary value analysis: test cases are based on boundary conditions which are the values at, immediately above or below the boundary or “edges” of each equivalence classes.

Other approaches consider lightweight models of the system according to its properties/features. Feature oriented testing generates test cases starting from the features of the system as they are represented into a feature model [Kitamura, 2012]. The relationship between inputs and outputs is addressed in another paper that exploits such rules to reduce the complexity of the brute force approach [Schroeder, 2002]. Learning techniques exploits the implicit knowledge that comes from the real interaction between the user and software in interactive systems to automatically derive test cases [Mariani, 2011].

White Box Testing

Normally, this kind of automation test refers to Control Flow Graph (CFG) which is a directed graph $G=\{V, E, s,e\}$ in which V is the set of nodes each one representing a single statement, E is the set of edges connecting the statements; s and e are the start and end points of the program under test. The main goal in white box testing is to find an admissible path on CFG to reach a sufficient level of code coverage. Many techniques support testing automation in the white box approaches:

- Random Testing: this approach aims at producing random test data to give as input to SUT. This method normally is used during Integration or Unit testing because it aims to large structural coverage. The benefits that random testing techniques bring to the testing process are mainly its inexpensiveness, its capability to be applied in the evaluation of software reliability level, its capability to be exploited to perform stress testing. The disadvantages are that there is no assurance for full code coverage and that it needs an automatic log verification phase due to the huge size of produced logs.
- Path-Oriented Methods: this approach generates test data using a path, or series of paths, on the CFG. The analysis can be static or dynamic. In the first case test case generation is made without program execution. A major technique in this approach is the Symbolic execution. It involves executing a program using symbolic values of variables instead of actual values. With this method, it is possible to obtain inequalities describing the necessary conditions to cross the path. In general

formulation, path-oriented testing is NP-Hard, but with linear constraint it is possible to use linear programming techniques. The way to overcome statically problems is to use dynamic approach in which the analysis is made during run-time of program under test. Program execution flow is monitored during run-time: in case of deviations from expected flow, some techniques based on (meta-)heuristics like simulated annealing [McMinn, 2004] or backtracking, are used to identify the problem. The major disadvantages of this technique derive from program execution due to the high number of iterations needed to generate the test suite.

- Goal-Oriented: instead of path oriented approach, in goal-oriented one, test generator can generate test suite also for non-complete path, in which there are missing path. This operation is easier than respective in path oriented. One of the major techniques based on goal-oriented approaches are the chaining and the assertion approaches. In the first, a sequence of events is used to find a testing path; the sequence is based on the software nodes to visit in order to reach the objective of the test. On the other hand, an assertion can represent pre-conditions or post-conditions that must be satisfied during the program execution. The objective of testing is to find a path that satisfies the assertions. The most peculiar feature of this last technique is that test oracles are embedded in the code.
- Genetic Algorithm: genetic algorithms are search heuristics based on the natural process of the evolution [Srinivas, 1994]. Genetic algorithms start from a random population of solutions (called chromosomes). Through a recombination process and gene mutation operators, a genetic algorithm evolves the population to the (sub-)optimal solution. The first step of this process is the selection of the solutions in the current population that will be used as parents for the solutions of the next generation. The parameter on which the choice is done is called fitness. A solution closer to optimality has a higher fitness value than others. The most important operators are: selection, mutation and recombination. In testing context, single tests are chromosome and the fitness is the code coverage. Many works were conducted on genetic algorithms [Pargas, 1999] [Tonella, 2004].

Regression testing

Automated approaches are present also during the regression testing phases. Regression testing deals with rerunning previously defined test cases during the lifetime of the system. Typically, this is necessary during maintenance and late development lifetime phases; it mainly aims at checking that changes have not affected previously tested software functions. Regression test can be performed in several ways: one of them is to rerun all tests in the test suite, but this is the most expensive way. The research objective in this context is to minimize test cases to re-execute. A tool that has this objective can be found in [Fischer, 1982]: the tool is based on objective functions subject to some constraints. Moreover, regression testing can exploit model based techniques in order to achieve high order of efficiency [Felderer, 2012].

3.2 Model-based Testing

According to [Zander, 2011], there are several approaches for automatic test case generation in literature. In the rest of the report the main focus will on such techniques more suitable to critical embedded systems. Model based testing is a first-class citizen in testing safety critical systems from a system perspective: Figure 3-2 depicts a reference schema of the approaches for embedded systems.

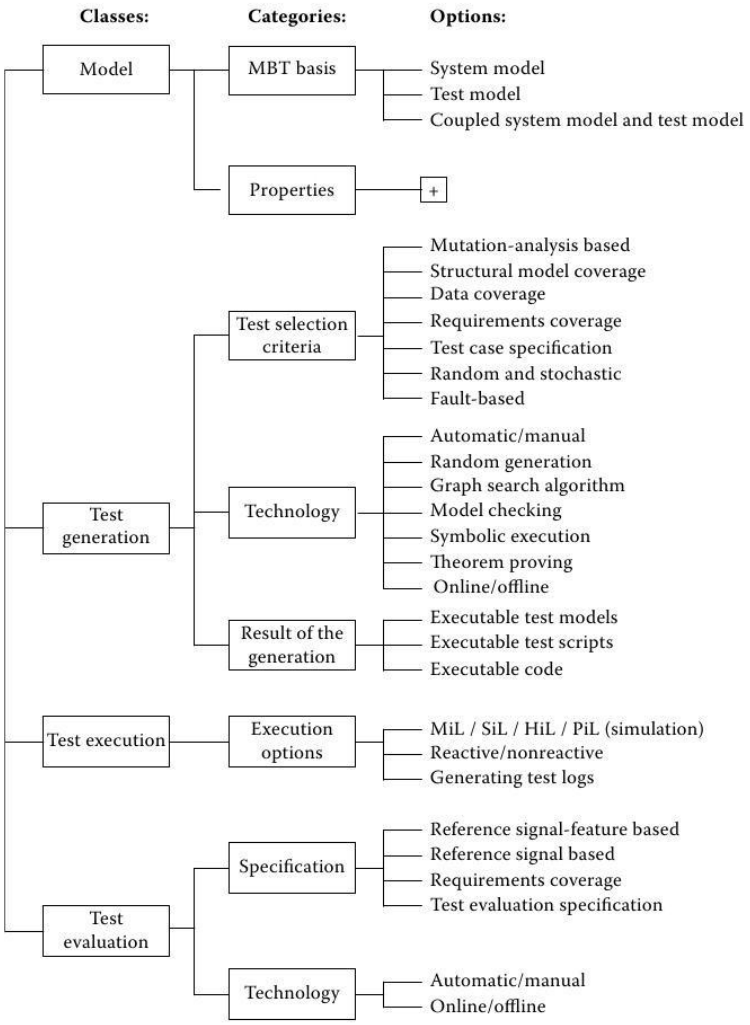

Figure 3-2: Model Based Testing overview (from [Zander, 2011])

The expression “model-based” applied to testing assumes several means [Utting, 2012], like:

- Generation of test input data from information about the domains of the input values (domain model). This reduces hand-made work but does not provide any information to know whether a test has passed or failed;
- Generation of test cases from an environment model. A model represents the environment of SUT. This approach does not provide information on testing outcome, too;
- Generation of tests cases from a behavioural model. A model describes the expected behaviour of the SUT. This approach needs oracle information to compare outcomes;
- Generation of test scripts from abstract representation of tests. Models describe test cases. From them proper transformations generate machine-readable tests.

The success of this approach is based on: notation used for model (clear and scalable), generation test algorithms and tools used for test execution and outcomes evaluations. Some example of model-based testing can be found: some researcher of BellCore produce an easy notation to model input domain [Dalal, 1999], this notation suffer of testing oracle absence and of dimension of the system; in other approaches model-based is used for testing automation by means of a combination of model-based testing with formal methods concept to create a tool named Torx [Tretmans, 2003].

Model based test case generation has been studied in some European research projects (as MOGENTES), too [MOGENTES, 2008].

According to a review of the scientific literature and to the needs of real world industries, the methodology presented in Subsection 3.3 is mainly based on the choice of specific techniques inside the discussion about methodological concerns. These themes include: the modelling paradigm, the process schema and the algorithm for test cases generation.

In the rest of this Subsection a discussion and a motivation for each of these three themes is reported.

3.2.1 A motivation for state-based modelling

Suitable formalisms for modeling reactive systems span from UML via Process Algebras to domain-specific languages. State based notations are widely used in modeling safety critical systems. Specifically, in the rail domain, this technique is highly recommended by applicable standards and norms in developing SIL 1 to SIL 4 systems. In fact, the behavior of safety-critical systems shall be completely specified against every input in every state: a state based representation describes how the system moves from one state to another depending on the input and the current state.

Hence, the behavior of the SUT and the requirements of a critical system are usually specified in form of state-transition machines and/or tables. Automatic test generation processes require that the input models are expressed by means of a machine-processable notation.

At this aim, several state-based modeling languages may be used. A not-exhaustive list includes Finite State Machines (FSMs), Extended FSMs, Abstract State Machines (ASMs), Statecharts, UML State Machines, Timed Automata, Input-Output Automata, Petri Nets, State Flow Diagrams and Markov Chains. The operational semantics of state-based models enable a variety of activities to be performed, such as model checking, reachability analysis and simulation.

We restrict the following discussion to not-timed notations, as our application domain does not require introducing a notion of time explicitly. In particular we focus on FSMs and their extensions (EFSMs, ASMs, State Charts and UML state machines with no notion of time), as they allow for defining the control structure of the system not only in terms of states and possible inputs, but also by specifying the associated actions.

The usage of FSMs is explicitly mentioned by the CENELEC standards [CENELEC, 2004]. FSMs reflect the system complexity; they are modular and can be structured in accordance to the structure of the system.

FSMs can also be statically checked for completeness (an action and new state must be specified for every input in every state), for consistency (only one state change is defined for each state/input pair) and reachability (whether or not it is possible to get from one state to another by any sequence of inputs). These are important properties for critical systems and they can be checked on the model before its implementation and/or verification and validation activities.

FSMs have been extended in several ways to improve the description of complex system behaviors, for example by enabling the construction of hierarchical models, introducing the representation of composition (parallelism), inter-level transitions, history states, etc. A very useful feature is the nesting of internal states and transitions, which enables the possibility to reveal or conceal the internal states at need.

3.2.2 A motivation for model driven techniques

Model Driven Engineering (MDE) is a promising approach that is able to cope with the increasing complexity of platforms. More in general, model-driven techniques bring together two important aspects:

- Domain-specific modelling languages (DSMLs), which formalize the application structure, behaviour, and requirements within a specific domain;
- Transformation engines and generators that analyse certain aspects of models and synthesize different types of artefacts, such as source code, simulation inputs, XML deployment descriptors, or representation of models.

The MDE initiative proposes a process definition wider and not limited to the development as for other approaches (Model Driven Architecture [OMG, 2003], Model Driven Software Development [Mellor, 2003]). A brief description of the key aspects of MDE and MDD is provided in D603.011.

MDE methods and techniques are applicable to general purpose software systems as well as to critical systems. In particular, in this last case, the effort must be oriented to support the qualitative and quantitative analyses since the verification of system properties plays a crucial role for the system success. Great benefits may derive from the adoption of MDE in the development of critical systems: the paradigm of systems design “construct-by-correction”, typical of processes based on testing and verification of the late-time properties of a system, can be replaced by paradigms “correct-by-construction” where, by verifying the correctness of both initial system model and model transformation, one can assure the correctness of the final model. The effort of verification can thus be concentrated in the initial stages of the system lifecycle.

Within the quantitative evaluation, outlined above, and within the scope of the techniques and methods of the MDE, a first scenario of application of these techniques to critical systems is clearly defined: there is the possibility to increase the spread of formal methods in industrial development processes in real systems. In fact, the ability to define high-level languages closer to the user (for abstraction and ease of use) as well as the ability to generate automatically models (formal models for quantitative analysis) enables the usage of formal methods in a “transparent” way.

In the context of this project, model-driven techniques play an important role since they are able to fill the gap between high level – user friendly modelling approaches and lower level test case generation algorithms.

3.2.3 A motivation for model checking based test case generation

Among test case generation techniques, a very interesting approach is the one based on formal methods and model checking. Model checking is a technique used to analyze a finite-state representation of system for property violations. The main component, the model checker, analyses all reachable states evaluating if the property is verified. Otherwise, if a violation of the property is observed in a state, the model checker returns a “counterexample”, i.e. a sequence of states which conduct in the state where the property is violated.

Model checking in automation testing context is used for two reasons [Gargantini, 1999]: first, the model checker can be used as a oracle for test outcomes evaluation and second for the model checker ability of generate counterexamples which is used to construct test sequences. This is the major challenge for testing based on model checking because it is necessary to force the model checker to construct test sequences. [Gargantini, 1999] proposes a method to generate tests case from properties: if we have to verify property P, we have to express it by a temporal logic (e.g. CTL). As our goal is to generate test case we, ask to model checker to verify the negation of P. To demonstrate violation, the model checker produces a counterexample which is the trace of steps that start in valid state and end in the state in which there is the violation: this trace contains steps of single test of test suite.

In the context of this project, the usage of model checking to generate test cases is very interesting. Model checking exploits both formal languages and analysis techniques that are recommended for critical systems by international standards. Moreover model checking uses a state-based modeling language that has been

already discussed. Notwithstanding, model checking gives the absolute confidence that a test is feasible or not feasible, helping the system developer or test engineer to verify all the system requirements.

3.3 Test Generation Methodology

In this Section, a summary description of the proposed methodology is reported.

3.3.1 General Concept

Figure 3-3 depicts the schema of the macro-activities that will be conducted during the development of the methodological bricks related to the UC 5.1. This schema is already defined in the deliverable D_603.011 related to WP 6.3. In D_603.011 where a generic “abstract” version of the process is defined; here a concrete version is provided, instantiated by starting from the techniques previously discussed and motivated.

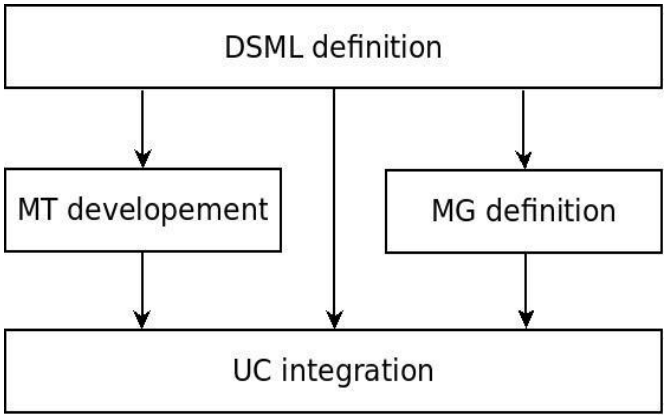

Figure 3-3: Developing the TG methodology: macro-activities to be performed

A first macro-activity (*DSML definition*) requires that a modelling language is defined to represent the system behaviour and the test properties. The second macro-activity (*MT development*) requires the definition and implementation of proper Model Transformations (MT) in order to generate the artefacts needed in the test generation step from the models represented by means of the DMSL: this step will tailor the target languages for the model transformation into formal languages able to be easily analysed by means of model checkers. The third macro-activity (*MG Definition*) concerns the definition of *Modelling Guidelines* (both for system and test properties) since, in order to define industrial appealing process, some guidelines (in particular by using system and testing specification patterns) will be developed. Finally, the fourth macro-activity (*UC Integration*) addresses the integration of the resulting model driven process and tools into existing and assessed V&V processes as they are adopted by industries.

3.3.2 Process definition

According to the general schema of the Figure 3-3, a more detailed process of the activities to follow is reported in Figure 3-4.

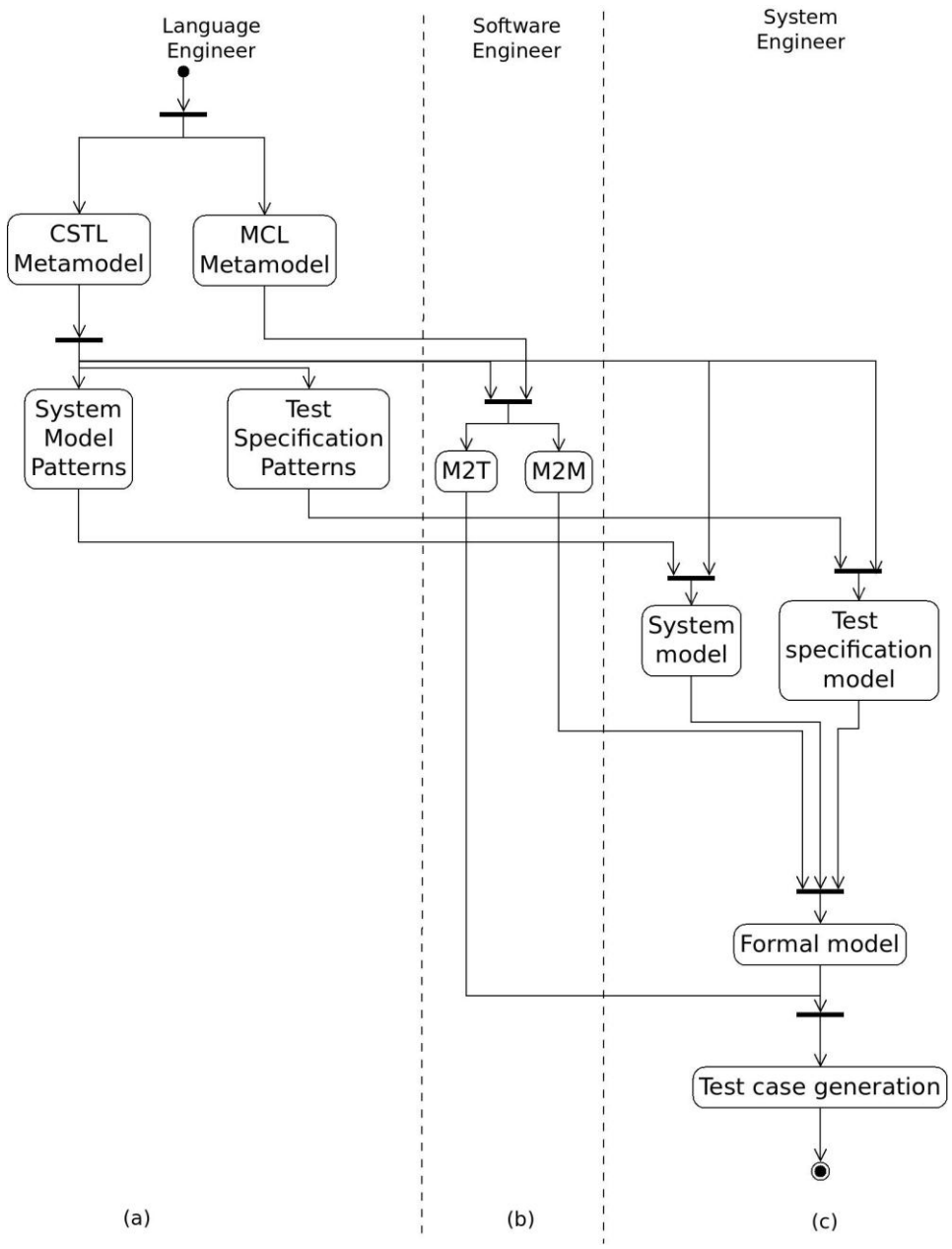

Figure 3-4: Detailed process schema

At the middle of phase (a) the modelling languages involved in the process are defined. Two different classes of languages are needed: DSMLs, used to model testing concepts for critical system domain, and languages involved to describe the formalism used to generate test cases (i.e. a model checking language as Promela, etc.): namely, the *CSTL Metamodel*[1] and the *MCL Metamodel*[2].

The definition of both metamodels is a part of WP6.12 and it will be described in the related deliverables. It is important to underline that these metamodels represent languages strongly formalized with respect to semantics aspects.

[1] CSTL stands for Critical System Testing Language
[2] MCL is a language that supports the application of model checking algorithm

Phase (a) also deals with the definition of proper modelling guidelines that are in charge of helping the modeller by means of patterns and best practices. Two different activities are to be developed since patterns are needed for both system modelling and test specification modelling (*System Model Patterns* and *Test Specification Patterns*).

Phase (b) copes with the definition of model transformations. Two kinds of model transformations are needed: Model-to-Model (*M2M*) and Model-to-Text (*M2T*) transformations. *M2M* transformations are in charge of transforming high level into solvable formal models. *M2T* transformations are in charge of translating formal models into a textual representation format, according to the tool used to solve the model. As for the *M2M* transformations, the source and target languages are respectively the *CSTL Metamodel* and *MCL Metamodel*. The same holds for *M2T* transformations: in this case the source language is the *MCL Metamodel*.

Finally, phase (c) consists of the activities that must be performed within the V&V process of a system. A high level model of the system is created during the *System Model* activity using both the language defined by the *CSTL Metamodel* and the patterns defined in *System Model Patterns*. At the same way a model of the test specifications is created during the *Test Specification Model* activity using the patterns defined in *Test Specification Patterns* instead of *System Model Pattern*. The *M2M* transformations are then applied to generate formal models which perform either model-based analysis or are used as an intermediate step towards test cases automatic generation. During the same activity, test cases are obtained by applying model checking techniques. Finally, proper *M2T* transformations are used during the *Test case generation* activity in order to obtain textual representations of the generated test cases.

3.3.3 Motivations for the proposed methodology

Table 3-1 reports how some requirements given by the UC owner are mapped onto the methodological choices described in this Section.

Requirement	State-based formalism	Model Driven Techniques	Model checking based test case generation
REQ.501.010_01	State based formalisms are indicated by CENELEC as highly recommended during the V&V phases and are widely used to produce a model of the system under V&V which expresses the behaviour of the entire system and the surrounding environment. All the information needed by the source models are available at this stage.		
REQ.501.010_02		Model Driven techniques boost the productivity of organization by improving the level of automation.	

REQ.501.010_03		Model Driven techniques enable to transform a model into another and/or into text, hence it is possible to maintain information about the source model version during the transformation chains.	
REQ.501.010_04	State based formalism support the creation of a model as composition of different sub-models.		
REQ.501.010_05		Model Driven techniques can cooperate with concurrent versioning systems which manage versioning.	
REQ.501.010_06		The definition of modelling languages strongly formalized with respect to semantic aspects and the use of proven-in-use languages and engines for model transformations open for a further phase of verification and assessment of each step of the proposed methodology.	
REQ.501.010_07		Model Driven based approaches are extensible. Both DSML and model transformation may be specialized and/or developed by reusing the existing ones.	
REQ.501.010_08	By assumption.		

REQ.501.010_09		Model Driven techniques rise up the level of abstraction of the engineering process and enable the usage of graphical modelling languages. Hence they improve the usability level and allow for the development of user-friendly graphical interfaces.	
REQ.501.010_10			By exhaustive search, model checking guarantees that, if the test is not feasible, no test case (counterexample) is found.
REQ.501.010_11			By writing proper test specifications, test cases can be generated to test specific parts of the model (those affected by some modifications).

Table 3-1: UC requirements-methodology mapping

4 Terms, Abbreviations and Definitions

ASM	Abstract State Machine
CFG	Control Flow Graph
CO	Confidential, only for members of the consortium (including the JU).
CRYSTAL	**Cr**itical S**YST**em Engineering **A**cce**L**eration
CSTL	Critical System Testing Language
DSML	Domain Specific Modelling Language
EFSM	Extended Finite State Machine
ERTMS	European Rail Traffic Management System
ETCS	European Train Control System
FSM	Finite State Machine
GSM-R	Global System for Mobile Communications – Railway
IEC	International Electrotechnical Commission
M2M	Model-to-model
M2T	Model-to-text
MBT	Model Based Testing
R	Report
RBC	Radio Block Centre
SIL	Safety Integrity Level
SUT	System Under Test
TC	Test Case
UML	Unified Modelling Language
UNISIG	Union of Signalling Industry
V&V	Verification and Validation

Table 4-1: Terms, Abbreviations and Definitions

5 References

[CENELEC, 2004]	CENELEC, *EN50128 - Railway applicationsCommunications, signalling and processing systems - Software forrailway control and protection systems*; 2004
[Dalal, 1999]	S.R. Dalal, A. Jain, N. Karunanithi, J.M. Leaton, C.M. Lott, G.C. Patton; *Model-Based Testing in Practice*; Proceeding of ICSE'99 (ACM Press) (Proceedings of ICSE'99 (ACM Press)), 1999.
[Felderer, 2012]	P. Zech, M. Felderer, P. Kalb, R. Breu , *A Generic Platform for Model-Based Regression Testing;* Leveraging Applications of Formal Methods, Verification and Validation, Technologies for Mastering Change; Lecture Notes in Computer Science Volume 7609, pp 112-126; 2012.
[Fischer, 1982]	K.F., Fischer; *A test case selection method for the validation of software maintenance modification*; Proceedings COMPSAC82, 529-537; 1982.
[Gargantini, 1999]	A. Gargantini, C. Heitmeyer; *Using model checking to generate tests from requirements specifications*; ESEC/FSE-7 Proceedings of the 7th European software engineering conference held jointly with the 7th ACM SIGSOFT international symposium on Foundations of software engineering, 146-162; 1999.
[Kitamura, 2012]	T. Kitamura, N. Thi Bich Do, H. Ohsaki, L. Fang, S. Yatabe; *Test-Case Design by Feature Trees*; in Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change; Lecture Notes in Computer Science Volume 7609, pp 458-473; 2012.
[Mariani, 2011]	L. Mariani, M. Pezzè, O. Riganelli, and M. Santoro: *AutoBlackTest: a tool for automatic black-box testing*. In Proceedings of the 33rd International Conference on Software Engineering (ICSE '11). ACM, New York, NY, USA, 1013-1015; 2011 (DOI: 10.1145/1985793.1985979).
[McMinn, 2004]	P. McMinn; *Search-based software test data generation: a survey*; Software testing, verification and reliability (STVR), 105-156; 2004.
[Mellor, 2003]	S.J. Mellor, A.N. Clark, T. Futagami; *Model-driven development - Guest editor's introduction*, Software, IEEE , vol.20, no.5, pp.14,18, Sept.-Oct; 2003 (DOI: 10.1109/MS.2003.1231145)
[MOGENTES, 2008]	MOGENTES research team, *MOGENTES: Model-Based Generation of Test-Cases for Embedded Systems - State of the Art Survey - Part a: Model-based Test Case Generation Techniques* Vers. 1-19a 1.1r; 2008.
[Myers, 2011]	G.J. Myers, C. Sandler, T. Badgett ; *The Art of Software Testing; 3rd Edition*, Wiley , ISBN: 978-1-118-03196-4; December 2011.
[Nordland, 2003]	O. Nordland: *A critical look at the CENELEC Railway Application Standards*. Presented at the TÜVIT seminar Application of the international standard IEC 61508, held in January 2003 in Augsburg, Germany.
[OMG, 2003]	Object Management Group, *Model driven architecture*, V. 1.0.1; 2003
[Pargas, 1999]	R.P. Pargas, M.J. Harrold , R. Peck; T*est data generation using genetic algorithms*. Software Testing, verification and reliability; 1999.
[Schroeder, 2002]	P.J Schroeder, P. Faherty, B. Korel; *Generating expected results for automated black-box testing*; Automated Software Engineering, 2002. Proceedings. ASE 2002. 17th IEEE International Conference on, pp.139,148; 2002 (DOI: 10.1109/ASE.2002.1115005)
[Srinivas,	M. Srinivas ,L.M. Patnaik; *Genetic Algorithms : a survey*. IEEE computer, 17-26; 1994.

1994]	
[Tonella, 2004]	P., Tonella; *Evolutionary testing of classess*; ISSTA '04: proceeding of the 2004 ACM SIGSOFT international symposium on Software Testing and Analysis, 119.128, 2004.
[Tretmans, 2003]	J. Tretmans, E. Brinksma; *TorX : Automated Model Based Testing*; First European Conference on Model-Driven Software Engineering, 31-43; 2003.
[Utting, 2012]	M. Utting, A. Pretschner, B. Legeard; *A taxonomy of model-based testing*; STVR 22:5; 2012.
[Zander, 2011]	J. Zander, I. Schieferdecker, and P. J. Mosterman; *A Taxonomy of Model-Based Testing for Embedded System;* 2011