 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 FRO M SPECIFIC NATIONAL PROGRAMS AND / OR FUNDING AUTHORITIES.

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

Use Case Definition
D502.010

DOCUMENT INFORMATION

Project	CRYSTAL
Grant Agreement No.	ARTEMIS-2012-1-332830
Deliverable Title	Use Case Definition
Deliverable No.	D502.010
Dissemination Level	CO
Nature	R
Document Version	V1.01
Date	2013-11-04
Contact	Christoph Scherrer
Organization	Thales Austria Gmbh (TRAIL)
Phone	+43 1 27711 5864
E-Mail	christoph.scherrer@thalesgroup.com

AUTHORS TABLE

Name	Company	E-Mail
Johannes Mach	TRAIL	johannes.mach@thalesgroup.com
Christoph Scherrer	TRAIL	christoph.scherrer@thalesgroup.com
Rupert Schlick	AIT	rupert.schlick@ait.ac.at
Egbert Althammer	AIT	egbert.althammer@ait.ac.at
Thomas Gruber	AIT	thomas.gruber@ait.ac.at

CHANGE HISTORY

Version	Date	Reason for Change	Pages Affected
V1.00	2013-10-18	Version for external review	ALL
V1.01	2013-11-04	Revised version for release	ALL

CONTENT

List of Figures

List of Tables

1 Introduction

1.1 Role of deliverable

This document (D5.2.1) is the first deliverable of the WP5.2 of the CRYSTAL project. It reports the current state of definition of the Use Case 5.2 “Integrated Modelling of Core Algorithms in TAS Control Platform”, as agreed between the industrial partner (TRAIL) and the tool provider (AIT).

The focus of the use case definition is put on the detailed description of the considered modelling technologies. Furthermore it contains a first, preliminary set of functional requirements on the tools and methods (called technology bricks in CRYSTAL).

1.2 Relationship to other CRYSTAL Documents

The functional brick requirements contained in this document will be the basis of the interface requirements analysis that will result in the UC 5.2 bricks interface requirements document (D5.2.2).

Together, D5.2.1 and D5.2.2 will form the basis for setting up the system engineering environment (SEE) for UC 5.2 to be reported in the implementation and integration report (D5.2.3).

It will also be used as reference for the SEE and bricks assessment phase that will be delivered together with the demonstrator D5.2.4.

1.3 Structure of this document

Chapter 1 is this introduction.

Chapter 2 summarizes the context of UC 5.2 including a brief overview of the TAS Control Platform and a rough description of the target development.

Chapter 3 explains the goals and motivation for the introduction of the modelling technologies that will be investigated in this use case.

Chapter 4 provides a detailed description of the envisaged modelling technologies and foreseen tools.

Chapter 5 describes the foreseen integration of new methods into the UC 5.2 development process.

Chapter 6 contains a preliminary list of functional requirements for these methods and tools (bricks).

Finally, terms and abbreviations are explained in chapter 7, whereas references are given in chapter 8.

2 Use Case Context

2.1 Overview of TAS Control Platform

The idea of TAS Control Platform (TAS PLF) is to build railway applications on top of a generic computing platform and thereby support the fulfilment of the overall RAMS requirements. This separation reduces the direct dependency of long-lived railway applications on short-lived hardware.
The TAS Control Platform includes all necessary generic elements for constructing replica deterministic, fault-detecting 2oo2 ("2-out-of-2") or fault-tolerant 2oo3 architectures. A basic set of services is provided for 1oo1 architectures, too. The services include a global time base, voting, membership, recovery and fault management services, as well as a health monitor. The use of POSIX as generic interface for applications enables the transparent integration of fault detection mechanisms. Applications can build upon the generic TAS Control Platform safety concept and safety case to reach their safety goals.

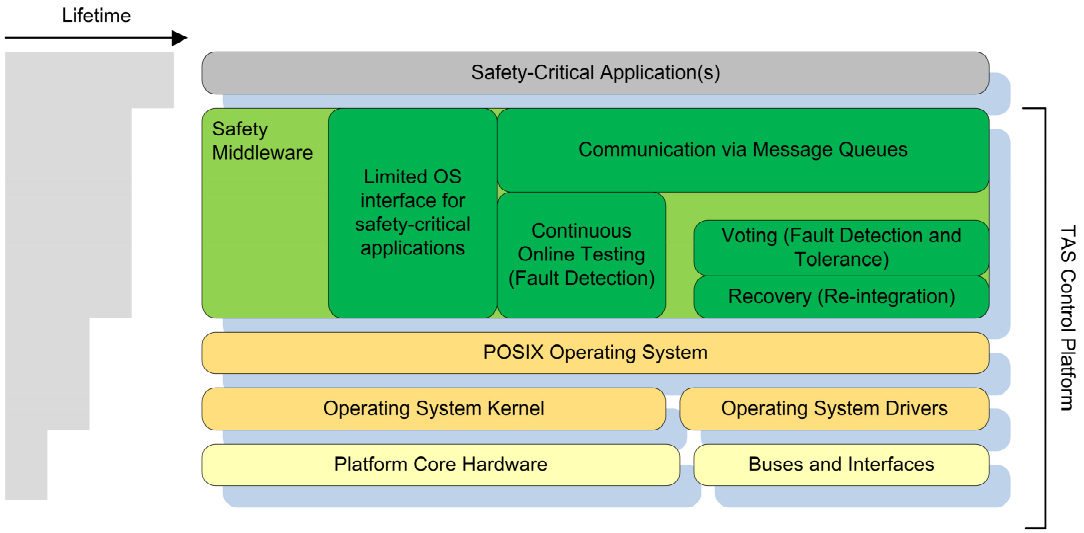

Figure 2-1: The TAS PLF layer structure: The safety middleware layer (light green) consists of several components (dark green).

The TAS Control Platform itself is structured in layers as depicted in Figure 2-1. The safety middleware layer enables safe application execution on top of a common off-the-shelf POSIX operating system with its kernel and drivers. Within the safety layer, all safety relevant platform services are executed to ensure detection and/or isolation of any faults occurring in the layers below.
The figure also shows the relative lifetimes of components in the individual layers. This illustrates how the safety and operating system layers are used to achieve a long lifetime for the interface to the application, as well as the application itself.
The common time base, membership and voting services are implemented within the safety layer. They are provided to the application via a message queue interface. Continuous online testing is performed by the health monitor service and recovery is implemented as a separate service.

The applications on top of the layered architecture provide the actual services. The platform can currently be operated in three different redundancy configurations, as shown in Figure 2-2.

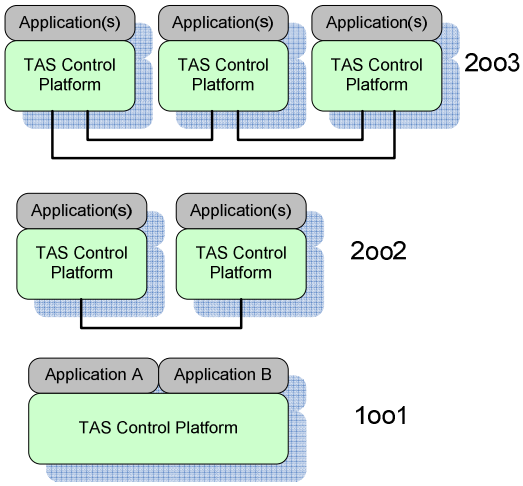

Figure 2-2: TAS PLF redundancy configurations

To guarantee system availability, the safety layer also provides the functionality to re-integrate a failed node into the system ("recovery"). A failed node that is re-integrated after a reset or after a hardware replacement has to be brought into the same state as the other nodes. This also includes replicated applications, which are recovered transparently during operation without interruption of service.
To ensure that no latent faults are aggregated in the hardware, the platform also performs continuous online testing of the hardware. This online testing, which is performed by a background task, covers the CPU, memory, buses, clocks, and disks.

2.2 Target Development

UC 5.2 will deal with the modelling of typical safety-relevant functions in railway control systems, e.g.:

(1) Safe communication protocol according to CENELEC standard EN 50159

(2) Synchronization algorithms for replicated state machines

(3) System monitoring and diagnosis functions

The exact development scope for UC 5.2 will be selected according to the expected effort and the suitability for demonstration.

3 Use Case Goals

3.1 Business Goals

In agreement with the CRYSTAL project goals, Thales Austria pursues the following business goals that shall be achieved by means of model-driven engineering (MDE).

(1) **Efficiency:** Accelerate overall development process, improve time-to-market

(2) **Safety:** Increase efficiency of safety engineering

(3) **Quality:** Improve SW and documentation quality

(4) **Maintainability:** Improve design for maintainability

3.2 Technical Goals

In order to achieve these business goals, several technical goals have been identified that shall be addressed within UC 5.2 according to their associated priority and available resources (see Table 3-1).

Please note: Goals with medium or low priority (T4, T5) are OPTIONAL for UC 5.2. These goals will be addressed only if sufficient resources are available. The actual effort needed for achieving the high priority goals (T1, T2, T3) cannot be accurately predicted. Therefore the decision has to be taken at a later stage.

ID	Goal	Description	Priority
T1	Support of safety analysis	Safety analysis shall be supported by modelling techniques (beginning from functional analysis / architectural design) by e.g.: + Safety viewpoint + Efficient generation of FMEA, FTA + Integration of safety requirements & SAC	HIGH
T2	Automatic test case generation	Automatic generation of system / component test cases based on test model or specification model	HIGH
T3	Support of verification and validation	Support of V&V planning and management e.g. by means of: + Guidance for V&V planning according to standards + Systematic checklist for validation + Verification status monitoring + Traceability to verification evidence	HIGH
T4	Model-based code generation	Code generation e.g. from a SCADE implementation model	MEDIUM
T5	Formal specification	Formal system specification e.g. using Rodin/Event-B: + consistent and complete specification + verifiable by formal proof	LOW

Table 3-1: Technical goals of Use Case 5.2

All modelling artefacts shall support traceability, enabling at least the following traceability links:

+ From user requirements to system requirements
+ From system requirements to related component requirements
+ From component requirements to design elements and source code
+ From requirements to test cases

4 Modelling Methodologies

4.1 Model-Based Safety Analysis

4.1.1 General Concept

One major objective of UC 5.2 is model-based safety analysis (MBSA), in particular FMEA and FTA analyses. The basis for this objective will most probably be a “safety viewpoint” of the system model, which shall represent all safety-relevant properties and interactions of system components. It is expected that this representation will use a dedicated safety meta-model (or profile), which allows to express e.g. fault probabilities, possible fault propagation, fault detection etc.

4.1.1.1 Safety Viewpoint Approach

In the viewpoint approach different views of a system can be created and analysed separately but still refer to the same set of components, thereby ensuring consistency between e.g. the safety viewpoint and the architectural design viewpoint [Thomas, 2011]. For example, it is not possible to add a redundant component in the safety analysis without adding it in the architecture, too.

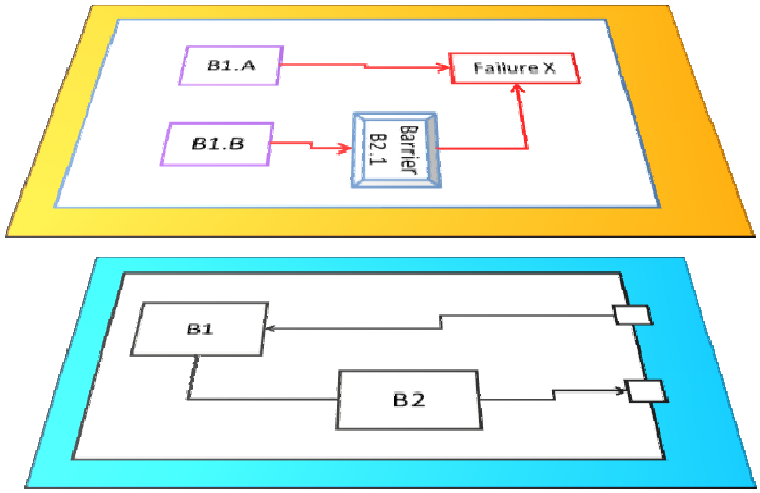

Figure 4-1: Example illustration of viewpoint approach

4.1.1.2 Technical Goals

It is expected that the efficiency of the RAMS process, which is regulated by [EN-50126], can be significantly improved by means of model-driven engineering. Therefore, the following technical goals are addressed in UC 5.2:

1. Tool-based support for FMEA:
   - Derive basic structures of FMEA from the (architectural/functional) system model based on a set of predefined fault models (depending on component / function type), see [IEC 60812]. These predefined fault models (e.g. “too early”, “too late”, “no signal”) shall be derived from an analysis of available information about existing approaches.
   - Perform semi-automatic effect analysis based on knowledge about fault propagation
   - Traceability links from failure modes to SAC (safety application conditions)
   - Traceability links from failure modes to safety requirements
   - The focus is on a qualitative analysis rather than on a quantitative one.
   - Generally, the focus is on software, but hardware failures are in scope, too.

2. Tool-based support for FTA:
    - Treat also the effects of combined (multiple) faults by enabling tool support for fault tree analysis (FTA), see also [IEC 61025].
    - Traceability links from (combined) failure modes to SAC (safety application conditions)
    - Traceability links from (combined) failure modes to safety requirements
    - The focus is on a quantitative analysis, but simple qualitative analysis shall be possible, too.
3. Forcing consistency between functional system model and safety viewpoint:
    - Provide consistency checking tool: In the first run a consistency checker tool shall be selected (from existing ones) or designed (newly). Most important is that barriers are directly traceable to the respective items in the system model.
    - Further, analyse the possibilities for creating the safety viewpoint automatically out of the (architectural/functional) system model: According to available time and budget design a tool for automated creation of the safety viewpoint can be added (at a later stage). This can help to improve efficiency of the process by effort reduction through automation.
4. Visualization of the safety aspects by a safety viewpoint
    - Define a suitable meta-model (or profile) to express safety properties and relations, e.g. failure causes, failure modes, fault probabilities, fault propagation, barriers, etc
    - This model shall be visualized, on the one hand supporting safety analysis and on the other contributing to the safety case.
5. Tool support for designing safety mechanisms
    - This activity is optional and will be carried out depending on available time and budget.
    - A tool can propose appropriate barriers where - according to the safety analysis - part of the system violates safety conditions
6. Impact analysis of changes
    - Changes on the system model, on subsystems and components, on requirements or on SACs have an impact on the validity of the safety analysis. Based on the implemented full traceability, any influence can be tracked to all affected elements in the safety analysis.
    - As a result of a respective check function, the affected parts of the safety analysis can be annotated or recalculated.
7. Interoperability
    - Models for software elements are mostly created with Eclipse tools, for which traceability to the requirements in DOORS is important. It is intended to realize the coupling between Eclipse and DOORS through an OSLC interface. In general, the use of a standard OSLC interface shall enable to use other requirements databases than DOORS.

4.1.2 Tool Description (MB-RAMS)

Generally, the model-based (MB) RAMS process shall be achieved mainly by re-using existing MB tools and where necessary adapting them to the specific needs of the use case. The tools shall be integrated with WEFACT where appropriate (see also chapter 4.3.2).

MB-RAMS activities start from an architectural and functional system model expressed in UML/SYSML. For this purpose using the shareware tool Papyrus Eclipse is foreseen. However, also UML/SYSML models created and maintained with IBM Rational Rhapsody shall be integrated.

The FMEA shall be realized as a structured list, for which an appropriate tool will be selected. One of the first choices there is the Eclipse plug-in ProR. The Failure Cause objects contained in this list shall be associated with the respective attributes (according to the FMEA process) and linked to requirements and SACs (safety application conditions).

A dedicated safety meta-model (or profile) for the safety viewpoint will be designed, covering all relevant properties like fault probabilities, possible fault propagation, fault detection, etc.

A tool for creating the safety viewpoint will be selected and adapted if required. Tools like the Eclipse extension OBEO will be assessed with respect to their applicability for creating and maintaining the safety

viewpoint. Particular attention will be dedicated to the aspect of traceability to requirements, architecture elements, FMEA entries and safety application conditions (SAC).

Figure 4-2 in section 4.1.3 depicts the use of the tools with the various models and views.

4.1.3 Data Flow

Figure 4-2 shows the data flow for MB-RAMS activities and the tools used in the MB-RAMS activities.

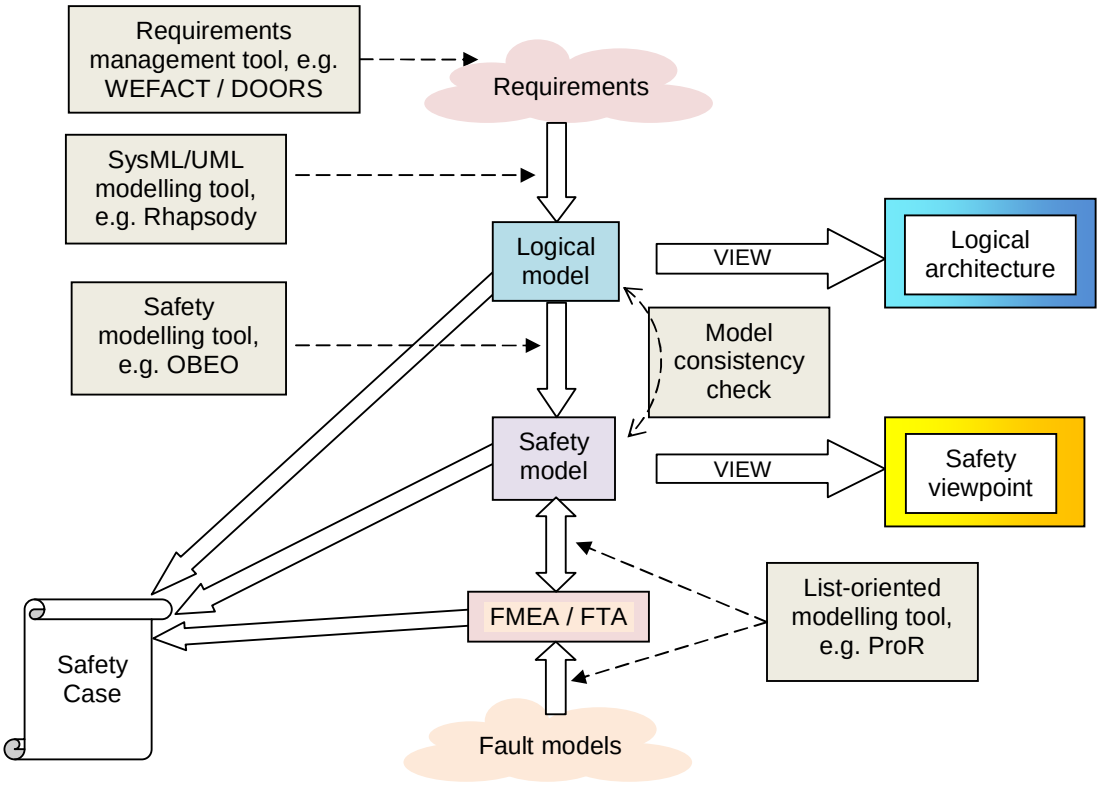

Figure 4-2: Data Flow for MB-RAMS activities

Wide arrows represent the data flows; dashed arrows show for which data flows or checks tools are used. Note that traceability is not depicted here but in Figure 5-2.

4.2 Model-Based Test Case Generation

4.2.1 General Concept

Another major objective of UC 5.2 is the automatic, model-based generation of test cases. The basic logic of this approach is explained in Figure 4-3. In this approach, instead of manually writing and implementing test cases, they are automatically derived from a model. Test cases are defined in terms of sequences of input and output events of the system under test (SUT). The input events are used to stimulate the SUT in the test run, the output events serve as reference for the test case verdict (pass/fail decision).

There are two possibilities: (1) If a separate test model is created independently from the implementation model, the tests will check not only the implementation (e.g. code generation) but also the correct interpretation of requirements in the system model. (2) If the system model is used also as test model, only the code generation step will be verified. In that case other measures have to be taken to strengthen the verification of the system model (e.g. formal verification, model checking).

In Figure 4-3 these two alternatives are indicated by green and violet arrows.

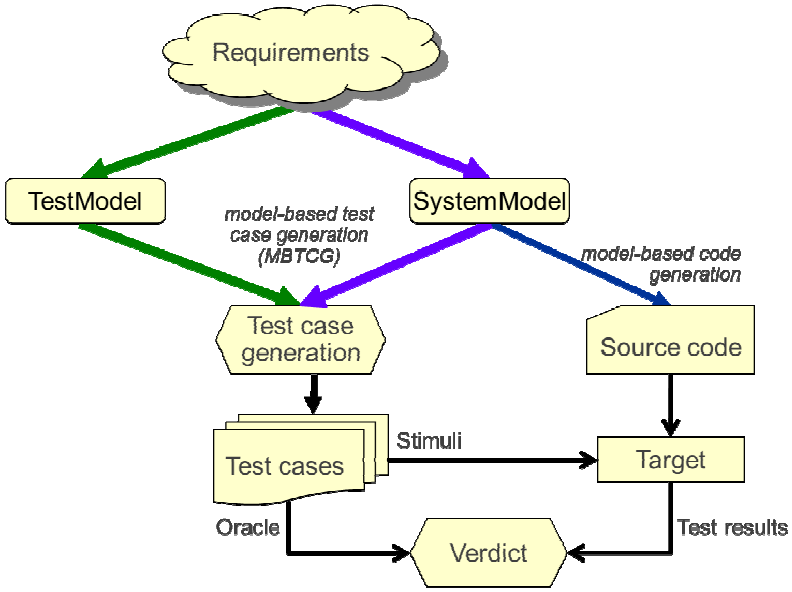

Figure 4-3: Logic of model-based test case generation

It is possible to combine test case generation from a test model with test case generation from an implementation (either code or implementation model), thereby achieving both requirements and code coverage with an overall optimized test suite.

There are various techniques for model-based test case generation. The test case generation tool to be used in UC 5.2 is based on the discrimination of model mutants. An overview of this tool will be given in the following section.

4.2.2 Tool Description (MoMuT::UML)

MoMuT::UML uniquely combines a powerful fault-based test case generation strategy with standard techniques to deliver high quality test suites with an excellent cost/benefit ratio. The heart of this new technology is the concept of fault seeding or mutation. Figure 4-4 depicts our underlying model-based mutation testing approach: *MoMuT::UML* uses customizable mutation operators to derive mutated models from the original test model. A mutated model is an exact copy of the original minus one change introduced by the mutation operator. Given a mutant and the original specification, the backend searches for a sequence of inputs and outputs that uncovers any design refining (“implementing”) the mutant instead of the

original. It is in the nature of mutation-based test case generation that one such sequence, i.e. test, finds ("kills") multiple mutated models and, hence, has the ability to find faults that are not directly modelled by a mutation operator.
Mutation-based test case generation is the most fine-grained and versatile test generation technique available today. In principle, mutation-based test case generation can not only be used to test functional properties of designs but also to generate tests that detect certain non-functional defects. It also allows *MoMuT::UML* to know exactly which faults are caught by a particular test case, analyze or extend existing test sets, and help localizing faults by (a) automatically selecting a set of mutants that can explain faulty behaviour and (b) creating a short test case to help with debugging.

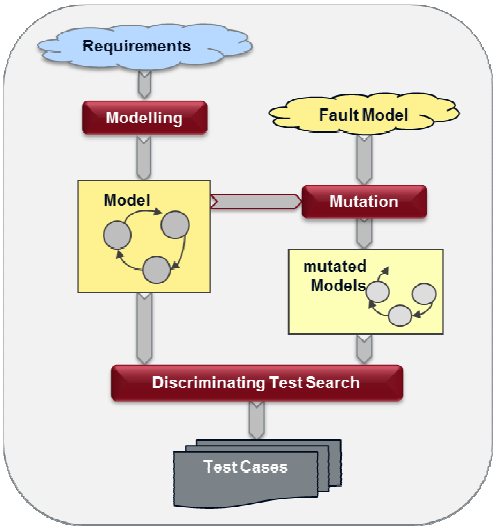

Figure 4-4: Mutation-based test case generation is at the heart of MoMuT::UML

MoMuT::UML either uses the *ioco* (input-output conformance) relation or the refinement relation. It translates UML to an internal representation with clearly defined semantics (action systems). The frontend can connect to state-of-the-art model checkers for further design verification and is able to trigger both, the concrete and the symbolic test case generation backend. The backends also support model animation features.

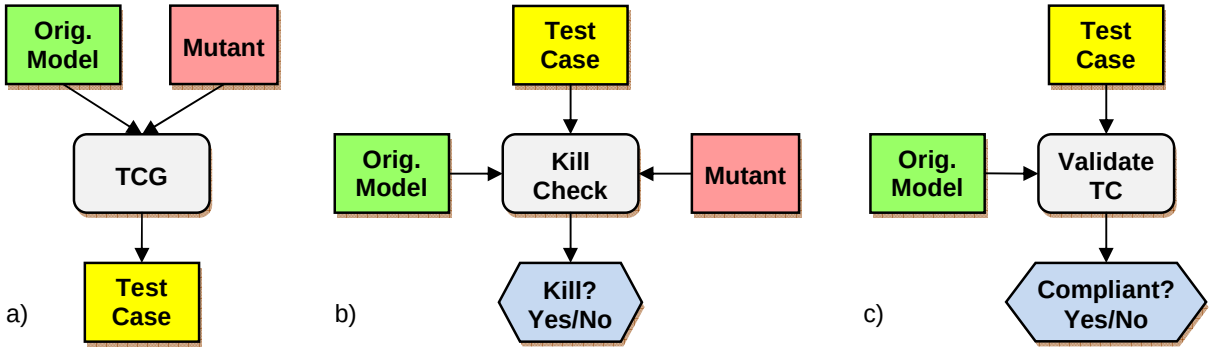

Figure 4-5: Usage variants of mutation-based TCG tools
a) generate test case for killing a certain mutant
b) check whether an existing test case kills a mutant
c) check whether an existing test case complies with the original behaviour

As shown in Figure 4-5, there are multiple ways to use the tool – generating the test cases for a mutant, checking the quality of test cases by checking how many mutants they kill and using the model as an oracle to decide which behaviours are compliant with the model.
By combining variants a) and b), pre-existing test cases can be evaluated for their mutation coverage and only test cases for the missing mutants are created then. The pre-existing test cases can be legacy test cases, can come from other tools (including white-box test case generation) or can come from prior system iterations.

Figure 4-6 shows an example screenshot of the graphical user interface of MoMuT::UML, which was produced with the symbolic TCG backend.

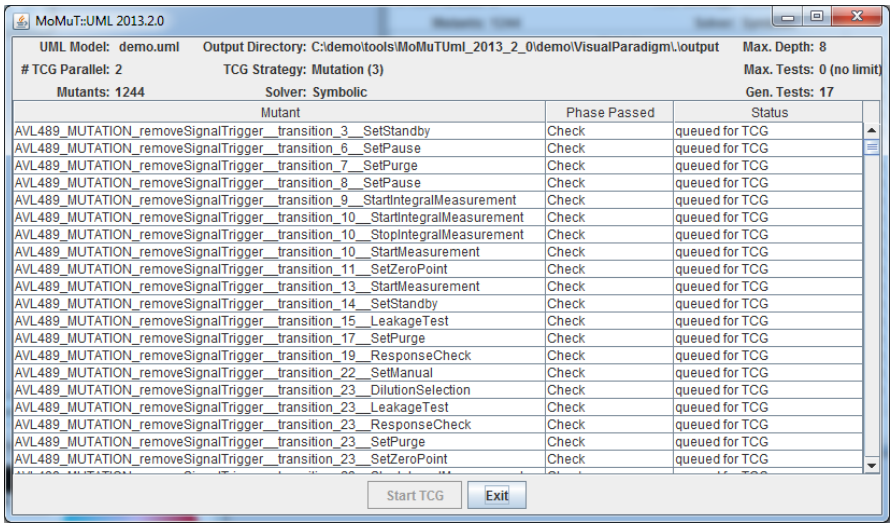

Figure 4-6: Screenshot of the MoMuT::UML tool

4.2.3 Data Flow

The following artefacts are involved in the envisaged data flow of model-based testing:

Input to TCG:

(1) Test model (System model including traceability from requirements to model objects)
(2) Test configuration (definition of model variables, instantiation, etc.)
(3) Pre-existing test cases (optional)

Output of TCG:

(4) Abstract test cases (sequences of input / output events)
(5) Traceability matrix test case to requirements

The role of these artefacts is depicted in the following data flow diagram.

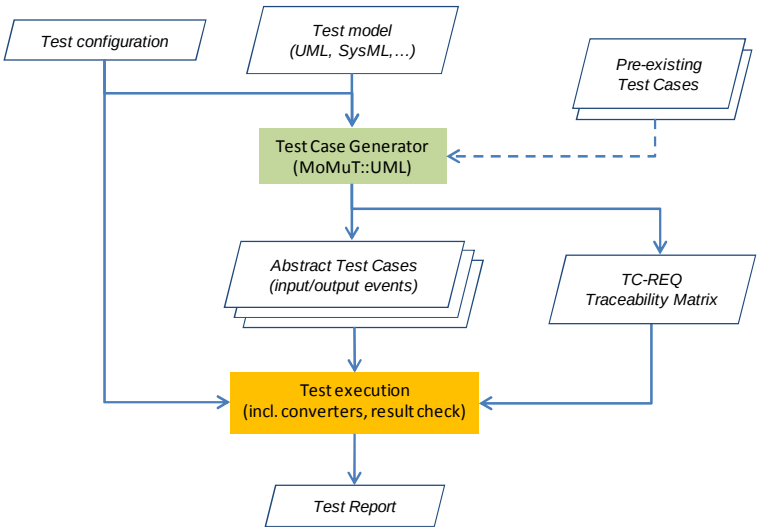

Figure 4-7: Automatic test case generation and related data flows

The test execution may contain several stages, including

- test case translation (to specific test scenario and golden trace files)
- actual test run
- test result check (verdict)
- test report compilation

MoMuT::UML can be configured by means of a configuration file that controls the test case generation. This file defines e.g. the mutation operators and the mutated elements.

Information about failing test cases can be fed back into the test case generator for generation of short test cases to isolate the problem.

4.3 Verification & Validation Management

4.3.1 General Concept

The idea of the V&V management tool is to support the following activities

- Define verification activities (tests, analyses, reviews) according to applicable standards
- Trace and check verification evidence
- Monitor status of verification activities
- Check completeness of verification activities
- Re-validate documents and test results after changes

In addition, the tool can serve as systematic guidance for validation and safety audits. For this purpose, the envisaged V&V management tool should have the following features:

- Systematic guidance through requirements of applicable CENELEC standards
- Creation and maintenance of a hierarchical list of verification activities
- Edit verification status of each defined activity
- Create traceability links to elements of other artefacts (paragraph in document, test reports, etc.)
- Detect need for re-validation of documents / test results
- Generate overall statistics of verification status

4.3.2 Tool Description (WEFACT)

The “Workflow Engine for Analysis, Certification and Testing” (WEFACT) [Erwin Schoitsch, 2006] has the goal to facilitate validation, verification and certification of safety-critical systems in a modular manner.
WEFACT consists of the WEFACT framework which provides a flexible infrastructure for defining and executing the V&V process and the external resources – external processes, tools and standards – which are integrated into the WEFACT framework by well-defined interfaces. Additionally, an extensive on-line user guide (“help file”) including a v-plan cook book (“How to develop a v-plan”) is available.
The overall organization of the WEFACT framework is shown in Figure 4-8. The gray boxes show the elements of the WEFACT framework, the white boxes show the rest of the elements of the WEFACT (belonging to the external systems), vertical alignments indicate ‘uses’ or ‘consists of’ relationships whereas the arrows indicate major information flows.
The safety case is an argumentation to convince a licensing authority that a product is “sufficiently safe”. Typically a safety case comprises the necessary safety arguments which correspond to the validation plans (v-plans) for each artefact under test (AUT) and the related evidence.

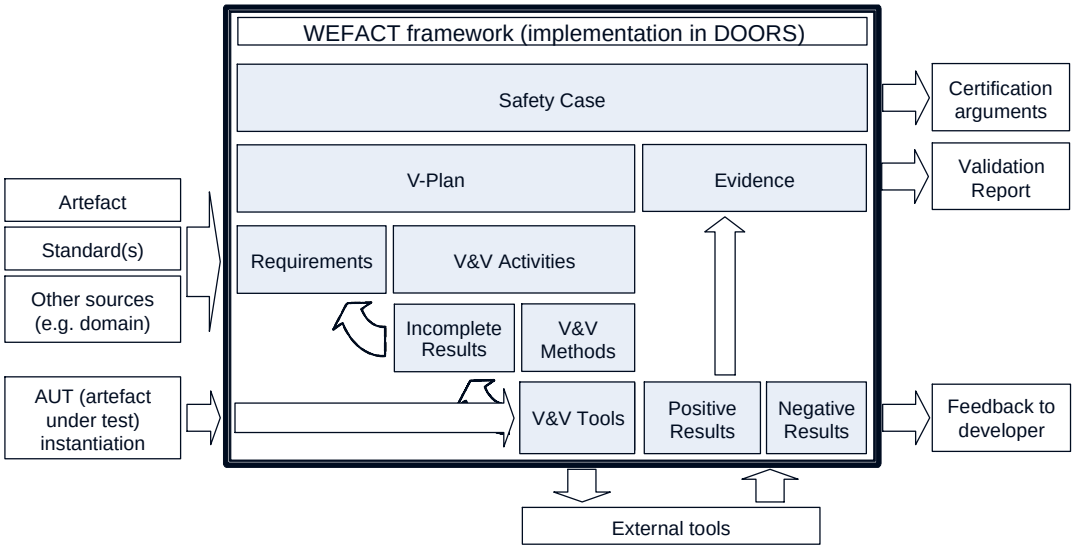

Figure 4-8: Overall organization of the WEFACT framework

The WEFACT framework is implemented with IBM Rational DOORS which is based on distributed client/server architecture. The data such as v-plans, V&V activities, requirements is stored in the central DOORS database whereas the documents such as evidence and reports are stored in a separate document repository. In order to setup the WEFACT framework for a user, he or she installs the DOORS client in order to have access to the data and sets up the access to the document repository.

4.3.3 Data Flow

The WEFACT tool does not exchange data with other tools but rather manages links and references to elements of other documents, data base records, model elements, etc.

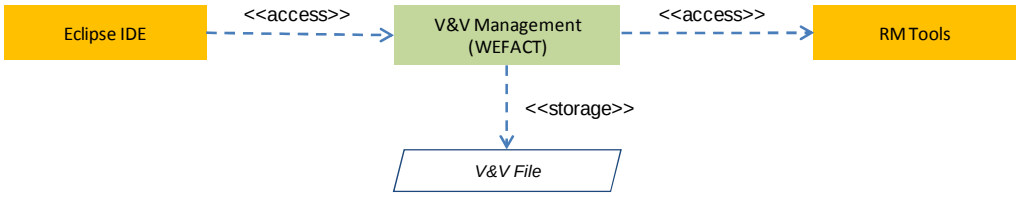

Figure 4-9: Dependencies of V&V management on other artefacts

Based on the CRYSTAL IOS, the tool will interface with any other requirements management tool (not only DOORS). Access from Eclipse tools to the V&V management will also be possible.

4.4 Code Generation (OPTIONAL)

To close the gap between design modelling and SW code, another goal for UC 5.2 is the integration of automatic code generation from a design / implementation model. However, this goal is optional and will be addressed in UC5.2 only if sufficient resources are available.

The tool that is currently considered for code generation is Esterel's SCADE Suite. It comes with a C code generator that is certified according to several international safety standards, e.g. [EN-50128].

The SCADE modelling language is a data-flow oriented modelling language that combines state diagram and activity diagram elements in a common diagram. The model is structured by operators which can be reused by instantiation. The model is processed in a fixed processing loop, which provides unambiguous runtime semantics. It therefore represents a platform independent model (PIM).

More information can be found e.g. in the SCADE Suite's online documentation [SCADE].

4.5 Formal Specification (OPTIONAL)

Another innovative modelling method, which will be optionally integrated into UC 5.2, is model-based, formal specification using the formal specification language Event-B together with the Eclipse-based development platform RODIN.

Formal specification using Event-B is able to provide a consistent and complete system specification that can be verified by formal proof. Complex systems can be managed using the principles of abstraction and refinement. Please refer to [Abrial, 2009] or Abrial's textbook [Abrial,2010] for more detailed information.

Recently emerging graphical modelling tools for Event-B, e.g. UML-B [Snook, 2008], have increased the usability of formal specification in engineering of complex systems.

The RODIN Platform is an Eclipse-based IDE for Event-B that provides effective support for refinement and mathematical proof. The platform is open-source, contributes to the Eclipse framework and is further extendable with plug-ins [Event-B, RODIN].

5 Integration of Processes

A major goal of CRYSTAL is to smoothly integrate the new MDE methods and artefacts into the overall development processes.

For this two different kinds of integration between artefacts have to be considered:

(1) **Data processing interfaces:** A tool can access and process data of another tool. (In case of inter-operability two different tools can process the same artefact.)

(2) **Traceability:** The elements of an artefact can carry references to the elements of another artefact.

The following sub-sections summarize the interfaces and traceability links that are relevant for UC 5.2.

5.1 Data Processing Interfaces

Figure 5-1gives an overview of the main tools and artefacts of the UC 5.2 development process. New artefacts that shall be integrated within CRYSTAL are highlighted in cyan. The yellow boxes stand for conventional model artefacts.

Furthermore, the figure shows the new data processing interfaces between tools. Those that are foreseen for automatic processing are marked by green connectors. Semi-automatic processing, which is going to be applied for FMEA and FTA generation, is marked by orange arrows.

Please note that the usage of model-based test case generation for integration testing is technically possible. However, its practical usefulness has to be further analyzed.

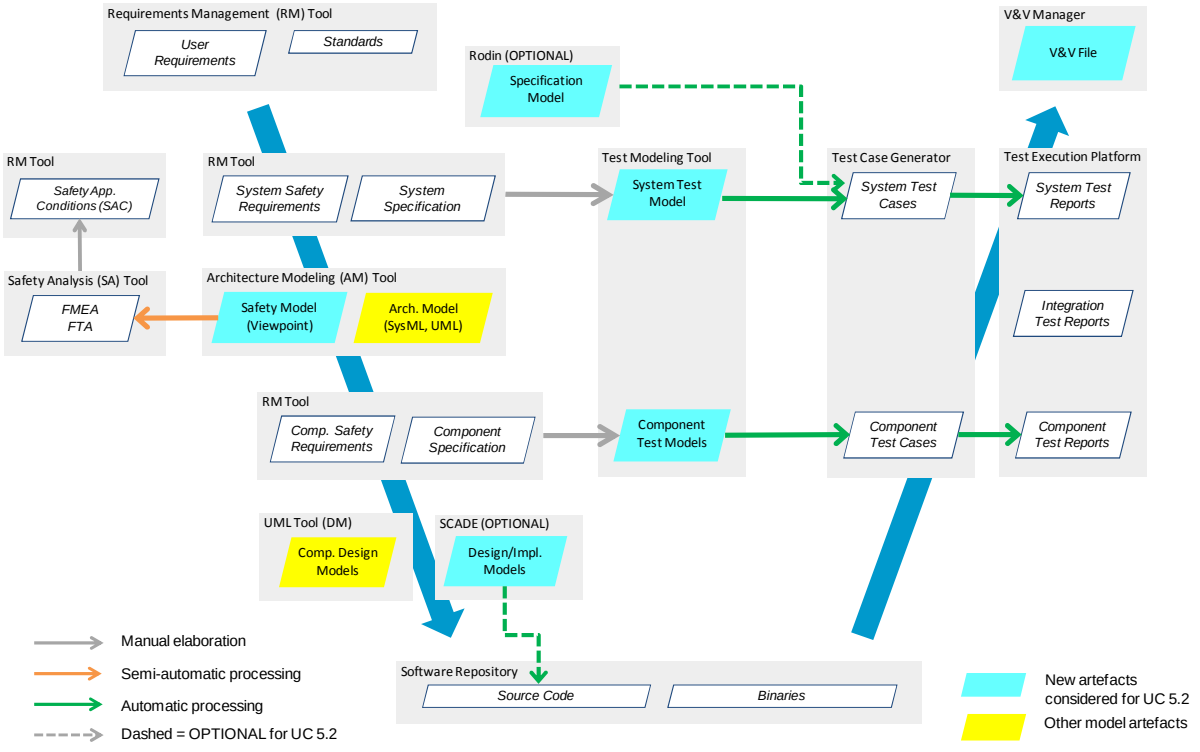

Figure 5-1: Overview of UC 5.2 development process showing data processing interfaces

5.2 Traceability

Traceability between artefacts is of ultimate importance for the engineering of complex and safety-critical systems. This applies even more for traceability with or from model artefacts.

Traceability does not only consist of the storage of links. The tools must also support tracing, i.e. following the links and also relating objects via multiple links. For example, the following traceability capabilities will be useful in UC 5.2:

- Tracing failed V&V activities to requirements
- Tracing failed test cases to model elements in the test model
- Tracing successful V&V activities to fulfilled (safety) requirements

There are two types of traceability links that have to be distinguished in order to avoid accidental corruption and inconsistency of traceability links:

- Primary links: Links that must initially be established by an engineer.
- Derived links: Links that are derived from primary links and cannot be edited.

The following subsections describe the necessary traceability links that have been identified for the high-priority bricks (referring to goals T1-T3) in a preliminary analysis. Primary links are represented by red connectors, derived ones by black ones with a dashed line.

5.2.1 Traceability for Model-Based Safety Analysis

The primary and derived traceability links that are relevant for MB-RAMS are depicted in Figure 5-2.

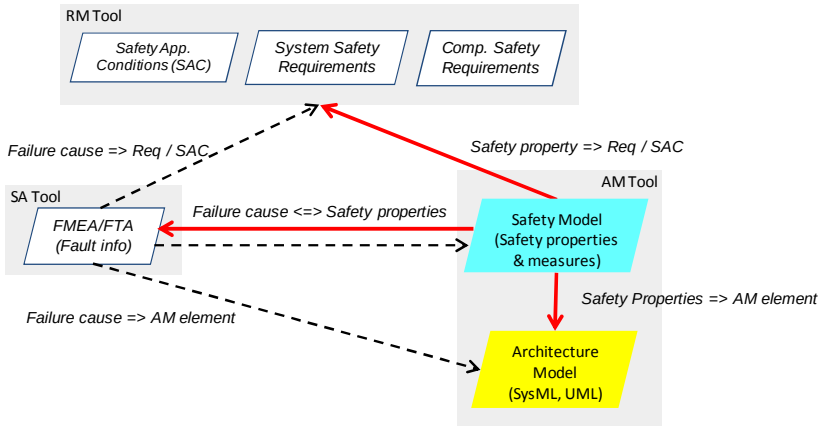

Figure 5-2: Relevant traceability links for model-based safety analysis

5.2.2 Traceability for Model-Based Test Case Generation

The primary and derived traceability links that are relevant for MoMuT::UML are depicted in Figure 5-3.

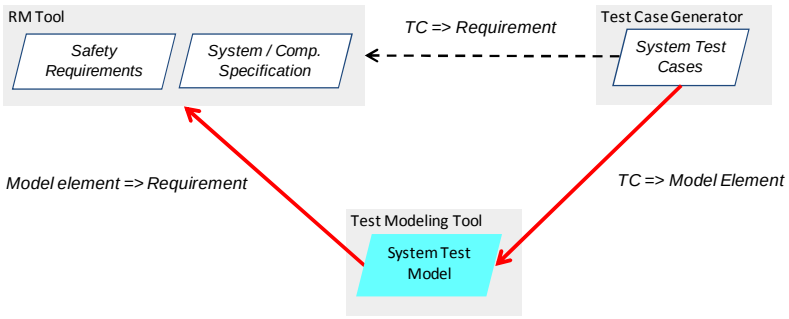

Figure 5-3: Relevant traceability links for model-based test case generation

5.2.3 Traceability for V&V Management

The V&V file is essentially a large collection of traceability links to verification targets (e.g. a paragraph of a safety standard) and to verification evidence items. Therefore the V&V Management Tool has to support various kinds of traceability links, e.g. to:

- entire artefacts (version)
- individual requirement objects
- model objects
- source code objects
- text documents (or sections of it)

An overview of the applicable traceability links that are relevant for WEFACT is given by Figure 5-4.

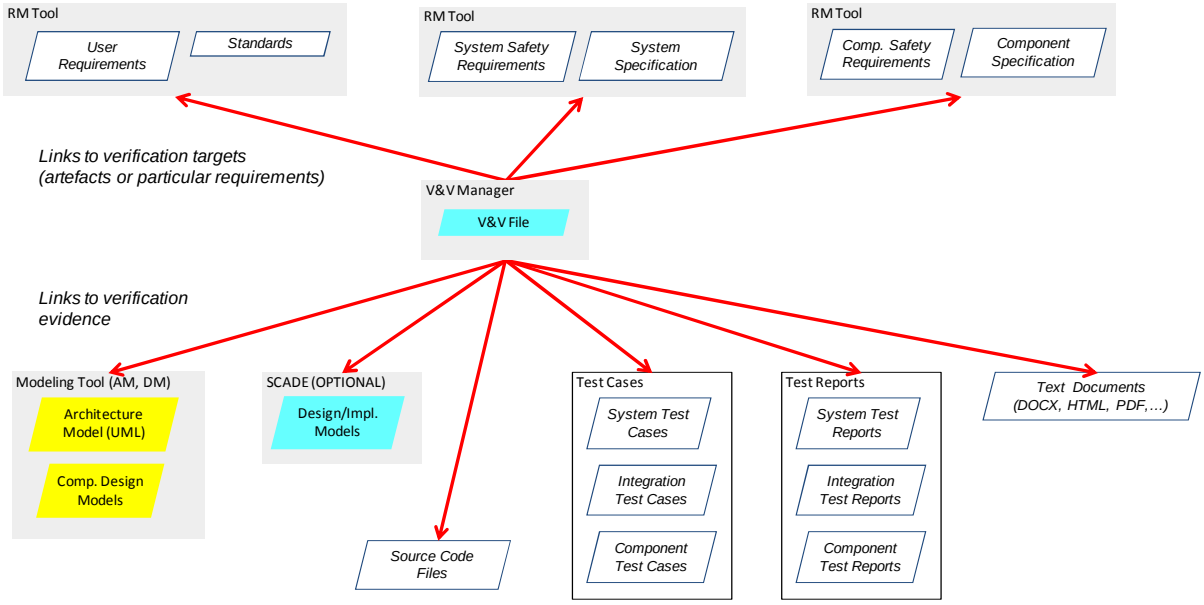

Figure 5-4: Relevant traceability links for verification & validation management

6 Functional Requirements on Modelling Tools

6.1 General and Interoperability Requirements

ID	Title	Description	Priority
GEN-01	Eclipse Integration	Thales Austria aims for integration of as many development related task types as possible into their Eclipse based development environment. The need for switches to other tools/work environments for the tasks addressed in the SEE shall be minimized (within reasonable effort).	HIGH
GEN-02	RM agnostic integration	Integration with requirement management shall be transparent with respect to the used Requirements Management Tool and the real location of the requirement for other tools in the SEE.	HIGH
GEN-03	Traceability between MDE artefacts and code	Traceability shall be granted for all levels of requirements and associated artefacts down to the source code level	HIGH

6.2 Model-Based Safety Analysis (MB-RAMS)

ID	Title	Description	Priority
MBR-01	Safety model tool support	Tool support for creating the safety model	HIGH
MBR-02	Safety model check	Consistency and completeness check between architectural/functional model and safety model	HIGH
MBR-03	Safety model visualisation	Visualization of the safety model (Safety Viewpoint)	HIGH
MBR-04	FMEA structures	Deriving structures for a qualitative FMEA from system model and fault models.	HIGH
MBR-05	Safety model traceability	Realize full traceability between (1) all items in the safety model including safety properties and relations (2) functions and components or elements in the system model, (3) the safety requirements and (4) objects of the FMEA and FTA.	HIGH
MBR-06	FMEA generation	Semi-automatic generation of qualitative FMEA including effects, based on system model, fault models and safety model	HIGH
MBR-07	FTA generation	Semi-automatic generation of FTA for treating effects of multiple faults based on system model and fault models	MEDIUM
MBR-08	Impact analysis	Tool-based support for analysing the impact of changes in requirements, functional/architectural model or components used on the safety model	MEDIUM
MBR-09	Safety model generation	Automatic creation of the safety model from system model, FMEA, FTA and SACs.	LOW

MBR-10	Safety mechanisms tool	Provide tool support for (a) checking implemented safety mechanisms for consistency with safety model and safety requirements, or (b) proposing or creating respective safety functions (barriers) to cope with the safety requirements	LOW

6.3 Automatic Test Case Generation (MoMuT::UML)

ID	Title	Description	Priority
TCG-01	Generate test cases from a UML test model	Generate test cases from a system test model in UML (Black Box Testing)	HIGH
TCG-02	Coverage selection	Select coverage for test cases by mutation operators, related requirements, and model elements.	HIGH
TCG-03	Test model elements traceability	Relate test model elements to requirements (safety or not)	HIGH
TCG-04	Test case - model traceability	Relate test cases to model elements	HIGH
TCG-05	V&V activity - test case traceability	Relate V&V activities to test cases	HIGH
TCG-06	Generate component test cases (Black Box Testing)	Generate test cases from a component test model	HIGH
TCG-07	Generate test cases from an implementation model	Generate test cases from an implementation model in SCADE	MEDIUM
TCG-08	Incremental test case generation	Extending component test cases to integration or system test cases.	MEDIUM

6.4 Verification & Validation Management (WEFACT)

ID	Title	Description	Priority
VVM-01	V&V success tracing	Trace successful V&V activities to fulfilled (safety) requirements	HIGH
VVM-02	V&V fail tracing	Trace failed V&V activities to requirements	HIGH
VVM-03	TCG fail tracing	Trace failed test cases to model elements in the test model	HIGH
VVM-04	Standards guidance	Systematic guidance through requirements of applicable CENELEC standards	HIGH
VVM-05	Hierarchical V&V activities list	Creation and maintenance of a hierarchical list of verification activities	HIGH
VVM-06	Verification status input	Edit verification status of each defined activity	HIGH
VVM-07	Creation of traceability links	Create traceability links to elements of other artefacts (paragraph in document, test reports, etc.)	HIGH
VVM-08	Detection of re-validation need	Detect need for re-validation of documents / test results	HIGH

VVM-09	Verification status statistics	Generate overall statistics of verification status	MEDIUM
VVM-10	Artefact versioning	Support versioning of associated artefacts	HIGH
VVM-11	V-plan version control	Support version control of V-plan	HIGH
VVM-12	Traceability for new artefacts	Transfer link to new version of target artefact	MEDIUM

7 Terms, Abbreviations and Definitions

Please add additional terms, abbreviations and definitions for your deliverable.

AM	Architecture Model
CRYSTAL	**Cr**itical S**YST**em Engineering **A**cce**L**eration
CO	Dissemination level “Confidential” (only for members of the consortium, including the JU)
DM	Design Model
FMEA	Failure Mode and Effect Analysis
FTA	Fault Tree Analysis
IDE	Integrated Development Environment
MB-SA	Model Based Safety Analysis
MB-RAMS	Model Based RAMS
MDE	Model Driven Engineering
MOMUT	Model Mutation based Test case generation
R	Deliverable Type “Report”
RAMS	Reliability, Availability, Maintainability and Safety
RM	Requirements Management
RODIN	Rigorous Open Development Environment for Complex Systems
SA	Safety Analysis
SAC	Safety Application Condition
SEE	System Engineering Environment
SM	Safety Model
SP	Subproject
TAS PLF	Abbreviation for “TAS Control Platform”
TCG	Test Case Generator
UC	Use Case
UML	Unified Modelling Language
WEFACT	Workflow Engine for Analysis, Certification and Testing
WP	Work Package

Table 7-1: Terms, Abbreviations and Definitions

8 References

[EN-50126]	European Standard EN 50126; *Railway Applications – The specification and demonstration of Reliability, Availability, Maintainability and Safety (RAMS)*; CENELEC September 1999
[EN-50128]	European Standard EN 50128; *Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems*; CENELEC June 2011
[IEC 60812]	International Electrotechnical Commission; *Analysis techniques for system reliability – Procedure for failure mode and effects analysis (FMEA)*, 2006
[IEC 61025]	International Electrotechnical Commission; *Fault tree analysis (FTA)*, 2006
[SCADE]	*Online documentation of Esterel SCADE Suite*; URL: http://www.esterel-technologies.com/products/scade-suite/
[Event-B, RODIN]	*Event-B and the Rodin Platform*; URL: http://www.event-b.org/
[Schoitsch, 2006]	Egbert Althammer, Henrik Eriksson, Jonny Vinter, László Gönczy, András Pataricza, György Csertán; *Validation and Certification of Safety-Critical Embedded Systems - The DECOS Test Bench* ; International Conference on Computer Safety, Reliability and Security - SAFECOMP, Proceedings, pp. 372-385, 2006
[Gerstinger et. al., 2008]	A. Gerstinger, H. Kantz and C. Scherrer; *TAS Control Platform: A Platform for Safety-Critical Railway Applications*; October 2008 (ERCIM News Nr. 75)
[Abrial, 2010]	J.-R.Abrial; *Modeling in Event-B – System and Software Engineering*; text book, published by Cambridge University Press
[Snook, 2008]	C. Snook and M. Butler; *UML-B and Event-B: an integration of languages and tools*; IASTED International Conference on Software Engineering – SE2008, Innsbruck
[Abrial, 2009]	J.-R. Abrial et. al.; Rodin: *An Open Toolset for Modelling and Reasoning in Event-B*; International Journal on Software Tools for Technology Transfer, 12
[Thomas, 2011]	F. Thomas, F. Belmonte; *Using Topcased and a Viewpoint-based Framework to describe Safety Concerns of Railway Signalling Systems*; Proceedings of the Topcased Days, Toulouse, France, February 2011