MOGENTES

Model-based Generation of Tests for Dependable Embedded Systems

Strategies for Tool Certification

Project	MOGENTES		**Contract Number**		216679
Document Id	8-10_1.1r_D8.5	**Date**	2011-05-04	**Deliverable**	D8.5
Contact Person	Rickard Svenningsson		**Organisation**		SP
Phone	+46 10 516 57 89		**E-Mail**		rickard.svenningsson@sp.se

Distribution Table

Name	Company	Department	No. of copies	Hardcopy/ Softcopy
ALL Mogentes Participants			1	SC

Change History

Version	Date	Reason for Change	Pages Affected
0.1	2010-11-30	Initial version	All
0.2	2010-12-20	Updated draft	All
0.3	2010-12-22	Updated draft	All
1.0	2011-03-28	Released version	All
1.1	2011-05-04	Correcting format issues	Title Page, References

Table 1-1: Revision History

Contents

Contents of Figures

Contents of Tables

1 Introduction

This report describes how the test generation tools that are developed within the MOGENTES [MOGENTES] project are to be treated when they are used as supporting tools for the development of a safety-related system. Even though the industrial partners in the project are from the automotive (on- and off-highway vehicles) and railway application domains, the tools that are developed within the project are not limited for use only within these domains. Other domains may include aerospace and medical applications. Therefore, as many of the domain-specific standards (e.g. ISO DIS 26262 [ISO26262] and the group EN50126/50128/50129 [EN50126][EN50128][EN50129]) are specializations of the generic safety standard IEC 61508 [IEC61508], the IEC 61508 standard is used as the foundation for the discussions in this report.

The domain specific standards ISO DIS 26262 and EN 50128 are investigated to take into account domain-specific requirements for the tools if these are to be used in the automotive and railway signalling domains, respectively.

Safety means that a system does not present unacceptable risks of health, environmental, or economical damage. Damage can arise either directly or indirectly as a result of damage to property or the environment. Functional safety is that part of overall safety that depends on whether a system or component operates correctly with the input signals that it receives. Functional safety must not be confused with electrical safety, which is concerned with protection against electric shock and fires caused by electricity.

Even though certification of tools is desirable for tools that shall support the development of safety related systems, development according to quality management standards (e.g. ISO 9000 [ISO9000] and ISO 9001 [ISO9001]) is always part of a sound approach to achieve a high quality tool and therefore assumed in this report. This report will therefore focus on the requirements of some common safety standards on tools that support the development of electronic, safety-related, systems.

This report is organized as follows: The tools developed within MOGENTES are described briefly in section 2. In section 3, requirements of the standards are discussed together with their implications on certification of the tools. Section 4 summarizes this report and provides some concluding remarks. This is followed by abbreviations and references. This report also has an appendix that describes the Prover iLock certification flow.

2 MOGENTES Tools

This section briefly describes the test case generation tool chains that have been developed or improved within the MOGENTES project. The purpose is to classify each tool chain according to IEC 61508 and the domain specific standards of the demonstrators that they are applied to. In the subsequent subsections, these classes are further investigated to find specific requirements. All the tool chains have in common that they are used for test case generation (TCG) from system models, but differs with respect to their designated model language and testing purpose. In addition to test case generation, the Prover iLock tool [iLock] also supports system design and software generation.

2.1 TCG from UML/OOAS

This tool chain aims at TCG from dedicated test models expressed in a subset of UML.
UML semantics is not directly accessible to formal approaches for two reasons:

- UML semantic is given mainly in natural language, which is not always unambiguous and complete
- UML features a set of so called semantic variation points, which explicitly identify areas where the semantics are intentionally underspecified to provide leeway for domain-specific refinements of the general UML semantics (e.g. by using stereotypes and profiles)

One option to deal with this in the context of testing would be to consider implementations compliant to any valid semantic interpretation as correct. However, the creator of the UML model has usually a semantic interpretation in mind which is much more stringent.
Therefore, we use the other option, assigning a concrete semantic interpretation and expressing it in a form which is unambiguous. The chosen formalism to do this is using Action Systems, because they offer a rather simple but powerful semantics, backed by the body of existing work (e.g. refinement theory, hybrid action systems, object-oriented action systems).
The TCG approach is mutation based, that is, it searches for traces that allow distinguishing between the action system from the original UML model and an action system from a modified UML model.
This test case generation tool chain is applied to automotive and railway demonstrators.

2.2 TCG from UML/UPPAAL

Key components of the approach are a UML profile that enables the modeller to introduce such extra information into the model that is necessary for simulation, model checking and test case synthesis and the UML to UPPAAL transformation itself implemented as a set of plug-ins for the Papyrus modelling environment.
The steps in test case generation with the UML/UPPAAL tool chain are as follows:

- The modeller constructs the UML model of the application.
- The UML model is automatically transformed to an intermediate formalism where the static structure is transformed to an easy to process form, AGSL code fragments are parsed and state machines are represented by Kripke transition systems. The intermediate formalism also holds the relevant parts of the static structure (packages, classes, etc.) thus the intermediate model can be seen as a self-contained pre-processed image of the input UML model. Although there is no need for manually tuning the intermediate model the tool chain also provides a tree editor for editing intermediate models. The transformation is implemented in various Papyrus plug-ins and hidden behind a simple user interface, i.e. a menu item in a pop-up menu.
- The intermediate model is automatically transformed to an UPPAAL model. This model can be simply loaded into UPPAAL without any further human intervention. This transformation is also implemented by Papyrus plug-ins and can be invoked through a menu.
- Test case generation is performed by
  i. choosing the goal the test case should demonstrate, e.g., “the test case should demonstrate that the alarm system is able to perform acoustic alarm”
  ii. specifying the logical negation of the test goal as a globally true formula e.g., “it is globally true for all possible execution traces that the alarm system never performs acoustic alarm” – obviously this statement is false
  iii. providing the corresponding formula to the model checker interface of UPPAAL and the counterexample returned by the tool will be a trace that is a test case for the requirement. In

order to facilitate the invocation of the command line version of the UPPAAL model checker (which can be used in batch mode with specific parameters), an easy to use Eclipse plug-in was implemented.

- The final step of the tool chain is the transformation of the UPPAAL trace to abstract test case format. Since the abstract test case must reference the elements of the UML model, a mapping between the UML and the UPPAAL models is necessary. This information is exported in the form of a mapping file, during the transformation from UML models to intermediate models. In order to generate abstract test cases, both the UPPAAL trace and the mapping file are parsed. An Eclipse plug-in is implemented, which executes the parsing and the test-case conversion.

This tool chain is used for the automotive and railway demonstrators.

2.3 TCG using Simulink/CMBC

The test case generation tool chain with Simulink/CBMC consists of the following basic steps:

- The Simulink models under test are automatically transformed into an intermediate representation amenable to static analysis. This translation process conclusively determines the semantics of the model.
- The test coverage and goals are formulated in terms of fault models, which are based on the syntactic and semantic modifications of the original Simulink models. In this way, we achieve a flexible and general framework that subsumes standard coverage criteria and is directly related to functional and non-functional requirements specifications.
- A model checker such as CBMC is applied to satisfy such coverage, which can generate counterexamples that demonstrate the reachability of certain statements or conditions. The use of symbolic model checking makes it possible to explore the behaviour of models with high precision, taking intricate details such as the actual floating-point semantics of execution platforms into account.

This tool chain is applied for the automotive demonstrators.

2.4 TCG using Simulink/MODIFI

The purpose of the fault injection tool chain is to:

- *Carry out early evaluation of model robustness against faults*. Different design choices can be benchmarked to find the most robust version for the specific set of faults considered. The result from the fault injection experiments is used to design error detection mechanisms (EDMs) and error recovery mechanisms (ERMs) which can be added to the model.
- *Exercise and evaluate added EDMs and ERMs in the model*. Results from experiments, with and without fault handling mechanisms, are compared to verify that the capability of the model to handle faults is improved. Considering software models, added EDMs and ERMs will be automatically generated inside the source code from the software model. Hence, the EDMs/ERMs are then automatically deployed to the actual system together with the rest of the software.
- *Create test cases for fault injection on the real system*. During fault injection at model level, effective errors leading to failures (safety requirement violations) are candidates for fault injection on the embedded system later on. To be able to e.g. reproduce faults injected on signals in the model a mapping is carried out between signal names in the model and physical addresses in a microcontroller memory where the signal (variable) value is stored. Test cases for physical fault injection are based on the XML output from the MODIFI which contains information about the golden (fault-free) run and all experiments.
- *Simulate effects of hardware faults already at model level to save time during system test*. Early design and evaluation of fault handling mechanisms, during the modelling phase, will have the potential of saving time during system test.

This tool chain is applied for the automotive demonstrators.

2.5 TCG using Prover iLock

The Prover iLock track has applied the following TCG methods:

1. TCG based on a Generic Test Specification defined in PiSPEC and instantiated for a specific application using configuration data. This is a standard Prover iLock TCG method.
2. TCG based on Minimal Cut Set (MCS) generation using (provable) safety requirements for a specific application and a set of faults, which are injected into the design of the specific application. This creates a set of test cases with active faults that lead to violation of safety requirements, and establishes also which faults cannot lead to violation of the system's safety requirements.
3. TCG based on structural coverage criteria of the specific application's design. This creates a set of test cases that demonstrate the functional behaviour of the design.

The first kind of TCG is based on the observation that it can be generically expressed what kind of functional testing should be performed for a *family* of systems. For example, a generic test case can define that if a certain sequence of input events takes place for a train route, then all signals along the train route shall display a proceed aspect. Given a specific application, one can generate all instances of this generic test case for the specific application. This kind of test generation is part of the Prover iLock core, thus not particularly developed for MOGENTES.
While the first kind of TCG is used for efficient validation of functional correctness, the second kind is used for efficient generation of test cases with a minimum number of faults that lead to the system's safety requirements being violated. This type of TCG presupposes the existence of safety requirements for the system, since minimal cut sets are identified based on whether they lead to a violation of a safety requirement. This TCG method takes an upper limit of the number of simultaneous faults as input parameter and generates a set of test cases that demonstrate all safety requirement violations due to the active faults. A typical use case is to generate a set of test cases that is guaranteed to find all violations that can occur if *at most one* fault is active. This TCG method was particularly developed for MOGENTES.
The third kind of TCG is intended to be used as a push-button check for a system's "liveness", by generating TCG objectives corresponding to a coverage measure on the specific application's design. For example, it could be checked that each finite state machine state in the design can be reached. Using this TCG method, it is possible to check that all signal lamps can be turned on and off, that all switches can be moved, and so on. Also this kind of test case generation was particularly developed for MOGENTES.

3 Tool Requirements According To Safety Standards

A safety-related system is a system that, when it malfunctions, can compromise safety. For functional safety of safety-related *E/E/PE* (Electrical/Electronic/Programmable Electronic) systems, the standard IEC 61508 is applicable. This is a generic standard that provides general requirements for E/E/PE safety-related systems where application sector standards do not exist. Domain-specific standards are often derived from IEC 61508, including but not limited to EN 50126/50128/50129 for railway signalling applications and ISO DIS 26262 for automotive applications. This section therefore starts with IEC 61508 after which differences between this and domain specific standards are described.
Common in the described standards in this report are the classification of safety integrity, so called safety integrity levels (SIL). A higher number or letter implicates higher safety requirements. IEC 61508 and EN 50129 define four safety integrity levels, SIL 1-4. ISO DIS 26262 also defines four levels of safety integrity ASIL A-D. The safety integrity levels are not possible to directly map between the standards.

3.1 Programmable Electronic Safety-Related Systems (IEC 61508)

Support tools are tools that support the development or are used to gain confidence in a safety-related system. Such tools include, but are not limited to, development and design tools, language translators, testing and debugging tools, and configuration management tools. Support tools are further classified according to their influence on the system. Software on-line support tools are tools that can directly influence the safety-related system during its run time (IEC 61508-4, clause 3.2.10). Off-line support tools are tools that support a phase of the software development lifecycle and that cannot directly influence the safety-related system during its run-time.
Examples of these classes are operating systems and code generators, respectively. Since the tools developed within the scope of MOGENTES aim at supporting testing of safety related systems, they are all in the second class i.e. off-line support tools.
Off-line support tools can be further classified into three different classes according to their contribution to the executable code (IEC 61508-4, clause 3.2.11).

T1: Generates no outputs which can directly or indirectly contribute to the executable code (including data) of the safety related system.
Examples of such tools include text editors and requirement tools that do not generate source code.
T2: Supports the test or verification of the design or executable code, where errors in the tool can fail to reveal defects but cannot directly create errors in the executable software.
Examples of such tools include test case generators and static analysis tools
T3: Generates outputs which can directly or indirectly contribute to the executable code of the safety related system.
Examples of such tools include compilers and code generators. Figure 1 shows the support tool classification with examples.

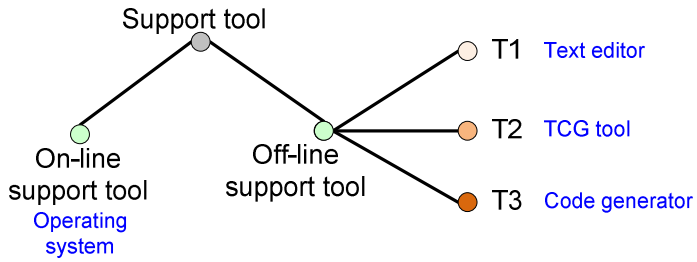

Figure 1 Support tool classification according to IEC 61508

Since all tools that are developed within MOGENTES, excluding Prover iLock, are test-case generators that do not directly affect the source code of the system they all belong to the T2 class of off-line support tools. Prover iLock however is used for code generation, formal verification and test case generation, thus belonging to the T3 class of off-line support tools.

Clause 7.4.4 in IEC 61508-3 describes some general requirements for off-line support tools. It states that off-line support tools shall be selected as a coherent part of the software development activities. This include that tools shall be selected to be integrated, e.g. by using the output of one tool as input to the next tool without human intervention in order to minimize the possibility of introducing human errors. The selection of such tools shall also be justified. In MOGENTES, this is supported by the tool integration framework.
The standard further requires off-line support tools in classes T2 and T3 to have a specification or product manual that defines the behaviour of the tool together with instructions and constraints on its use. It also requires such tools to be assessed with the aim to determine the level of reliance that shall be placed on the tool, and potential failure mechanisms that may affect the executable software (T3 only). Examples of how to achieve this is to perform software HAZOP, restrict tool functionality, check tool output and to use diverse tools for the same purpose.
For T3 class applications, evidence shall be available that the tool conforms to its specification or manual, so called tool validation (Clause 7.4.4.7 in IEC 61508-3). This may be based on a combination of history of successful use in similar environments and for similar applications (proven in use) and of tool validation.
Such tool validation shall cover:

- a chronological record of the validation activities
- the version of the tool product manual being used
- the tool functions being validated
- tools and equipment used
- the results of the validation activity; the documented results of validation shall state either that the software has passed the validation or the reasons for its failure
- test cases and their results for subsequent analysis
- discrepancies between expected and actual results

Every new version of a support tool shall be certified. This may rely on evidence provided for earlier versions if sufficient evidence provides that the modifications do not affect tool compatibility with the rest of the tools in the integrated tool chain and that the new version is unlikely to contain significant new, unknown faults.

For tools that are used for software module testing (such as those developed within MOGENTES) that include automatic test case generation, Annex C in IEC 61508-3 describes the maximum level of rigorousness that can be achieved for the assessment of software systematic capability properties. For model based testing techniques, high levels of rigorousness are reachable if test cases are generated (e.g. Table C.5 in IEC 61508-3).

Conrad et al. [Conrad] have summarized their experience from the ISO/DIS 26262 tool qualification support gained during the qualification of the Mathworks Real-Time Workshop Embedded Coder code generator and the PolySpace code verifier.

3.2 Safety-Related Electronic Systems in Road Vehicles (ISO DIS 26262)

In the automotive electronics safety standard ISO/DIS 26262, the usage of tools for software quality assessment is addressed more detailed than in IEC 61508.

Clause 11 of the 8th part of the standard describes the qualification requirements of software tools that shall be used within the development of a safety related system. A software tool is defined as a tool that is used in the development of a system or its software or hardware elements. General requirements on these tools are similar to those for IEC 61508 regarding e.g. documentation and version handling.

To use software tools, a certain confidence is required in that these tools shall achieve:

- the risk of systematic faults in the developed product due to malfunctions of the software tool leading to erroneous outputs is minimized
- the development process is adequate with respect to compliance with ISO 26262, if activities or tasks required by ISO 26262 rely on the correct functioning of the used software tool.

To determine the required level of confidence, two metrics are needed. The first one is the possibility that the malfunctioning tool and its corresponding output can introduce or fail to detect errors in a safety-related item or element being developed, called *TI* (Tool Impact). The second metric is the confidence in preventing or detecting such errors in the output of the tool, called *TD* (Tool error Detection).
Tool impact is selected such that if there is an argument that there is no possibility that errors can be introduced or that the tool fails to detect errors, **TI1** is selected, otherwise **TI2** is selected.

For tool error detection there is a three step scale. **TD1** is selected if there is a high degree of confidence that a malfunction and its corresponding erroneous output will be prevented or detected. **TD2** is selected if the corresponding degree of confidence is medium. Otherwise, **TD3** is selected.

To determine the required tool confidence level (TCL), the following matrix is used.

Table 3-1 Tool confidence level according to ISO DIS 26262

	TD1	**TD2**	**TD3**
TI1	TCL1	TCL1	TCL1
TI2	TCL1	TCL2	TCL3

Further, depending on the safety integrity level of the tool target (i.e. ASIL A-D, where higher level implies a more safety critical part) and the tool confidence level, a suitable set of qualification methods are chosen. A software tool classified as **TCL1** do not require any qualification measures.

The specific methods are described in ISO DIS 26262-8, clause 11.4.6. A general reflection is that the higher safety integrity level of the tool target, the more formal are the requirements. E.g. for **TCL3** or **TCL2** in combination with **ASIL D**, the tool is expected to be developed in compliance with a safety standard (or a suitable subset of such), e.g. IEC 61508 or ISO DIS 26262. A validation according to ISO 26262 is also required, which is described in ISO DIS 26262-8, clause 11.4.9.

3.3 Safety-Related Electronic Systems for Railway Signalling

The development of safety-related electronic systems for railway signalling is not controlled by a single standard, but by a group of three standards. These are EN 50126 (Railway applications – The specification and demonstration of reliability, availability, maintainability and safety (RAMS)), EN 50128 (Railway applications – communications, signalling and processing systems – software for railway control and protection systems) and EN 50129 (Railway applications – communication, signalling and processing systems – safety related electronic systems for signalling). This group of standards is based on IEC 61508 but with modification for the railway application domain.

For supporting tools, i.e. tools that support to development of software (e.g. test case generators), EN 50128 clause 10.4.8 states that automatic testing tools shall be used when applicable. Further, in section B.7 (which is part of the *informative*, i.e. not normative, annex B) it is stated that tools "*should be certified so that some level of confidence can be assumed regarding the correctness of the outputs*". Such certification is often carried out by an independent body against independently set criteria, e.g. national or international standards. A suitable standard for such a certification is e.g. IEC 61508.

3.3.1 Prover iLock Certification Flow

Prover has defined a strategy for the certification of safety-critical software which fulfils the requirements of EN 50128.

The EN 50128 standard states that "There is no known way to prove the absence of faults in reasonably complex safety-related software" and it is concluded that well-documented production processes is the only available method to guarantee safety. Prover has observed that the situation has changed drastically since the standard was written 10 years ago: formal verification is now applicable to very complicated systems. If the system implementation can be proved to be safe using a formal verification process that itself satisfies CENELEC requirements, it is not important anymore to place the same level of trust in the tools used for development of the system. In fact, if a tool would introduce a safety-related error, it would be discovered by the trusted formal verification process. Therefore, one can relax requirements on development tools, except for the additional tools that perform the final, trusted formal verification of the system. This approach is applied for Prover iLock: an independent formal verification process and tool chain which in itself satisfies the requirements of CENELEC SIL 4 is used for safety assessment of system implementations. The formal verification task is performed by Prover Certifier, a software product which has been developed according to CENELEC SIL 4 requirements. It has been used to produce safety evidence for interlocking systems (including Urban metro as well as ERTMS level 2) as well as for Communication-Based Train Control systems. In these processes, which have been approved as compliant with CENELEC SIL 4, Prover Certifier has the full responsibility for the safety argumentation, with no additional safety testing whatsoever.

A detailed description of the Prover iLock Certification Flow is found in appendix A.

3.4 Qualification of Tools in Other Application Domains

In the aerospace domain, software certification is regulated by the standard DO-178B [DO178B]. Although it is focused on the developed airborne software, it also puts requirements on supporting tools which automate activities of the software lifecycle process, i.e. the tools need qualification. The tools are divided into two classes depending on if they can introduce errors in the software, software development tools, or if they cannot introduce errors but fail to detect them, software verification tools. This classification is similar to the one of IEC 61508.

According to the standard, the qualification criteria for software development tools include:

a. The software development process for the tool should satisfy the same objectives as the software development process of airborne software.
b. The software level assigned to the tool should be the same as that for the airborne software it produces, unless a reduction in software level of the tool can be justified to the certification authority.
c. It should be demonstrated that the tool complies with its operational requirements, e.g. through a trial period.
d. Software development tools should be verified to check the correctness consistency, and completeness of the operational requirements and to verify the tool against those requirements.

The qualification criteria for software verification tools should be achieved by demonstration that the tool complier with its operational requirements under normal operational conditions.

Requirements on support tools for development of safety-critical software in the nuclear domain are presented in IEC 60880 [IEC60880]. Among other aspects, requirements on support tools were compared between IEC 61508 and IEC 60880 [Lahtinen], and the conclusion was that IEC 60880 covers all but T1 of IEC 61508 off-line tools. Also there are more information and analysis related to support tools in IEC 60880.

3.5 Examples of Commercially Available Certified/Qualified Support Tools

Table 3-2 below, briefly summarizes commercially available support tools that either are certified or provide certification/qualification support via qualification/certification kits/packages.

Table 3-2 Examples of commercially available certified/qualified support tools

Design tools	**IEC 61508**	**ISO/DIS 26262**	**EN 50128**	**DO178B**	**IEC 60880**
SCADE Suite KCG (code generator tool)	SIL 3 certification kit		SIL3/4 certification kit	Level A certification kit	Compliance kit
Mathworks Real-time Workshop Embedded Coder (code generator tool)	IEC certification kit	IEC certification kit		DO qualification kit	
dSPACE Target Link (code generator tool)	Certified workflow	Certified workflow	Certified workflow?		
Sauer Danfoss PLUS+1 GUIDE (code generator tool)	SIL2 certified				
Verification tools					
Mathworks Polyspace (static analysis code verifier)	IEC certification kit	IEC certification kit		Qualification kit	
LDRA tool suite (static and dynamic software analysis)				Qualification support pack	
IPL Cantata++ (software unit and integration testing tool)				Qualification pack	
PRQA QAC++/QAC (software source code analysis tool)				Qualification package	
Vector Software VectorCAST (software testing, code coverage)				Qualification package	
IBM Rational Test RealTime (software testing tool)				Qualification kit	
IBM Rational Logiscope (static code analysis tool)				Qualification kit	
Klocwork Insight (static source code analysis tool)				Qualification package	

Certeron CertTEST (test case generation tool)				Qualification kit	
Bullseye Testing Technology BullseyeCoverage (code coverage tool)				Qualification kit	
Parasoft C++test (static analysis and unit testing tool)	Certified as T2 off-line support tool				
Hitex Tessy (software unit testing tool)				Qualification package	
AbsInt aiT (WCET analysis tool)				Qualification support kit	
AbsInt StackAnalyzer (stack usage analysis tool)				Qualification support kit	
Metrowerks CodeTEST (software analysis tool)				Qualification package	

The tools that have qualification kits for e.g. DO-178B are not certified or qualified as such, but the kits contain artifacts which are necessary for tool qualification. However, the qualification depends on the use and application of the tool in a specific project. Typically, the qualification kits for the verification support tools contain: tool requirements and specification, qualification procedures, and test cases for tool validation.

As can be seen from the table, there are much more certification support for DO-178B than for the other standards. The reason is that, since its introduction several years ago, DO-178B has had explicit tool qualification requirements, whereas such have been introduced in the recent new version of IEC 61508 and in ISO DIS 26262. Since tool support is becoming more and more popular, certified tools and certification/qualification support kits will be commonplace in the future.

4 Conclusion

This report describes how three different safety standards treat tools that supports the development of safety related electronic systems (e.g. the test case generators developed within the MOGENTES project). The standards that are covered are the generic IEC 61508, the automotive application standard ISO DIS 26262 and the railway standards EN 50126/50128/50129.

The IEC 61508 and ISO DIS 26262 describe quite specifically how support tools shall be treated with respect to quality assurance, while the railway standards only suggest that such tools should be certified.

For the MOGENTES tools to be used as supporting tools for development of safety related systems according to IEC 61508, it is required that sufficient documentation is available. This shall include description of the tool behavior, instructions and constraints on its use. Further, the tool shall be assessed to determine the level of reliance that can be put on the tool, e.g. by performing a HAZOP.

For usage within the development of safety-related systems with high SIL levels (ASIL D) according to ISO 26262, it is also required that the tools are developed according to a suitable standard.

5 The Abbreviations and Definitions

DIS	Draft International Standard
E/E/PE	Electrical/Electronic/Programmable Electronic
EDM	Error Detection Mechanism
EN	European standard
ERM	Error Recovery Mechanism
ISO	International Organization for Standardization
MOGENTES	Model-based Generation of Tests for Dependable Embedded Systems
TCG	Test Case Generation/Test Case Generator
UML	Unified Modelling Language
XML	Extensible Markup Language

6 References

[Conrad] M. Conrad, P. Munier, and F. Rauch, "Qualifying Software Tools According to ISO 26262", http://www.mathworks.com

[DO178B] *Software considerations in airborne systems and equipment certification*, RCTA/DO-178B, RCTA

[EN50126] *Railway Applications - The specification and demonstration of Reliability, Availability, Maintainability and Safety (RAMS)*, EN 50126:1999. European Committee for Electrotechnical Standardization (CENELEC)

[EN50128] *Railways Applications – Communications, signalling and processing systems – Software for railway control and protection systems,* EN 50128: 2001. European Committee for Electrotechnical Standardization (CENELEC).

[EN50129] *Railways Applications – Communications, signalling and processing systems – Safety related electronic systems for signalling,* EN 50129: 2003. European Committee for Electrotechnical Standardization (CENELEC).

[IEC 60880] *Nuclear power plants - Instrumentation and control systems important to safety - Software aspects for computer-based systems performing category A functions,* IEC 60880 edition 2.0:2006, International Electrotechnical Commission (IEC).

[IEC61508] *Functional safety of electrical/electronic/programmable electronic safety-related systems*. IEC 61508:2010. International Electrotechnical Commission (IEC).

[iLock] *Prover iLock*. Prover, http://www.prover.com

[ILOCKP] The Prover iLock Process. 2010. http://www.prover.com/company/downloads/.

[ISO26262] *Road vehicles – Functional Safety*, ISO DIS 26262:2010. The International Organization for Standardization (ISO).

[ISO9000] *Quality management systems,* ISO 9000:2005. The International Organization for Standardization (ISO).

[ISO9001] *Quality management systems – requirements,* ISO 9001:2008. The International Organization for Standardization (ISO).

[Lahtinen] J. Lahtinen, M. Johansson, J. Ranta, H. Harju, and R. Nevalainen, "Comparison between IEC 60880 and IEC 61508 for Certification Purposes in the Nuclear Domain", in Proceedings of SAFECOMP 2010, 2010.

[MOGENTES] *Model-based Generation of Tests for Dependable Embedded Systems (MOGENTES)*, http://www.mogentes.eu

7 Appendix A: Prover iLock Certification Flow

Prover iLock is an integrated toolset for automated generation, simulation, and formal verification of railway interlocking systems. Once configured for a generic family of interlocking systems, the Prover iLock Development Flow is used for creating specific applications. The Prover iLock Certification Flow [ILOCKP] is provided to certify that such a process satisfies the highest applicable safety standards.

The purpose of this document/section is to give an overview of the Prover iLock Certification Flow, as used for tools qualification. Many technical details have been omitted in order to simplify the presentation

Section 7.1 gives an overview of the Prover iLock Development Flow. Section 7.2 introduces the concept of Formal Verification-Based Certification and shows how a formal verification flow can be certified. In Section 7.3, we show how this concept is applied in the Prover iLock Certification Flow.

7.1 Prover iLock Development Flow

The Prover iLock Development Flow creates Specific Applications (railway signalling systems) based on a Generic Application and the Prover iLock software suite. The Generic Application is defined using three formal specifications in the language of PiSPEC. The individual formal specifications are processed by individual Prover iLock software modules, in order to perform code generation, functional simulation and formal safety verification of the application (see Figure 2). Test cases generated can be exported for external use, such as hardware-in-the-loop testing.

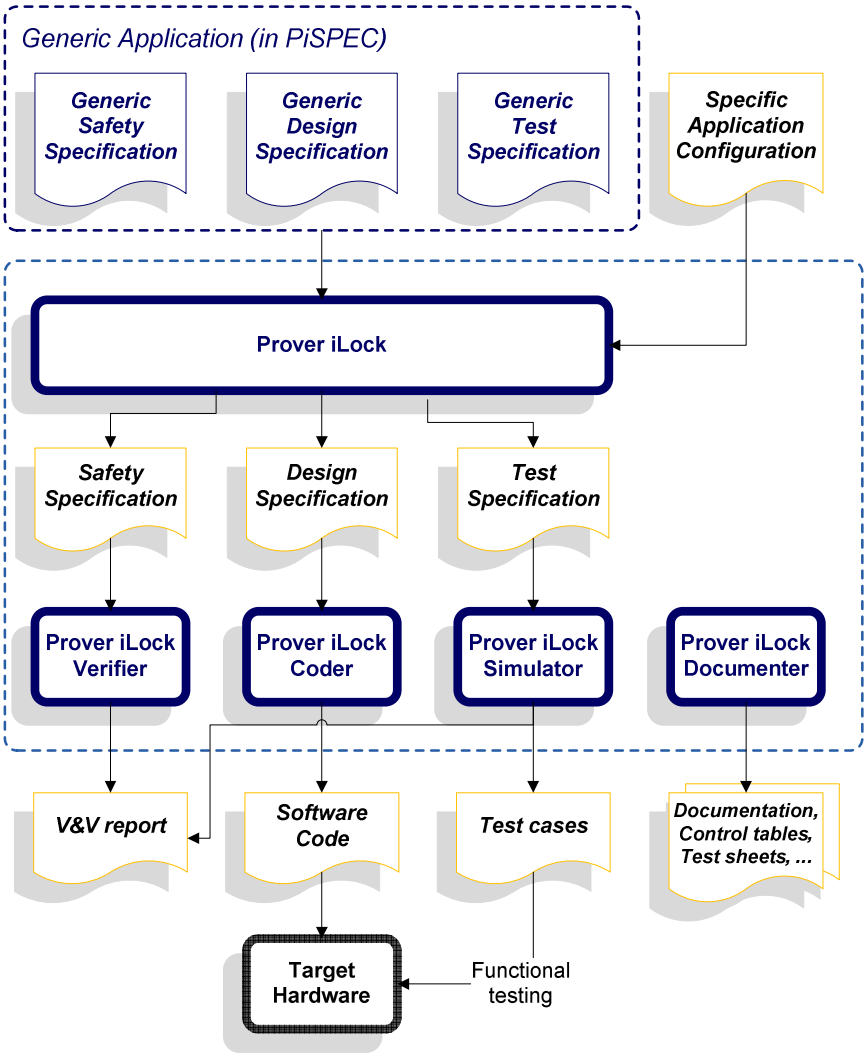

Figure 2 Prover iLock Development Flow

7.2 Verification and Certification

Railway interlocking software is safety-critical. Therefore, its production is surrounded by rules and regulations whose purpose is to ensure sufficiently (acceptable) low risks for residual safety issues. The concept of Safety Integrity Level (SIL) of the CENELEC standards is a measure of the level of safety required by a safety critical system. Depending on the SIL value, the CENELEC standards impose requirements on the development process that are to be fulfilled in order to certify that the software has been correctly produced.

This standard's approach for certification focuses on ensuring that the code is correctly produced from a functional specification that has been shown to be safe. The Formal Verification-Based Certification approach focuses on verifying that the produced code is safe and imposes at least the same CENELEC requirements on the software which performs this verification. In other terms, the certification task is delegated to a verification process which has at least the same SIL classification as the code.

7.2.1 The Verification Flow

In a typical formal verification flow, as depicted in Figure 3, the system's software code is fed to a program which translates it to a formal language which in turn is fed to a proof engine together with a set of formalized safety properties. Finally the proof engine checks whether the system satisfies the safety properties.

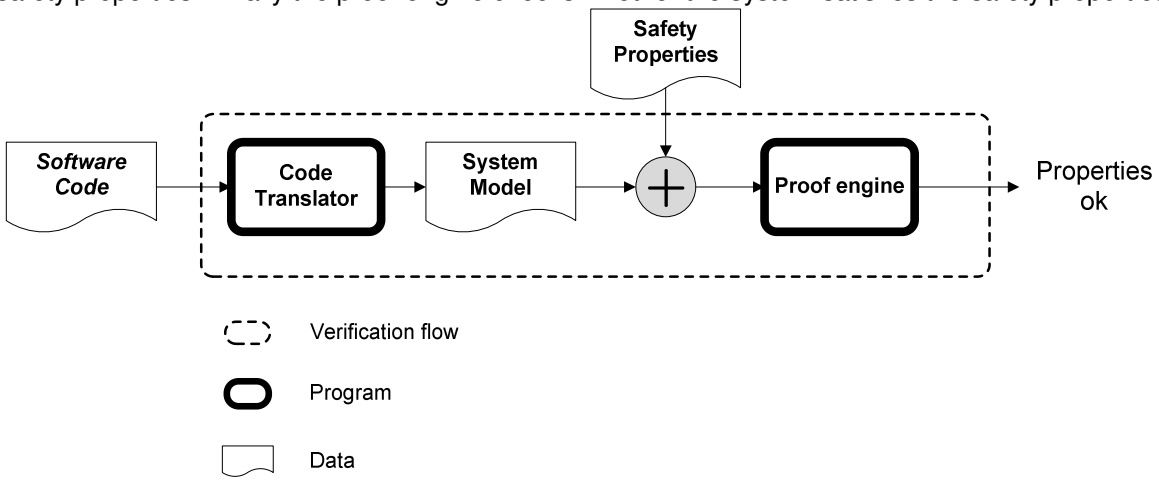

Figure 3 Non-Certified Verification Flow

In order to achieve a high level of confidence in such a flow, one has to guarantee that the safety properties, the code translator and the proof engine have at least the required level of confidence. If SIL 4 is the safety integrity level required, the resulting certifiable verification flow would be as depicted in Figure 4.

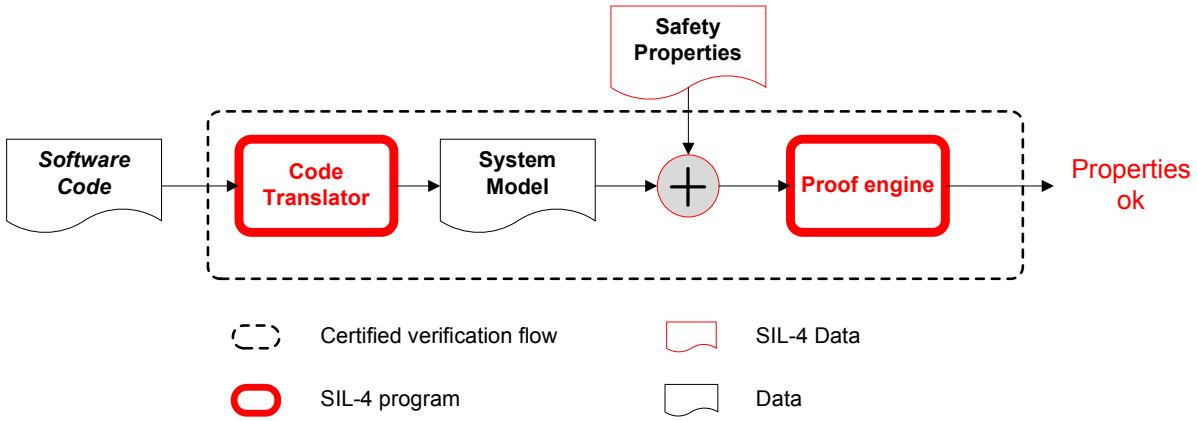

Figure 4 Certified Verification Flow

7.2.2 Prover Certifier

Prover Certifier contains two parts. The first is the proof engine. Its task is to search automatically for a formal correctness proof and record it in a proof log. Large parts of the proof engine concerns optimization

for speed and capacity, aspects that are important indeed for the usefulness of Prover Certifier, but irrelevant for the safety argument: the only thing that matters is that the proof found was correct. That this is the case can be ensured by inspection of the proof log, a task that is carried out by the second part of Prover Certifier: the proof checker. The proof checker has been developed in a CENELEC SIL 4-compliant process. If its judgment is that a correct proof log has been produced, it means that the system to be verified has been proved to be safe, and that this fact is reliable to the level of CENELEC SIL 4.

In Figure 5, we give a description of the architecture of Prover Certifier. The input language to Prover Certifier is Prover's High Level Language (HLL), an expressive formal language well suited as target language for certifiable translators from specification, design and code languages. The HLL input file, which contains the formalization of the safety properties and the system, is translated by a SIL 4 component to a logically equivalent representation in Prover's Low Level Language (LLL), which has been developed especially for proof engines. Prover Certifier's proof engine now checks whether the system satisfies the safety properties and, in that case, produces a formal proof and a proof log. Finally, the proof checker inspects the proof log and delivers its judgment.

To ensure the highest possible reliability, the translation from HLL to LLL has been implemented using a diversification approach: two independent teams have developed translators in two different programming languages, both applying SIL 2-compliant processes. The equivalence of the results is checked by a component that has been developed in a SIL 4 process.

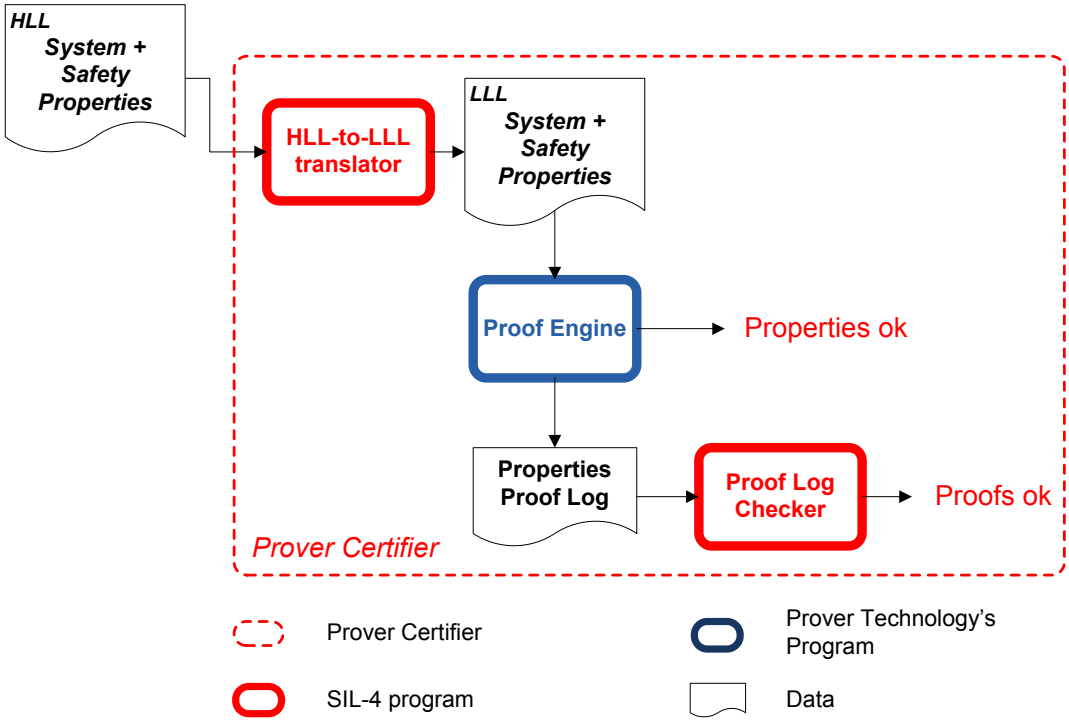

Figure 5 Prover Certifier Overview

7.3 Application to Prover iLock

Prover iLock is used for automatic generation of interlocking systems based on a Generic Application and a Specific Application configuration. The Generic Application is defined in the Prover iLock Specification Language PiSPEC. PiSPEC is a formal language, based on predicate logic and augmented with object oriented concepts, tailored for implementing generic design principles and requirements.

The Prover iLock Development Flow (cf. Figure 2) is used as depicted by the left-hand side of Figure 6. Based on a Specific Application configuration (including the track layout, the train routes, object configurations, etc) and the Generic Design Specification, Prover iLock applies the generic design for the given track layout to generate the specific application, which can in turn be converted into software code by Prover iLock Coder.

The right-hand side of the figure depicts all the components needed for achieving a certified flow. The inputs to the workflow consist of:

- the generic safety requirement specification (in PiSPEC),
- the track layout description (in XML format), and
- the software code for the system

Recall that Prover Certifier takes as input an HLL file describing the system and the safety properties (Figure 5). The goal of all the translators in Figure 6 is to build this file correctly based on their inputs. The (specific) safety properties are the combination of the generic safety requirements and the layout description. We ensure correctness of these translations by imposing SIL-4 classification on all the translators.

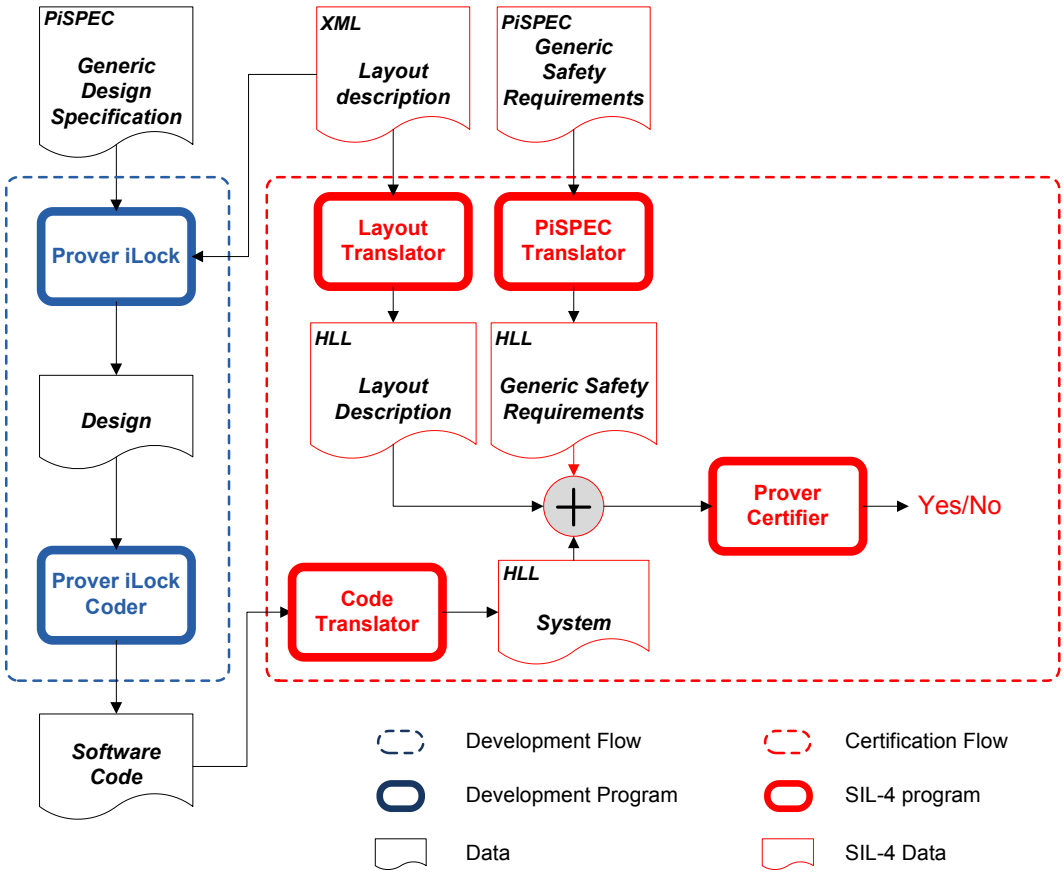

Figure 6 Prover iLock Certification Flow