Computer-aided Formal Specification to enable a fully automated Requirements-based Testing Process

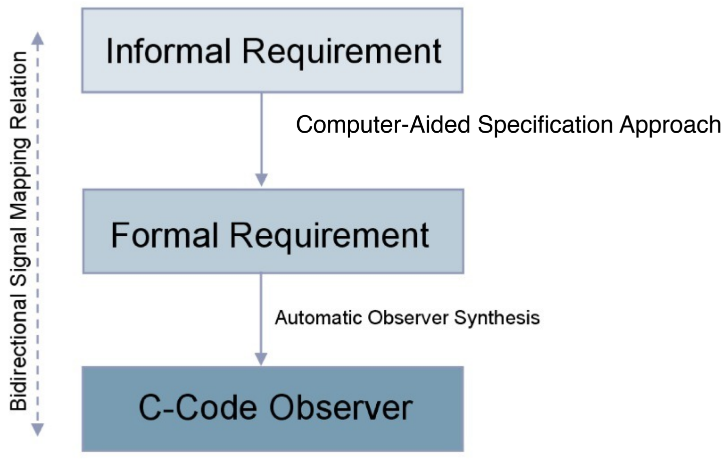
Hans J. Holberg
SVP Marketing & Sales, BTC Embedded Systems AG
An der Schmiede 4, 26135 Oldenburg, Germany
hans.j.holberg@btc-es.de

Dr. Udo Brockmeyer
CEO, BTC Embedded Systems AG
An der Schmiede 4, 26135 Oldenburg, Germany
udo.brockmeyer@btc-es.de

Abstract: The requirement specification stands at the beginning of any development process. It builds the basis for the further development of embedded control devices. Their quality is of great importance for the progress and success of each project. Especially in the area of safety critical development and the fulfillment of industrial standards like IEC61508 and ISO26262 in respect to functional safety, Formal requirement specification plays a major role. These are not very popular in the area of embedded software development, in comparison to model-based specification. Declarative methods, like formal requirement specifications, are considered to be too mathematical and too difficult to learn in general; it is observed as a method only for experts. Operational methods, like modeling, are widely accepted as typical activities of engineers. Formal specifications have various advantages in supporting a computer-aided fully automated requirement-based test process, due to its machine-readability, unambiguousness and traceability capabilities. This method presented here allows engineers, an intuitive and constructive specification of formal requirements, without having in-depth expertise in theory of Formal Methods. Starting point of the formalization process is the informal textual requirement, which is structured step-by-step by the user, in order to get a more and more unambiguous, machine-readable description. In the beginning, no concrete architecture, model, code or even implementation is mandatory. At later stages of the development process, the concrete architectures can be bound by the users to the former developed “virtual” formal requirement specification. This binding process always is accompanied by the wizard mechanism and allows a convenient and efficient handling. This automatically creates a linkage between the various artifacts of the development process. The specification environment enables automatic structuring and management of the corresponding links, which allows complete traceability throughout the entire development process. An important part of the described specification environment is the synthesis unit which automatically translates the formal specification into so called C-Observers. These C-Code-Functions are representing the particular requirements, which are executable as monitors/observers during the verification- and test process in the specific test environment. This allows a complete automated, requirement-based test approach, both at model level as well as at code and implementation levels. Outcome of many pilot projects in the automotive industry is the observation, that the necessary additional formalization effort can be compensated quickly by the highly automated test process. Additionally, the increased safety and quality level due to the higher precision and completeness of the requirement representation as well as a maximum level of reusability of this seamless methodology shows the advantage against conventional methods.

Field of Application

Modeling on the one hand has been established to improve the development process of embedded software. On the other hand, the formalization of requirements using formal methods still is a field for experts, while modeling is widely used by typical software development engineers. What is the reason for that? Modeling in an operational way seems to be very intuitive, especially if the models can be simulated immediately. Such executable specification directly shows the behavior of all combined target requirements, while any kind of formal specification of requirements in a declarative way ends-up with a list of isolated non-executable syntax constructs.
However, the quality of the requirement representation process plays an important role to ensure high quality of the final product of the development process.For the development of safety critical systems, clear, and unambiguous requirement specifications are essential.
Also, to enable automatic requirements-based testing, a syntactical und semantical sound specification, which is finally machine-readable, is necessary. Therefore the challenge in practice is to enable typical engineers to write those formal specifications in a safe and intuitive way.

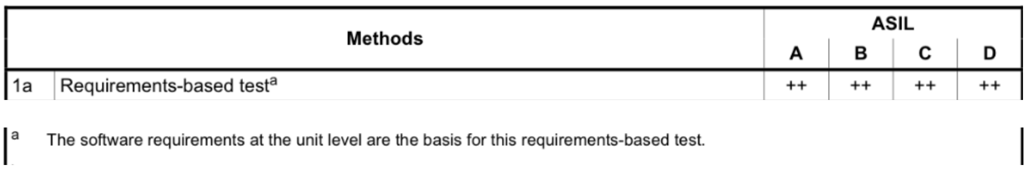

Informal Requirement
Computer-Aided Specification Approach
Formal Requirement
Automatic Observer Synthesis
C-Code Observer
Bidirectional Signal Mapping Relation

The presented technology[1] shall address this challenge by introducing a computer-aided process accompanied requirement specification method, which can be used from the very beginning of the development process. It starts with text written in natural language and it ends up with a machine readable C-Code-Observer description, which is used for automatic test and verification purposes.

[1] The presented approach has been realized in a first version as a requirement specification environment called BTC EmbeddedSpecifier. It comes as a model and code independent tool environment to enable formal specification in general, but is also smoothly integrated in the existing BTC EmbeddedTester automatic testing tool-chain especially made for dSPACE TargetLink.

Quality Aspects

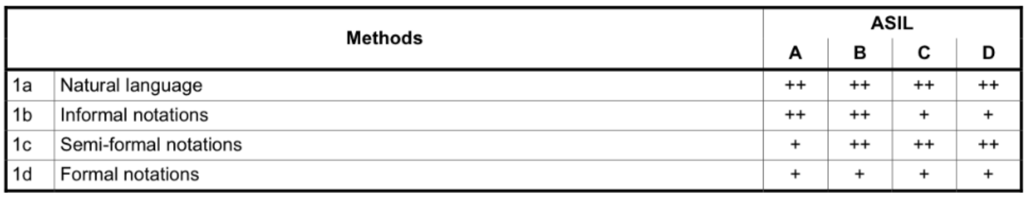
As the described method is currently mainly used in the automotive domain, quality aspects can be assessed by using relevant safety standards. Here it is of interest to have a look at the ISO 26262 standard [ISO], which is an adaptation and extension of the currently functional safety standard IEC 61508 especially for functional safety in automotive. In contrast to the IEC 61508 the ISO 26262 is taking the model-based development process into account. ISO 26262 is defining 4 different levels of safety, so called Automotive Safety Integrity Levels (ASIL A, ASIL B, ASIL C and ASIL D). Level A is the lowest and D the highest safety level.
Requirement specification plays a central role as basis for any verification process.



Methods		ASIL			
		A	B	C	D
1a	Requirements-based test[a]	++	++	++	++

[a] The software requirements at the unit level are the basis for this requirements-based test.

ISO 26262 highly recommends the **requirements-based testing** for all mentioned ASILs, but the higher the ASIL is taken into account, the more formal notations have to be addressed:

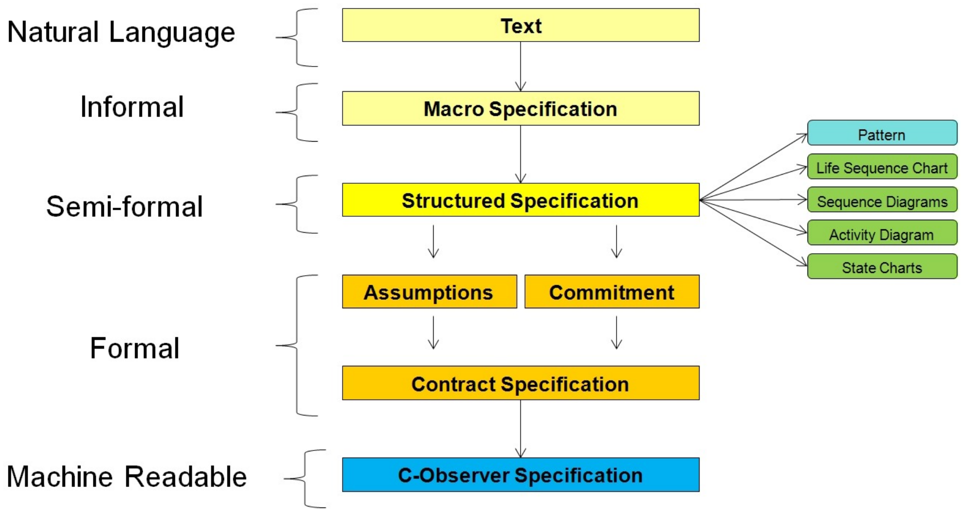


Methods		ASIL			
		A	B	C	D
1a	Natural language	++	++	++	++
1b	Informal notations	++	++	+	+
1c	Semi-formal notations	+	++	++	++
1d	Formal notations	+	+	+	+

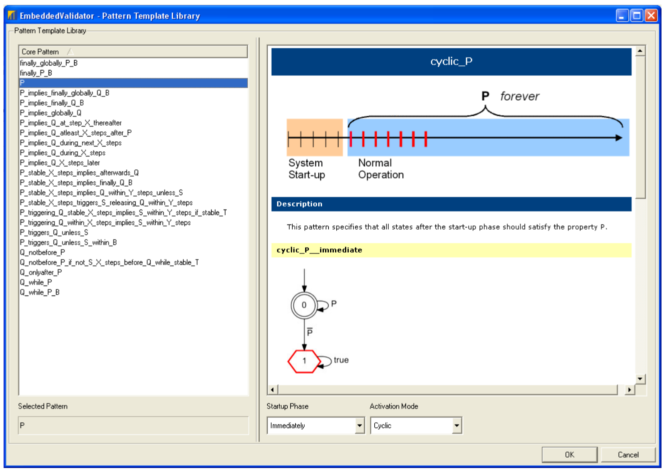
ISO 26262 highly recommends (double plus) for all ASILs a specification of the software design in "**Natural Language**" as a minimum. Additionally as a minimum level of specification, any kind of "**Informal Notation**" shall be used. For an "informal Notation" a well defined syntax is not necessarily mandatory, but it has to be written down. A notation can be called "**Semiformal**" , if a syntax is defined clearly, but the behavior interpretation can not be determined unambiguously.
"**Formal Notation**" requires a well defined syntax and semantics. This leads to a specification, which can be translated into a "**machine-readable**" format, which can be used for any further automatic analysis technology. These levels of notations are used in the presented specification method to allow to introduce a flexible step-by-step refinement methodology along the development process stages. Another very important aspect taken from the ISO 26262 is the highly recommended main principle: **Traceability** along the process.
All work products, which are developed and which are related to each-other shall be traceable linked to each other. This main principle is taken as a general foundation for the here presented specification method.

The Specification Method

The typical workflow of the presented requirements specification method is shown in the following figure. It describes a step-by-step formalization process, while the user specification becomes more and more structured and more concrete. It starts



with an informal requirement specification given in a natural language, and is written-down in an informal way as a **text**ual construct. In a next design step, the user identifies relevant objects in the given text, which will be identified as "**Macro Specifications**". These macros are used to identify any kind of events, conditions and timing information of the requirement under specification. In order to continue to structure the given textual requirement, the user can select the proper method of "**Structured Specification**". The proper method directly depends on the nature of the given requirement. Practice has shown, that not only one specification method can be selected to cover the specification needs of different application classes. For a huge set of automotive applications, like body electronics, power train, motor control, transmission and chassis, predefined requirement structures can be used to cover most of the mission and safety critical requirements. In the presented approach, so called Pattern Specifications [PATTERN] are used.



Core Pattern
finally_globally_P_B
finally_P_B
P
P_implies_finally_globally_Q_B
P_implies_finally_Q_B
P_implies_globally_Q
P_implies_Q_at_step_X_thereafter
P_implies_Q_atleast_X_steps_after_P
P_implies_Q_during_next_X_steps
P_implies_Q_during_X_steps
P_implies_Q_X_steps_later
P_stable_X_steps_implies_afterwards_Q
P_stable_X_steps_implies_finally_Q_B
P_stable_X_steps_implies_Q_within_Y_steps_unless_S
P_stable_X_steps_triggers_S_releasing_Q_within_Y_steps
P_triggering_Q_stable_X_steps_implies_S_within_Y_steps_if_stable_T
P_triggering_Q_within_X_steps_implies_S_within_Y_steps
P_triggers_Q_unless_S
P_triggers_Q_unless_S_within_B
Q_notbefore_P
Q_notbefore_P_if_not_S_X_steps_before_Q_while_stable_T
Q_onlyafter_P
Q_while_P
Q_while_P_B

Selected Pattern: P

cyclic_P

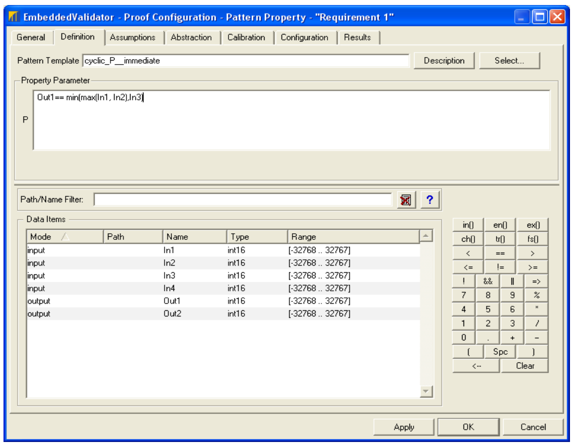
Description

This pattern specifies that all states after the start-up phase should satisfy the property P.

cyclic_P__immediate

Startup Phase: Immediately
Activation Mode: Cyclic

This approach provides a library of predefined patterns for specifying functional and safety critical requirements. Patterns can be instantiated simply by filling the pattern parameters with Boolean expressions ranging over architecture (e.g. model or code) elements/variables. Along the typical workflow, instead of directly



Pattern Template: cyclic_P__immediate

Property Parameter

P: Out1== min(max(In1 , In2),In3)

Path/Name Filter:

Data Items

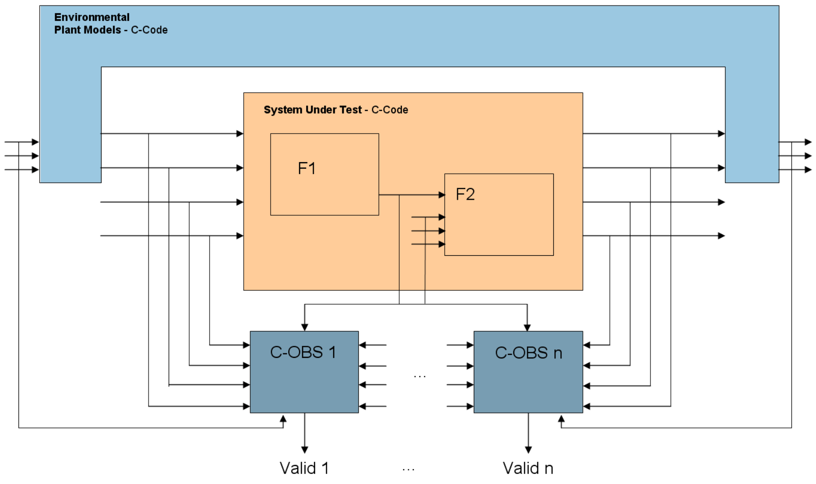
Mode	Path	Name	Type	Range
input		In1	int16	[-32768 .. 32767]
input		In2	int16	[-32768 .. 32767]
input		In3	int16	[-32768 .. 32767]
input		In4	int16	[-32768 .. 32767]
output		Out1	int16	[-32768 .. 32767]
output		Out2	int16	[-32768 .. 32767]

addressing architecture / interface objects, the user can use the former defined Macros in order to fill the pattern parameters with a semi-formal specification as an intermediate specification step. Later on in the specification process, the user can bind existing architecture/interface objects like model elements or code variables to the macros to finalize the formal specification. During this specification step, the complete traceability between the original text and the architecture / interface element is guaranteed via the macro specification.
The pattern specification method guarantees an easy user entry in the formal world, without having a deep mathematical and theoretical background. This schematic pattern approach allows full certainty about what has been formally specified, without any final doubt. In the future, the set of Structured Specifications will be extended in order to cover an extended list of application classes. Especially event driven requirements, which come in long sequences, need another specification approach. Here Sequence Diagrams (SDs) or Life Sequence Charts (LSCs) have been used successfully in several research projects in the past [LSC].

This described method is following the so called assumption/commitment style, which means, that not only the pure desired behavior (commitment) of a system under verification/test is taken into account, but also the specific environmental behavior (assumptions) can be specified by using the described structured specification elements, like pattern and/or SDs/LSCs. One commitment specification comes with as many assumption specifications as needed. The following formula shall describe the mathematical relationship between the set of Assumptions (A_1 … A_n) and the commitment C, which shall be granted by the system under verification/test:

$$\mathbf{A_1 \text{ and } A_2 \text{ and } \ldots \text{ and } A_n \Rightarrow C}$$

These connections between several assumption specifications and one commitment specification will be joined together within "**Contract Specifications**". These contracts finally represents a complete "**Formal Specification**" of a requirement, if all macros have been bound to existing architecture/interface objects of the system under verification/test. In a final fully automatic step, **"C-Code-Observers"** can be generated from those contract specifications in order be used for automatic verification and testing approaches. Therefore, this specification layer is called "**Machine Readable**". The next figure shows automatically generated commitment parts of C-Observers, which represents the specified requirements, automatically connected to the system under verification/test for an automatic testing approach [BTCEW2011] /



[BTCEW2010]. The assumptions of the contracts will be used to automatically synthesize the Environmental part of the verification/test architecture which is visualized in the figure above.

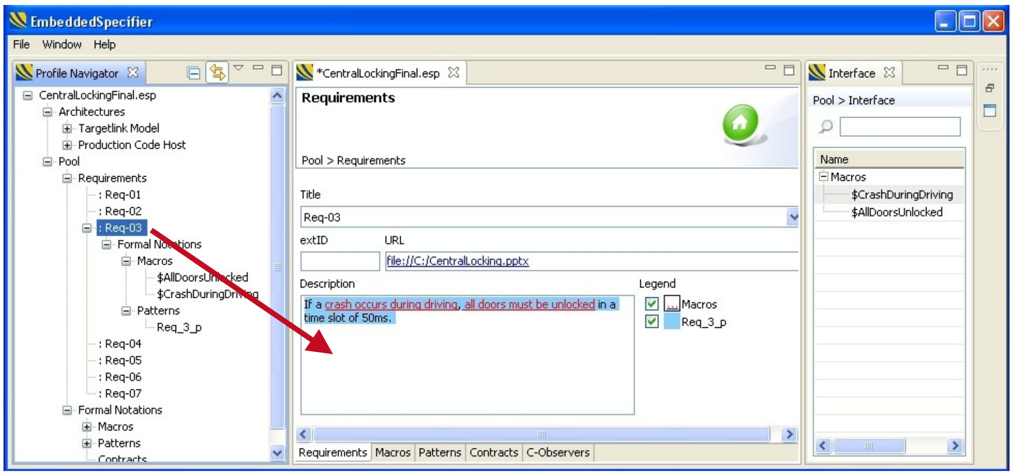
The specification elements in general can have three different modes:

- Informal
- Semi-formal
- Formal

Informal elements are generally linked to informal textual specifications. Semi-formal elements are filled with symbolical references like macros. Formal elements have to have a link to existing architecture/interface objects of an executable design (e.g. functional or implementation model) or implementation level (e.g. C-Code, Object Code, Hardware/Software integration etc.).
The specification method does not force the user to start the requirements specification process at the very beginning of the shown workflow, like a bottom-up approach. It give the full flexibility to enter this process at any point, e.g. starting top-down with Pattern specification while an architecture binding has been done immediately.

Computer-Aided User Interaction

The described specification method is realized in a tool-suite which supports the user during every specification stage. It automatically links the specification elements together, in order to guarantee full traceability and support automatic re-factoring activities. These two concepts allow firstly readability and secondly an efficient workflow, with a maximum of automatism and reusability.
The following figure shows an extract of the tool-suite which allows the user to work directly with an imported informal text specification to define Macros and Pattern Specifications.

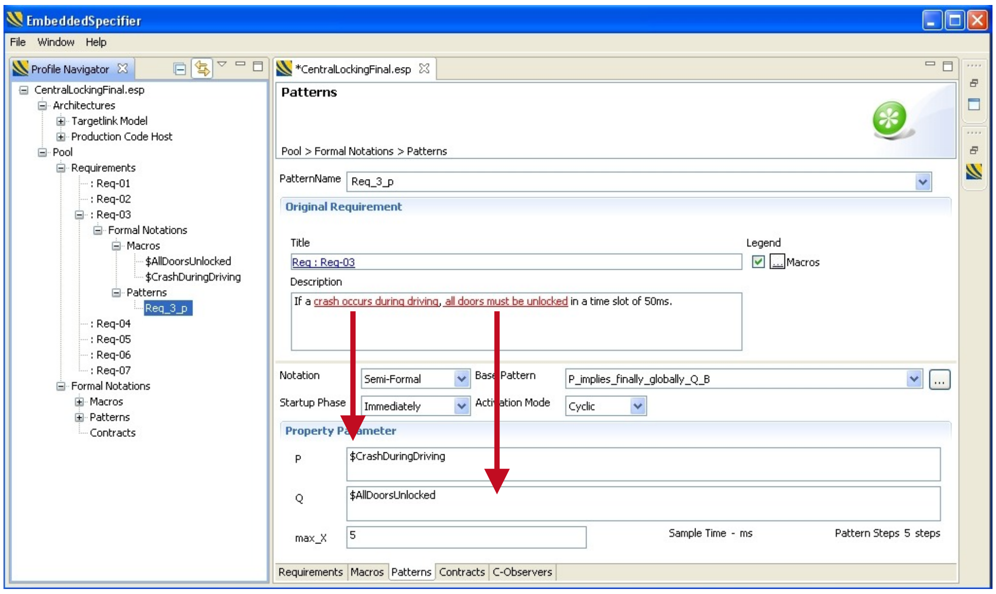


Title: Req-03

URL: file://C:/CentralLocking.pptx

Description: If a crash occurs during driving, all doors must be unlocked in a time slot of 50ms.

The text part, which represents a specification unit, simply can be selected by the user in order to create new macros and structured specification objects like pattern. The different specification objects are visualized as colorized text. Additionally, hyperlinks are added to the requirement view to allow fast specification access. This can be seen in the figure above on the "**Requirements**" dialog tab. All specification objects are hierarchically managed an accessible via "**Profile Navigator**"' as can be seen in the figure above on the left-hand side.
The next step of specification is the selection of the "Structured Specification"; this also can be done by selecting the corresponding text phrase and by creating e.g. a new pattern specification object. Following the created pattern link, the user can specify a semi-formal pattern requirement object while referencing user defined macros in the dialog tab "**Patterns**". Therefore a corresponding Pattern Template is selected out of the Pattern Library. This selection is done by the user based on the specific structure of the textual requirement. This process is accompanied by a tool-wizard to ensure an intuitive easy formalization workflow. After selecting the appropriate Pattern Template, the tool-suite shows the parameters (place holders) of the pattern in a separated dialog tab. Here, the user can define the parameters (P, Q etc.) simply by referencing Macros or Architecture/Interface objects, either for a semi-formal or for a final formal specification. The following figure shows the "Patterns" dialog tab.



Patterns

Pool > Formal Notations > Patterns

PatternName: Req_3_p

Original Requirement

Title: Req : Req-03

Legend: ✓ Macros

Description: If a crash occurs during driving, all doors must be unlocked in a time slot of 50ms.

Notation: Semi-Formal

Base Pattern: P_implies_finally_globally_Q_B

Startup Phase: Immediately

Activation Mode: Cyclic

Property Parameter

P: $CrashDuringDriving

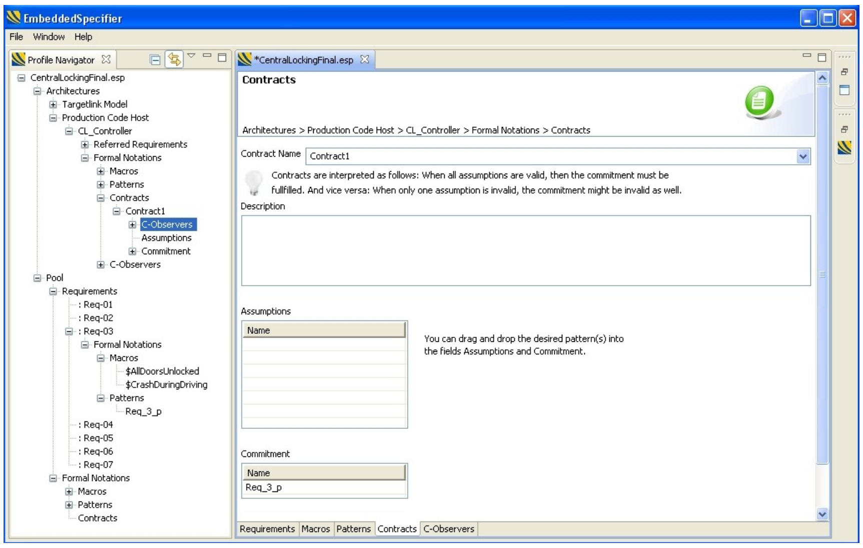
Q: $AllDoorsUnlocked

max_X: 5

Sample Time - ms

Pattern Steps 5 steps

Within the Patterns dialog, the user can select the desired level of formalization under "**Notation**". If any concretization is missing to reach the selected level of formalization, the tool-suite automatically identifies and visualizes the missing details to help the user. The user uses these specification method to formalize either commitments or assumptions. In the next interaction step if the formalization workflow, contract can be specified. The identification of the commitment and the set of assumptions can be done by simple drag-and-drop



Contracts

Architectures > Production Code Host > CL_Controller > Formal Notations > Contracts

Contract Name: Contract1

Contracts are interpreted as follows: When all assumptions are valid, then the commitment must be fulfilled. And vice versa: When only one assumption is invalid, the commitment might be invalid as well.

Description

Assumptions

Name

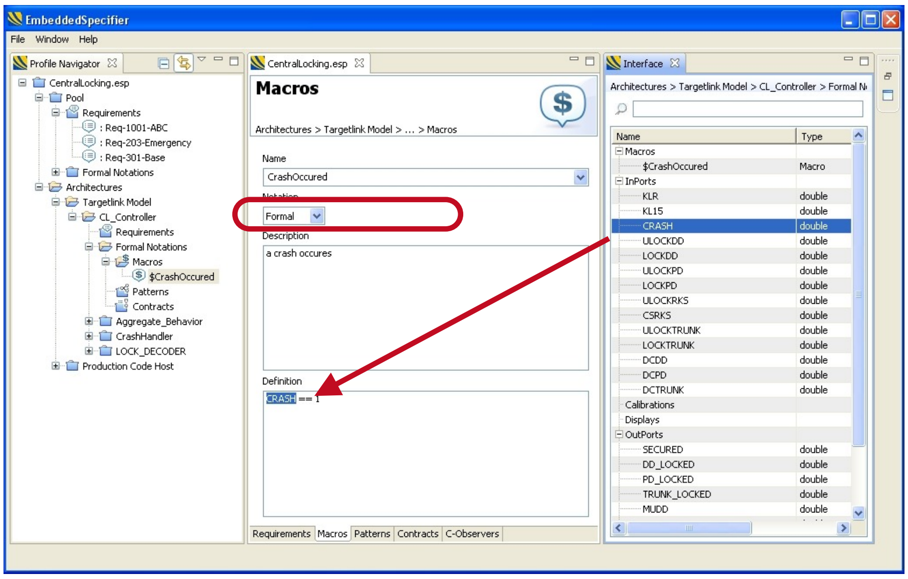
You can drag and drop the desired pattern(s) into the fields Assumptions and Commitment.

Commitment

Name
Req_3_p

activities in the Profile Navigator within the specification tree. The specification result is presented to the user in the "**Contracts**" dialog tab (see figure above).

Before a machine-readable requirements specification can be derived as C-Observers, a binding to an existing testing architecture/interface has to be established. For this reason, the user can import such architectures into the tool-suite. The binding between a semi-formal macro specification and the real variable/signal world of an execution layer of a system under verification/test has to be guided by the user once. For this, the user simply drags the desired specification object directly on the imported architecture. This is done on the specification tree visualized in the Profile Navigator. The tool-suite then automatically creates a new instance of such specification object, with the ability to refine the semi-formal parts into formal parts based on the corresponding



Architectures > Targetlink Model > ... > Macros

Name: CrashOccured

Notation: Formal

Description: a crash occures

Definition:

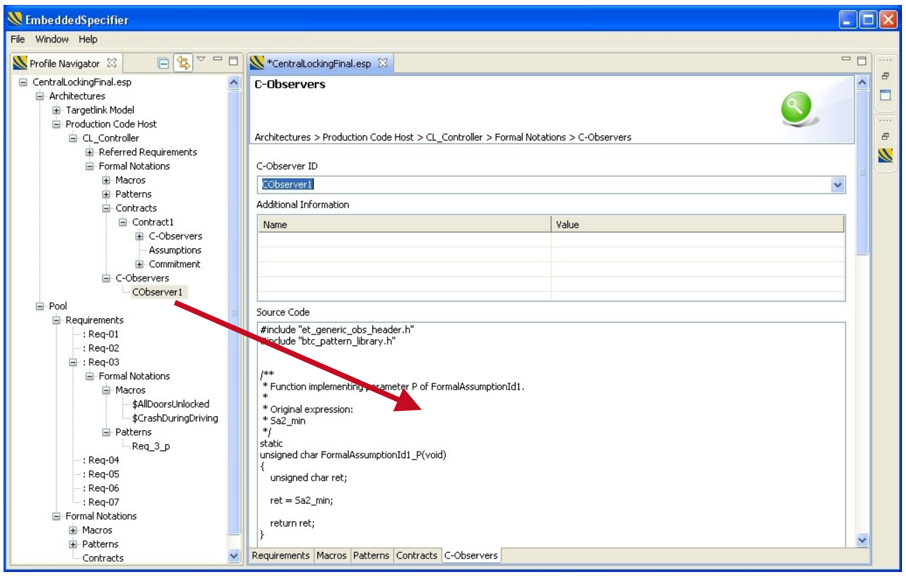
```
CRASH == 1
```

Architectures > Targetlink Model > CL_Controller > Formal N

Name	Type
Macros	
$CrashOccured	Macro
InPorts	
KLR	double
KL15	double
CRASH	double
ULOCKDD	double
LOCKDD	double
ULOCKPD	double
LOCKPD	double
ULOCKRKS	double
CSRKS	double
ULOCKTRUNK	double
LOCKTRUNK	double
DCDD	double
DCPD	double
DCTRUNK	double
Calibrations	
Displays	
OutPorts	
SECURED	double
DD_LOCKED	double
PD_LOCKED	double
TRUNK_LOCKED	double
MUDD	double

architecture/interface objects. The figure above shows the imported architecture/ interface on the right-hand side. A binding between the current macro specification, which links to an informal text, and the target test architecture can be performed by writing the corresponding "**Definition**" based on the accessible interface objects.

If the binding to the test architecture is done, C-Observer specifications for further automatic testing approaches can be generated fully automatically.



Architectures > Production Code Host > CL_Controller > Formal Notations > C-Observers

C-Observer ID: CObserver1

Additional Information

Name	Value

Source Code

```
#include "et_generic_obs_header.h"
#include "btc_pattern_library.h"


/**
 * Function implementing parameter P of FormalAssumptionId1.
 *
 * Original expression:
 * Sa2_min
 */
static
unsigned char FormalAssumptionId1_P(void)
{
   unsigned char ret;

   ret = Sa2_min;

   return ret;
}
```

The result can be seen in the figure above under the dialog tab "**C-Observers**".

Conclusion

The presented specification method enables the typical developer and/or tester of embedded control software to specify requirements in a formal way, to fill the gab between informal textual requirements and machine-readable specifications.
It provides a user-friendly way of requirements specification, which directly addresses the automotive ISO 26262 standard for functional safety for all defined Automotive Safety Integrity Levels (ASILs).
Due to the computer-aided way of specification refinements, and due to the intuitive step-by-step way of structuring informal requirements, formal specification methods becomes usable even for non-experts.
The automatic binding of existing testing architectures to the requirement specifications makes this method a very effective and efficient approach to enable automatic requirements-based testing. The key-technology here is the automatic C-Code-Observer-Generation out of formal specifications, which builds the bridge between the operational world (executable specifications) and the declarative world (requirement statements).
The automatic linkage mechanism, which accompanies the user during all specification stages, provides a lot of advantages, mainly complete traceability, readability and the possibility of automatic refactoring, if details of specifications will change over the life-cycle.
This described very flexible method can be used at any point in time during the software development process, either from the very beginning based on textual specifications, or directly on existing test execution stages, e.g. to test existing C-Code implementations etc.
The presented approach has been realized in a first version as a requirement specification environment called BTC EmbeddedSpecifier. It comes as a model and code independent tool environment to enable formal specification in general, but is also smoothly integrated in the existing BTC EmbeddedTester automatic testing tool-chain especially made for dSPACE TargetLink.

Bibliography

[ISO]	Road vehicles – Functional Safety, International Organization for Standardization, ISO 26262, 2011
[PATTERN]	Pattern Specification - BTC EmbeddedTester Version 2.7 Tool Documentation, BTC Embedded System AG, 2011
[LSC]	LSCs: Breathing Life into Message Sequence Charts, Werner Damm and David Harel, Formal Methods in System Design, 2001
[BTCEW2010]	Fully Automated Back-to-Back Testing to support ISO 26262 Compliant Software Development, Hans Holberg and Dr. Brockmeyer, Embedded World Conference 2010
[BTCEW2011]	ISO 26262 compliant verification of functional requirements in the model-based software development process, Hans Holberg and Dr. Brockmeyer, Embedded World Conference 2011