From Safety Requirements to Safety Monitors – Automatic Synthesis in Compliance with ISO 26262

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

Dr.-Ing Stefan Häusler
Product Manager, BTC Embedded Systems AG
An der Schmiede 4, 26135 Oldenburg, Germany
stefan.haeusler@btc-es.de

Abstract: The development of safety-critical electronic systems in the automotive domain is standardized by the ISO 26262 Road vehicles - Functional safety. Depending on the concrete risk classification (Automotive Safety Integrity Level, ASIL for short), necessary safety requirements and activities are specified in order to achieve an acceptable residual risk of the system.

In particular for the higher ASILs, the ISO 26262 standard recommends the application of model-based development processes and the usage of (semi-)formal languages to specific requirements. Besides process recommendations, the ISO 26262 standard itself recommends the use of external monitoring facilities and plausibility checks at the architectural level in order to be able to detect and resolve errors at run-time (ISO 26262-6, Clause 7.4.14). In its most simple form, such a safety monitor corresponds to a software watchdog that regularly observes the systems liveliness and triggers some kind of safety mechanism on error. A more advanced monitor is able to observe dedicated properties of the system in order to trigger suitable mitigation steps, e.g. in order to implement a graceful degradation of certain system parts. For short, safety monitors are responsible to check whether safety requirements are fulfilled during vehicle operation.

Therefore, a tool-assisted synthesis of monitoring functionality out of existing requirement specifications is clearly desirable for such architectures. First of all, it reduces the efforts in creating such functionality. Secondly, it increases the coherence and traceability between requirements and its implementation.

We will demonstrate, how a specification language for formal requirements, namely the Pattern Library developed by BTC Embedded Systems AG, can be used to automatically generate safety monitors within a model-based development process using Simulink® (developed by The MathWorks) and TargetLink® (developed by dSPACE). Furthermore, we will describe the necessary activities in order to embed this mechanism into an ISO 26262 compliant development process. Finally, we will show the benefits of lifting the development of monitoring functionalities to the level of model-based design. In particular, it will benefit from all its simulation and testing abilities along the development phases, including the early evaluation of safety mechanisms in rapid prototyping environments. The approach was evaluated successfully with European and Japanese customers from automotive industries.

1 Introduction

To reduce the efforts in creating safety mechanisms and to increase the coherence and traceability between requirements and its implementation, this paper presents a tool-assisted synthesis of monitoring functionality out of existing requirement specifications. In section 2, we will demonstrate, how a specification language for formal requirements, namely the Pattern Library developed by BTC Embedded Systems AG, can be used to automatically generate safety monitors within a model-based development process using Simulink® (developed by The MathWorks) and TargetLink® (developed by dSPACE). In section 3, we will describe the necessary activities in order to embed this mechanism into an ISO 26262 compliant development process. In section 4, we will show the benefits of lifting the development of monitoring functionalities to the level of model-based design. In particular, it will benefit from all its simulation and testing abilities along the development phases, including the early evaluation of safety mechanisms in rapid prototyping environments.

2 Requirement-based Synthesis of Safety Monitors

In this section, we describe the synthesis workflow of safety monitors out of formalized requirements given in terms of declarative patterns. The overall workflow shown in Figure 1 illustrates the different translation steps.
Input for the translation process is a pattern based formal requirement specification using the BTC Pattern Library [PATTERN] as described in Section 2.1. The resulting pattern specification given in XML is translated to the instantiation code as described in Section **Fehler! Verweisquelle konnte nicht gefunden werden.**. Section 2.3 finally describes code generation and its aggregation into a self-contained C-Observer. For this C-Observer code, a CustomCode block is generated which has an input port for each system variable that is used in the parameter expressions and a single output variable that states whether the specification is currently satisfied or not. The code-generator TargetLink® for the modeling tool Simulink® will then integrate the underlying C code with respect to the connected input signals and will provide the actual evaluation of the pattern specification at the outport of the block.

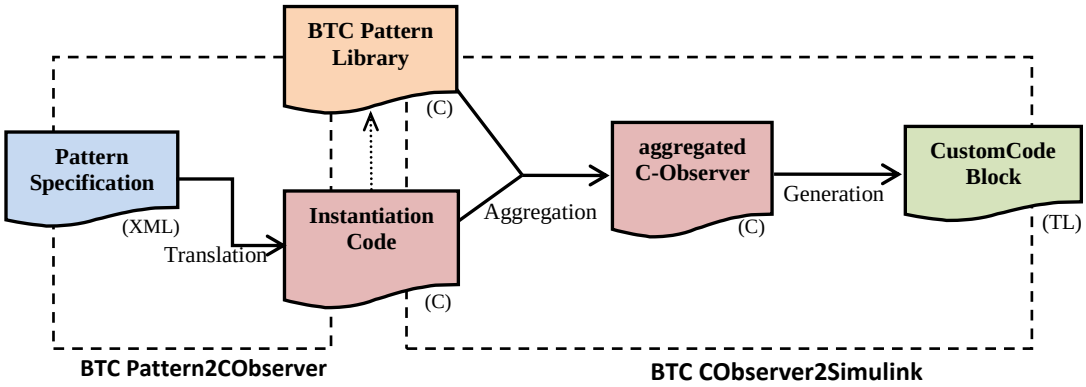

Figure 1: Abstract visualization of the generation steps.

To this end, we first introduce the pattern language in its abstract form and then describe how this language can be instantiated for obtaining code- and model-based observers in different variants.

2.1 The BTC Pattern Library

The BTC Pattern Library comprises a set of pre-defined specification templates and covers the most common types of temporal requirements used for the specification of embedded systems. It thereby eases the transition from an informal specification (e.g. a textual description) to a mathematically precise representation of the requirement. To do this translation, the user chooses a suitable pattern template and instantiates the pattern template parameters with concrete expressions for his requirement.
For example, the textual requirement

"The pressure must reach 3 bar at most 5 ticks after the valve has been closed."

can be formalized by using the pattern template

```
cyclic_P_implies_finally_Q_B
```

where the pattern template parameters P, Q, and B are mapped to suitable system expressions, e.g.

```
P : lastValveState == ST_OPEN && valveState == ST_CLOSE
Q : pressure >= 3
B : 5
```

The semantics of each pattern template in the BTC Pattern Library is precisely defined by an automaton description. These automata capture the temporal aspects of the patterns by distinguishing accepting from non-accepting system runs, which are well-understood concepts in the computer science.

The formal automaton description for the pattern template example used above is shown in Figure 2. Starting at the initial state 0, the automaton goes to state 1 when proposition P is true and simultaneously starts a counter X with value zero. If Q becomes true before the automaton has taken max_X times the self-loop at state 1, it enters the initial (accepting) state 0. If X reaches max_X and Q is not satisfied, the non-accepting state 2 is entered. That is, only those system runs where proposition Q becomes true at most max_X steps after each point in time where proposition P has become true, are accepted by the automaton.

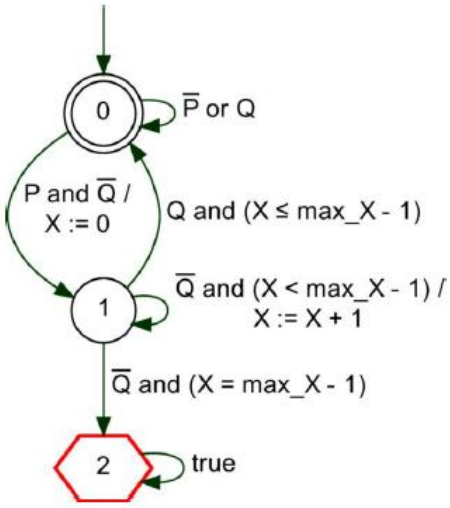

Figure 2: Automaton description for cyclic_P_implies_finally_Q_B

Note that the syntax and semantics of the BTC Pattern Library itself only reasons at this abstract level of propositions (i.e. the pattern template parameters) and numeric bounds, and has to be properly instantiated by assigning the concrete system expressions to its parameters.

In order to be usable for e.g. verification purposes, the BTC Pattern Library also provides pre-defined functions (written in the C programming language) corresponding to the pattern template automata. These C functions expect the pattern template parameters as arguments and return a value (of type (effectively) boolean) that states whether the underlying pattern specification is satisfied or not. At the C code level, the instantiation of a pattern template then corresponds to calling the pattern templates' C function with the concrete expressions as arguments.

Technically, a formal requirement based on the BTC Pattern Library can be specified using BTC EmbeddedSpecifier® [BTCEW2012], a tool to ease the formal specification process starting from informal requirements. This tool is able to generate the needed XML format that contains both the employed pattern template and the mapping of the pattern parameters to the system expressions.

2.2 Pattern Instantiation in C

As a convenient way to actually use the BTC Pattern Library, the tool BTC Pattern2CObserver translates an XML-based pattern specification to a C code fragment. The generated C code instantiates the pattern library as described above, by creating the corresponding assignment statements and function calls. For example, the pattern specification example given above would be translated the following kind of C code. Note that the actual code generated by the current version of the tool includes some more statements to account for certain integration aspects, but the basic structure is faithfully reflected in the following code snippet.

```
#include “btc_pattern_library.h”

static unsigned char PatternId1(void)
{
   static btcpatlib_state_t state;

   unsigned char P = (lastValveState == ST_OPEN && valveState == ST_CLOSE);
   unsigned char Q = (pressure >= 3);
   int B = 5;

   return btcpatlib_cyclic_P_implies_finally_Q_B(&state, P, Q, B);
}
```

The state variable is a C structure which holds the current state of the pattern automaton and the current values of the numeric counters. Its type is defined by the BTC Pattern Library and is used by the instantiation code as shown in the code fragment above.
Besides integrating this C code into the final production code, it can, for example, be imported with BTC EmbeddedTester® as a so-called C-Observer in order to automatically generate test vectors that cover the underlying pattern specification [BTCEW2011].

2.3 Synthesis of Safety Monitors

In a final step, the C code generated from the pattern specification is integrated into the final production code in order to use the boolean return value of the C function as a run-time observer. As already laid out in the introduction, such a pattern specification then typically expresses a certain safety condition, where a run-time violation of it needs to be mitigated by appropriate reactions, e.g. by entering a safe system state.
BTC Embedded Systems AG has developed the tool BTC CObserver2Simulink that allows to seamless embed the generated C-Observer into a model-based development process using Simulink/TargetLink. To this end, the tool takes the C

code representation of a pattern specification as input and generates a so-called TargetLink CustomCode-Block that internally contains the C code and the relevant portion of the BTC Pattern Library (see Figure 3). The user of the tool can then integrate this model entity by connecting the inputs and output of this block with his overall system model, and in particular use the output of the CustomCode block to trigger the appropriate system behavior.

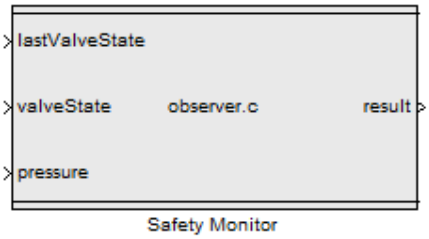

Figure 3: A generated CustomCode block containing the C code of an instantiated pattern.

3 Using Synthesized Safety Monitors in ISO 26262

In this section, we discuss the required tasks and activities that are necessary in order to enable the use of synthesized safety monitors in a development process according to the ISO 26262 standard. The relevant statements related to the use of external software tools and components are given in part 8 (“Supporting processes”) of the safety standard. In particular, clause 8.11 addresses the “Confidence in the use of software tools” and clause 8.12 defines the “Qualification of software components”.

Following this terminology, the BTC Pattern Library (in particular its set of pre-defined C function source code) can be considered as a software component according to clause 8.12. Hence, the library itself can be qualified accordingly by the vendor of the component. The additional instantiation artifacts (i.e. the code and model entities) that are used to instantiate and integrate the library depend on the actual system to be developed and hence fall into responsibility of the user of the component. This means that the appropriate activities of part 6 (“Product development at the software level”) of the ISO 26262 have to be ensured by the user, in particular unit testing of the instantiation code and verification of the integration aspects. The following figure visualizes the responsibilities.

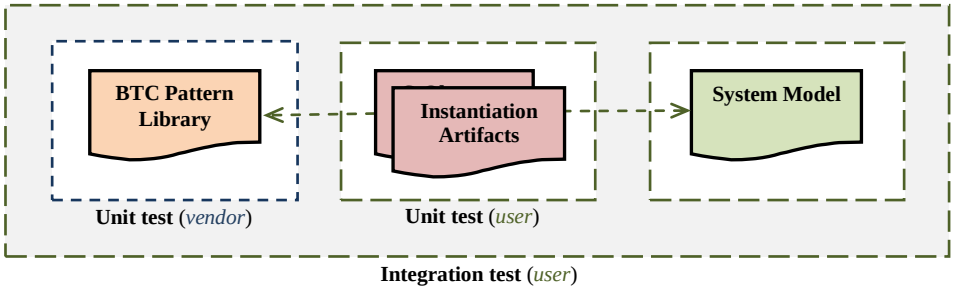

Figure 3: Visualization of the verification responsibilities.

In Section 3.1, we describe the qualification of the library component in more detail. The activities related to the translation and generation steps as described above are explained in Section 3.2. In Section 4, we focus on the model-based aspects of the proposed solution.

3.1 Qualification of the BTC Pattern Library

In ISO 26262-8 12.1, the objective of the qualification of software components is defined as “to provide evidence for their suitability for re-use”. Planning the qualification of a software component (12.4.2) includes the unique identification of the software component, the determination of the maximum target ASIL of any safety requirement to be allocated, and the identification of the activities to be carried out for qualification.

For the BTC Pattern Library, a unique library version will be fixed for identification. To support a broad class of applications, the library should be qualified with respect to ASIL D.

To be able to consider the library as qualified (12.4.1), the activities listed in the following subsection shall be carried out. Additionally, it has to be shown that the library is developed according to an appropriate national or international standard.

3.1.1 Specification of the Software Component (12.4.3.1)

This activity includes the specification of the components’ requirements and configuration, a description of its interface and integration and dependencies with other software components, its reactions under anomalous conditions and its known anomalies with possible workarounds. When appropriate, an application manual should be established.

The functional requirements of the BTC Pattern Library are already well-documented and in particular comprise a precise characterization of its semantics in form of automaton descriptions. Also, the complexity and resource usage of the library code has already been investigated. Some major findings are the following:

- no heap memory usage
- limited, statically determined stack/data segment usage
- constant determination complexity only dependent on the pattern type
- linear complexity depending on pattern number

The description of the interface and further integration aspects can be defined based on the signatures of the C functions. The behavior under anomalous conditions and known anomalies of the library itself will be documented according to the standard. An application manual which explains the temporal semantics of the pattern types and which defines the adequate instantiation of the library functions is to be provided by the vendor of the component.

3.1.2 Show Compliance of the Component with its Requirements (12.4.3.2/3/4)

The coverage of the components' requirements has to be shown by a suitable test suite on the unit level (in accordance with ISO 26262-6, Clause 9). For ASIL D, the structural coverage shall reach modified condition / decision (MC/DC) level. Both normal operation and the behavior in the case of failures shall be addressed. Note that the analysis is only valid for an unchanged implementation of the component.

The BTC Pattern Library can be verified independently from its actual usage on the unit level by the vendor of the component. Recall that the C function library implements the pattern semantics on an abstract level of propositions, and that the semantics is precisely given by means of automata descriptions. This allows for deriving suitable requirements-based test vectors, where a successful test means that the C implementation of a pattern template must return true if and only if the test vector is accepted by the corresponding automaton.

The automaton description will also ease the generation of the needed set of test vectors in order to reach MC/DC coverage of the source code. The task is to derive test vectors which, for each transition in the automaton, show that each conditional part of the transition annotation independently make the transition true or false, respectively. As the automaton transitions are directly mapped to conditions in the source code of the library, the MC/DC coverage will also be reached here. Note that this coverage is only guaranteed with respect to the host code and compiler, and a corresponding coverage for the actual employed target has to be ensured by the user. To this end, the set of test vectors obtained at the host can be used as a validation suite and re-executed by the user for the actual target system.

3.1.3 Show the Suitability of the Component for its intended Use (12.4.4)

This purpose of this activity is to ensure that the qualification of the software components including the validity of the results matches the indented use of the component. This is in particular needed when the components development context significantly differs from its application context, e.g. when transferring components between different industrial or automotive domains. Evidence has to be provided that the specification of the component complies with the requirements of the indent use.
By its design, the functionality of the BTC Pattern Library is almost independent of its application such that no special transfer from different development and application context has to be provided. Still it is of importance that the user of the library has the same understanding of its functionality as the component vendor. The mathematically precise specification of the pattern semantics help in this context, still the non-trivial semantics, in particular of the more complex pattern templates, require a good understanding of the temporal interrelations. Beside the functional specification, the supporting textual description of the BTC Pattern Library and its semantic characteristics given in the application manual need to be reviewed and accepted by the user of the software component.

3.2 Handling of the Instantiation Artifacts

Typically, the generated safety monitor will be used to (partly) implement a safety requirement that has been identified in a prior analysis phase of the development process. Hence, the generated instantiation code as well as the generated CustomCode block has to adhere to the general requirements imposed by ISO 26262-6 (“Product development at the software level”). The user of the component is responsible for this activity. In particular, both artifacts have to be tested on the unit level according to ISO 26262-6 Clause 9 (“Software unit testing”). The concrete activities for this process steps depend on the ASIL that has been allocated to the underlying safety requirement. At least, a requirements-based test and an interface test are included in these activities, and a certain structural coverage of the used test vectors has to be reached. In fact, there are not many decisions to be covered in the structural part of the instantiation artifacts as most of the code lines are solely needed to call the pattern library functions with the expressions given for the pattern template parameters. The expressions itself are provided by the user of the pattern library and have to be covered according to the required metrics.

The combination of the pattern instantiation code with the actual pattern library function is covered by ISO 26262-6, Clause 10 (“Software integration and testing”). Again, the concrete activities of the user depend on the underlying ASIL, but at least a requirement-based test and an interface test has to be performed on the integration level. The structural code coverage has to ensure function or call coverage, which, due to the linear control flow within the instantiation code, should be easy to achieve.

As the generated pattern instantiation code (and the corresponding CustomCode block) follows a well-defined construction schema, it is meaningful and feasible to include a manual walk-through of the generated code into the overall development process. This step in particular verifies that the BTC Pattern Library is used according to its application manual.

The confidence level for the tools is determined according to ISO 26262-8 Clause 11.4.5.2 as follows. As a malfunction of the tools has the potential to introduce errors in the safety-related item, a tool impact of TI2 has to be selected for the tools. By the activities described above, a complete examination of the output of the tools that generate the instantiation artifacts and the CustomCode block can be guaranteed. Thus, a tool error detection level of TD1 can be selected for these tools. In combination, this leads to a tool confidence level of TCL1 for which no further qualification methods need to be performed (ISO 26262-8, Clause 11.4.6.1).

4 Exploiting the Model-Based Approach

Note that the generated CustomCode block in particular supports the seamless integration of the synthesized monitors in a model-based development process. This kind of development in general supports a systematic and coherent view on the requirements-, design-, implementation- and verification- phases of the development. The ISO 26262 specially addresses model-based development and testing, e.g. in ISO 26262-6 Annex B. Especially for the integration testing phase, a back-to back comparison between model and code is recommended for higher ASIL in ISO 26262-6 Table 13. The basic principle is to apply a set of test vectors on both representations of the system and compare the results. If these test vectors cover a sufficient amount of the systems’ behavior, errors that have been introduced during the transition from the model to the code representation are uncovered. The same principle can be applied on different code levels, e.g. to compare the behavior of the host code with the final production code on the target system.

As the BTC EmbeddedTester® is aware of the C-Observer also as native source code, one can in particular define a tailored test that ensures a correct generation

and integration of the corresponding source code as a CustomCode model block. Here, a test vector which satisfies the C-Observer must also lead to a true output of the CustomCode block, and vice versa. Also, the further back-to-back testing activities during the subsequent integration and validation phases, which in particular include the safety monitors, can be handled by the BTC EmbeddedTester® as a qualified testing tool according to ISO 26262.

5 Conclusion

In this paper, we have presented how safety monitors can be synthesized from formalized requirements. By a pattern-based generation of such monitors on a model-based design level, the generated monitor functionality can benefit from the full range of model-based development advantages, including early simulation capabilities and derivation of test vectors for subsequent back-to-back testing for its integration up to the final production code.

We have in particular discussed the required activities for embedding the safety monitoring in an ISO 26262 compliant development process. This includes the qualification of the kernel pattern library as a software component according to ISO 26262-8, 11. Additional (testing) activities for using the library have been discussed from the view point of the component user. In general, the derivation of safety functionality out of existing safety requirements fits well into the spirit of the ISO 26262, as it provides a stringent traceability from the requirement to its implementation and vice versa. Also the usage of model-based design principle is encouraged by the ISO 26262 standard and is supported by the synthesis mechanism as proposed in this document.

The BTC Pattern Library itself provides a convenient way to formulate unambiguous specifications even for non-trivial embedded systems. While a precise specification itself is already of great value, its re-use for synthesizing parts of its implementation increases its benefit even more. Although most steps of the synthesis can be automated by tools, a thorough review and test of the output is needed, in order to both discover potential errors in the generation itself but also to ensure that the intended semantics of the safety requirements is correctly captured by the obtained safety mechanism. Using a qualified library will help the user in this activity as it allows focusing on the requirement itself. The approach was evaluated successfully with European and Japanese customers from automotive industries.

Aknowledgement

The work presented in this paper has been supported by the ARTEMIS-JU project MBAT.

Bibliography

[ISO26262] Road vehicles – Functional Safety, International Organization for Standardization, ISO 26262, 2011

[PATTERN] Pattern Specification - BTC EmbeddedTester Version 2.9 Tool Documentation, BTC Embedded System AG, 2012

[BTCEW2011] ISO 26262 compliant verification of functional requirements inthe model-based software development process, Hans Holberg and Dr. Brockmeyer, Embedded World Conference 2011

[BTCEW2012] Computer-aided Formal Specification to enable a fully automated Requirement-based Testing Process, Hans Holberg and Dr. Brockmeyer, Embedded World Conference 2011