Project Summary

ACSÄPT

Acceptanstest av säkerhetskritisk plattformsprogramvara

FFI

Fordonsstrategisk
Forskning och Innovation

SP Your Science Partner

VOLVO

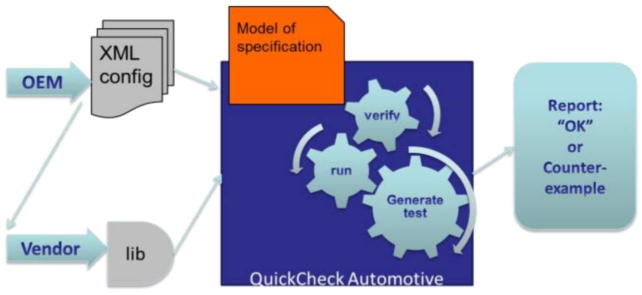
QuviQ

The Project

In this report we summarise the results of the FFI-project AcSäPt (Acceptanstest av säkerhetskritisk plattformsprogramvara). The overall objective of the AcSäPt project has been to develop and define an acceptance test method that can show adherence to the ISO 26262:2011 standard for safety related software.

Our solution is based on creating configurable formal models of all relevant AUTOSAR BSW (basic software) components. These models can be used for finding out which requirements from the software specification are safety-critical for a specific use-case. They can also be used for generating test cases for black-box testing of a BSW module from a supplier; the implementation can be tested until we achieve 100% coverage of the safety related features and scenarios.

The remainder of this report describes both the problem and our solution in more detail.



Model-based testing

The project started in 2012 and has run for two years with five partners: SP, Quviq, Volvo Cars, Mentor Graphics, and Mecel.

The Problem

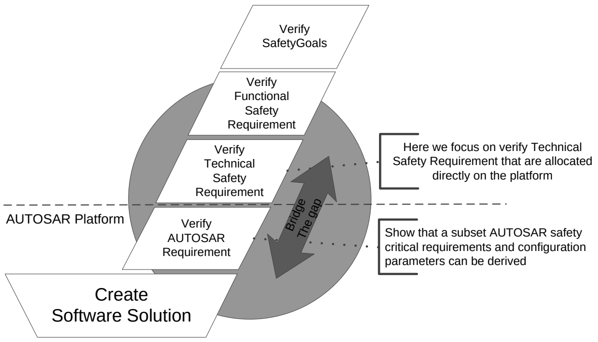
The AcSäPt project addresses the following questions:

- How can a vehicle OEM, such as Volvo Cars, ensure that the AUTOSAR platform software (BSW) of each ECU in a car is functionally safe?
- Can the OEM specify acceptance criteria for safety-critical BSW?
- What are the methodological implications?
- How to handle variant and version problems?

According to the functional safety standard for road vehicles, ISO26262, it is important to identify which parts of the electrical/electronic (E/E) systems are safety related, and which safety requirements will be put on them. All safety requirements are derived from hazard analyses, which are performed for the different functionalities that the E/E systems provide. In our project we investigated such safety requirements that are allocated to the AUTOSAR platform software. We addressed the challenge on how to claim that a particular implementation meets these safety requirements.

We identified five problems that make it hard in general to give arguments and evidence why a certain piece of BSW fulfils all safety requirements in a given context.

1. There is a functional gap between application and platform, and thus a gap between the implications of functional safety. Although it is clear how to break down the safety requirements of a particular functionality to functional subsystems and -components, it is unclear how the safety requirements relate between application and platform components. This relation may be highly implicit. To find arguments and evidence of fulfilling the safety requirements put on BSW, we need to explicitly identify what they are.


Verify SafetyGoals
Verify Functional Safety Requirement
Verify Technical Safety Requirement
Verify AUTOSAR Requirement
AUTOSAR Platform
Create Software Solution
Bridge The gap
Here we focus on verify Technical Safety Requirement that are allocated directly on the platform
Show that a subset AUTOSAR safety critical requirements and configuration parameters can be derived

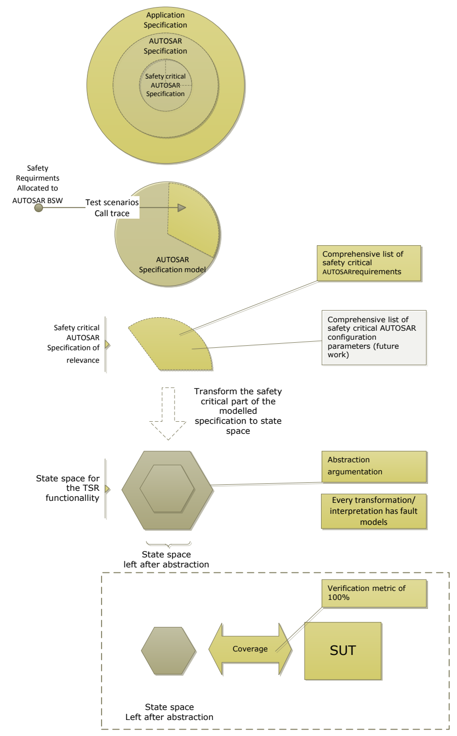
The functional gap between application and platform.

2. In the lingo of ISO26262 the BSW is almost always a Safety-Element-out-of-Context. This means that the BSW is developed before knowing all the functions that are going to use it. As a consequence we will not have explicit Technical Safety Concepts (TSC, terminology of ISO26262) of all the applicable items ready when the BSW development will start. This contrasts the life cycle model of ISO26262, which demands all safety requirements to be identified at the beginning of the development process.
3. Even if we make all safety requirements explicit, it is still hard to come up with a safety argumentation that is valid for higher automotive safety integrity levels (ASIL) based only on product evidence. This is an important point when we want to specify acceptance test criteria for safety critical platform software. It has been shown that coverage metrics applied in a traditional way will not give enough evidence for a safety argumentation. We need coverage criteria that can be applied for safety argumentation in an acceptance test.
4. It is possible that the BSW is delivered as a black box to the vehicle OEM. How does the OEM check that the BSW meets the stated safety criteria in case the source code is not provided?
5. The BSW is highly configurable. It might be very hard to say anything about the relation between two different configurations

in terms of how well they fulfil the safety requirements of concern. This is important when building the safety argumentations for all variants and all versions.

These five problems, and any combination thereof, imply that it is hard to analyse whether a certain BSW implementation can be regarded safe in a specific system context.

The AcSäPt Methodology


Application Specification
AUTOSAR Specification
Safety critical AUTOSAR Specification
Safety Requirments Allocated to AUTOSAR BSW
Test scenarios Call trace
AUTOSAR Specification model
Comprehensive list of safety critical AUTOSARrequirements
Safety critical AUTOSAR Specification of relevance
Comprehensive list of safety critical AUTOSAR configuration parameters (future work)
Transform the safety critical part of the modelled specification to state space
State space for the TSR functionallity
Abstraction argumentation
Every transformation/ interpretation has fault models
State space left after abstraction
Verification metric of 100%
Coverage
SUT
State space Left after abstraction

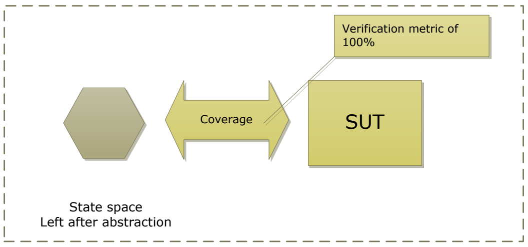
The AcSäPt methodology.

The AcSäPt project has addressed all of the above problems. The overall methodology is to create a configurable formal model of all relevant BSW components. This model is used in a number of different ways that together solve the problems specified. In this section we explain briefly how we solved the problems. In later sections we give a more detailed description.

The model can be regarded as an executable specification of the BSW. The model is annotated with the individual requirements from the specification documents (SWS). By executing safety related application use cases (i.e., uses cases that are associated with particular safety requirements) that make use of the BSW we can trace which requirements from the SWS specification are relevant. We use this information to bridge the gap between the application and platform context, and identify what the safety requirements are on the BSW. In addition, we also use the model to determine whether or not a configuration parameter is relevant for a particular use case.

Our acceptance test procedure allows the BSW supplier to provide the implementation as black box (i.e., without the source code). Each implementation is tested until we obtain 100% coverage. The 100% test coverage is reached when we have tested all requirements within all possible scenarios and with every possible input. We argue that it is better (sic!) to use a dedicated test configuration for this procedure, than the actual configuration used in an ECU. A dedicated test configuration can be adjusted such that it is easier to reach the 100% coverage.

100% coverage gives argument for acceptance test for any specific context, not relying on the BSW life-cycle out of context. If the metrics showing 100% is defined carefully this can be used as safety evidence.



Coverage of state space after abstractions.

To reach our 100% coverage in practice (days, not years), we apply a combination of formal completeness in the state space of the model, and reasonable fault models of any implementation. By applying a carefully chosen test configuration we can reduce testing time by several orders of magnitude, while still being able to reach get 100% coverage. If the test configuration is chosen correctly we can show that the 100% coverage still holds in the different real ECU configurations of concern.

A Deeper Look into the Models and the Testing

The BSW is modelled as an executable specification in a functional language. Each BSW module is a model of its own. These models are composable such that a certain ICC2 cluster (AUTOSAR term for integration conformance class) may be tested directly against the composition of the models. Each model is configurable in the same way as the BSW itself, i.e., by the same ECU configuration parameters in XML format.

Testing a BSW implementation is done by repeatedly executing the BSW (compiled for a PC test environment) with a sequence of API-calls (random valid API calls with valid data) and comparing the actual output to the output predicted by the model. There are no dedicated test cases for a SWS requirement or use case. Instead, testing continues until 100% of coverage is reached (or a failure is detected) with respect to the SWS requirements considered as safety relevant in a given context. During each individual test case (that is a sequence of API-calls) we check which SWS requirements have been covered. In order to reach 100% coverage it is necessary to generate a number of unusual test sequences, which would not typically be present if only one feature at the time was tested.

We reach 100% coverage when ‘all’ information in the BSW models are challenged with respect to the SWS requirements of concern. The model can be numerous different states and all these states are easily collected, containing information about requirements, specific API call data, and timer information. When all possible states have been visited in at least one test, we reach 100% coverage. From the model point of view this means that all internal model states of concern have been exercised in the test campaign. The testing tool stops generating new tests when 100% coverage is reached. As is discussed below, reaching 100% in a reasonable test time requires also some argumentation of what kind of failures that an implementation may have. The argumentation we use is translated into rules that become part of the model itself, which guarantees the safety testing to terminate in reasonable time still with high enough confidence for claiming ASIL D.

The reason why it is possible to reach 100% coverage is also partly because of the size of the test object. By accessing an ICC2 cluster directly with the API on its border, both controllability and observability is increased compared to if only the RTE and the field bus were the interfaces. In this project we have shown that it is feasible to reach 100% confidence for some of the modules and clusters as defined by Volvo. A slightly larger cluster would probably be manageable, but it would require too many test sequences if the entire BSW was to be tested as one monolith.

The fact that the models are composable and configurable makes them well suited for testing different sets of safety requirements and different versions and variants of AUTOSAR basic software.

Reaching 100% Coverage

Claiming a high ASIL is to say that the risk of remaining safety-critical bugs is very low. There is no way to argue safety integrity by means of test coverage, except by reaching 100%. Furthermore, we claim that some popular coverage metrics like MC/DC of the code are insufficient: even 100% MC/DC coverage can slip errors through. Using traditional requirement coverage is also not enough if this means that we just have to test each requirement in isolation and not all the strange combinations. Our conclusion is that 100% coverage to use in safety argumentation requires that all requirements are tested for all possible scenarios of input sequences. This means that many unusual sequences needs to be tested, not only the typical ones that hopefully were used in the first module tests by the designers themselves.

If we apply this demand for testing all possible input sequences very strictly, then we would require extremely many test sequences in which also all possible data paths, e.g. different PDU data values, should be considered. This cannot work, since there are too many sequences and too many data paths to consider. Our approach is to maintain the demand of 100% coverage with respect to the state space of the requirement model, but to make explicit what failure models that we assume for the implementation. One such assumption is data symmetry for most data path elements, like the PDUs. This assumption is based on that fact that the functionality of BSW implementations is based on algorithms that are independent of the content of the PDUs themselves. If this argument is assessed as valid, we only need to consider some typical and some corner cases. These assumptions are annotated in the models and are checked by the testing tool when checking when 100% is reached. The assumptions are made explicit and can thus be challenged by a safety assessor. Of course, they can also be communicated to the BSW supplier as check list questions regarding the implementation method.

Configuration choice for claiming safety

In current version of ISO26262 it is stated that testing software for safety should be performed with the actual configuration that will be used. At a first glance this requirement seems reasonable. In order to get high confidence it is not enough to test in any other configuration than the one that will be used in the safety-critical system.

The conclusion of the AcSäPt project research is the direct opposite. We claim that for most cases we can argue for safety integrity only when using a dedicated test configuration, and not for an actual configuration. The reason why we allow a test configuration, is because we can show that this configuration enables as much of the test model as any of the actual configurations. We can check this completeness by comparing the configured models and check that everything that is checked by the actual configuration also is checked by the test configuration. The problem with most actual configurations is that they are too large to reach 100% coverage in reasonable time. The conclusion is that in order to argue for safety we are more effective when using a test configuration then when using the actual configuration. The only additional test one does when using the real configuration is that the configuration tools are correct.

Comparison with other Methodologies

How does the proposed test procedure for safety assessment differ from other methods of BSW testing and from other methods of claiming safety?

In both the AUTOSAR community and among some OEMs, there have been defined acceptance tests for AUTOSAR BSW. However, none of these aim to reach the extreme degree of completeness needed for safety argumentation. In the acceptance test proposed by the AUTOSAR community, the BSW is tested on hardware target only accessible through RTE and field bus. This kind of testing aims at reducing the amount of problems showing up in the integration tests, and it would be very hard both to identify the coverage criteria for 100%, and to reach such test completeness in reasonable time. This test strategy is considered well suited for its purpose, but not applicable when claiming safety.

In the AUTOSAR BSW non-safety tests done for Volvo Cars so far, there is evidence that there have been bugs in candidate implementations that would not have been found when applying traditional dedicated test cases. This supports the demand for 100% coverage also with respect to unusual combinations of API calls.

According to how ISO26262 is used today, safety argumentation of BSW would be equal to claiming ASIL capability (no explicit safety requirements) mainly supported by process arguments. The AcSäPt recommendations can be regarded both as a way of being more precise of what is to be shown safe, and a methodology enabling the OEM to reproduce safety evidence for several variants and versions at the time.

Summary and Conclusions

The main result of the AcSäPt project is a methodology to perform acceptance tests by the OEM suited for safety-critical AUTOSAR BSW. This methodology addresses how to claim safety based on the observations of the delivered product (product arguments), and is not so dependent on the review of the development lifecycle (process arguments) as is the case today. The product-based safety argumentation relies on the fact that it is possible to reach 100% test coverage with respect to the applicable safety requirements. The methodology includes identification of the safety requirements applicable for the BSW. The 100% coverage metrics includes correctness of functionality in all possible combinations of input sequences and data paths. The coverage metrics is based on the formal model of the specification, and no source code of the BSW is needed. The safety acceptance tests are performed using dedicated test configurations. The conclusion of the project is that it is better to use test configuration than actual configuration, in order to show safety.

To conclude, the project shows feasibility of the methodology, and the recommendation is to continue to apply this for AUTOSAR basic software.

Contact Information

If you have questions, please contact:

Quviq AB	Thomas Arts, 070 4388567, thomas.arts@quviq.com
SP	Rolf Johansson, 010 516 5546, rolf.johansson@sp.se
Volvo	Fredrik Törner, fredrik.torner@volvocars.com

Further Reading

Martin Skoglund, Hans Svensson, Henrik Eriksson, Tomas Arts, Rolf Johansson, and Alex Gerdes. *Checking Verification Compliance of Technical Safety Requirements on the AUTOSAR Platform Using Annotated Semi-Formal Executable Models.* SafeComp 2014 Workshops, pp. 19-26, 2014.

Rolf Johansson, Henrik Eriksson, Hans Svensson, Kenneth Östberg, Thomas Arts, Alex Gerdes, and Martin Skoglund. *Don’t Judge Software by Its (Code) Coverage.* CARS 2013.

Thomas Arts, Michele Dorigatti, and Stefano Tonetta, *Making Implicit Safety Requirements Explicit - An AUTOSAR Safety Case*, SafeComp, Firenze, Italy; 09/2014.

Rickard Svenningsson, Rolf Johansson, Thomas Arts, and Ulf Norell. *Formal methods based acceptance testing for AUTOSAR exchangeability*. No. 2012-01-0503. SAE Technical Paper, 2012.

Rickard Svenningsson, Rolf Johansson, Thomas Arts, and Ulf Norell. *TESTING AUTOSAR BASIC SOFTWARE MODULES WITH QUICKCHECK*. Advanced Mathematical and Computational Tools in Metrology and Testing IX 84 (2012): 391.