Integrating the Concept of Standard Software into a Certifiable Development Process

Wolfgang Friess
AUDI AG
wolfgang.friess@audi.de

Dr. Franz Duckstein
3SOFT GmbH
franz.duckstein@3soft.de

Abstract: The need for a certifiable development process for automotive systems is increasing due to the rising number of safety-relevant software functions. In this article, we describe the actual work in the Bavarian research project *mobilSoft* on this topic and the usage of configurable commercial-off-the-shelf (COTS) standard software modules as a special aspect of automotive software development. Furthermore, the necessity for a formalised integration process for such COTS-modules as a precondition of a certifiable development process is explained and a concept for such a process is introduced. Finally, an outlook on further steps is given.

1 Introduction

As more and more functions realised by software directly influence the driving behaviour, the number of safety-relevant software functions will increase in the next years. A structured and certifiable development process is the key to ensure safety and quality. An important aspect in automotive software development is the usage of statically configurable commercial-off-the-shelf (COTS) software modules for system software functions. Typical examples for such modules are operating systems and bus drivers. Actual development processes focus on safety-relevant application software. To integrate COTS-modules into a safety-relevant system, a certified integration process for such modules has to be applied. In the following section, the different steps to integrate standard software modules are identified and described. After that, a general process for integrating standard software modules into an ECU software system is introduced.

2 Integration of Standard Software

Standard software often provides some safety related functionality itself, like the Protected OSEK [AG04] or AUTOSAR COM [1]. If standard software provides mechanisms for error detection, it must be assured that these mechanisms work correctly. Therefore, the configuration of standard software must be done without faults impacting the safety mechanisms,

[1] www.autosar.org

on which the complete system relies. So the configuration has to be 'safe', or the standard software has to be secured against configuration faults. The process of integrating COTS standard software modules into an ECU-software system can be divided into three main steps:

1. selection of software modules
2. parametrisation of the selected modules
3. building of the resulting binaries

Usually these steps are performed without explicit control. Therefore they are strongly depending on the individual skills of developers and integrators. For these steps we need to prove, that all possible sources of defects are identified, and that all of these defects will be discovered during configuration. These three configuration steps are controlled by a combination of project requirements, explicit constraints between or within the software modules, and individual decisions including existing experiences.

Selection of Software Modules

On the base of certified modules a controlled selection process will include

- documentation of selection
- documented proof of completeness on module dependency level[2]
- documented proof of compatibility

The proofs will be based substantially on the given module documentation. In order to provide tool support, the documentation must be formal. Possible check items are resource or scheduling constraints within the combination of modules. Example: an implicit module dependency by shared memory communication will potentially not be discovered by the linker.

Parametrisation of the Selected Modules

The selected software modules are the initial configuration, which should be subject of protection to avoid unintended manipulation. Software modules in the embedded area usually support functions with a broad flexibility. The use of a module needs a proper parametrisation per project, typical examples are

- hardware settings like controller-derivate, frequency or memory areas
- software settings for scheduling, communication or functional behaviour
- compilation settings like switches or optimisation

[2] all primary dependencies between the selected modules are resolved (e.g. the versions of the selected moduls are compatible and there are no unresolved external functions left)

As a prerequisite for certification, the parametrisation per module must also be documented. This includes for each parameter the dependencies to other parameters, the parameter effect, the necessity of use and the reasoning of choice. The verification of a parametrisation includes

- review of completeness per function (all affected modules)
- proof of compliance with parameter constraints
- prevention of unauthorised or unintentional changes

Constraints are a commonly used technique. They provide additional information like valid ranges or dependencies. As a typical solution, the parameter constraints are defined in a separate file (e.g. in XPATH or OCL format). If the parametrisation is encapsulated by a tool, parameter settings in contradiction with given constraints are blocked.

Building of the Final Binary

The build process is also challenging, in order to guarantee the proper input data, the intended creation sequence and its final results. Items of interest are

- using the proper tools (type and version)
- unambigous processing, secured against disruption or wrong intermediate results
- completeness of build and its documentation

The created results (binaries) should provide clear attributes, like generation settings, hash values and a significant labelling. This can be used during further processing to protect against unintentional changes.

Using separate tools for the different standard software configuration steps is state of the practice. But with the rising number of safety-relevant software functions and the increasing importance of certifiability, a seamless tool chain based on a formalized configuration process is needed.

3 Formalised Integration Process

To prove the correctness of standard software integration in real life projects, a formalised process and an appropriate tool support based on this process is essential. Therefore, a formalised integration process is introduced in the following. The starting point for the integration process are the requirements, that must be fulfilled by the standard software modules. These requirements arise from different sources, like the application, the used microcontroller or the network topology, the ECU is integrated in. For example, if the ECU is connected to a CAN network, a CAN driver suitable for the used microcontroller has to be selected. Detailed information about the network (like the used baud rate or network topology) is needed, in order to complete the step of CAN driver parametrisation.

Starting from these requirements, the steps of selection, parametrisation and building must be connected by the formalised process. To enable a seamless integration process a common description for all the possibilites of the three integration steps is needed. Therefore, the sum of all configuration possibilites is considered as a configuration space which is influenced by different requirements. This configuration space concept is the basis for the tool chain supporting the integration process. Figure 1 shows this concept.

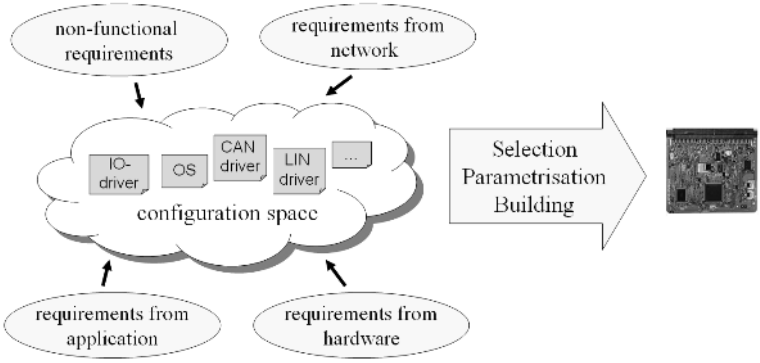

Figure 1: configuration space concept

The implementation of the configuration space can be done for example by a properitery data model or by standardised models like UML. The actual choice depends on outer circumstances like the surrounding tool framework or the established development process in the considered company. To support a seamless fulfillment of the requirements, we investigate feature models for describing the configuration space. Feature models were introduced by Kang [KCH+90] and are widely used in the domain of generative programming [CBUE02] and software product lines [Beu03] in order to model the configuration options in software systems. The formalised integration process, which is the precondition for a certifiable integration process, combines the configuration space with the integration steps explained in section 2. An overview about this process is shown in figure 2.

This formalised integration process is the basis for establishing a certifiable configuration process and a seamless tool support for the configuration of standard software which is still missing nowadays.

4 Outlook and Conclusion

The integration process described in this paper will be implemented based on existing tool solutions for the three integration steps explained in Section 2 namely the tresos configuration framework [Sch03]. After that, an extensive verification with several use cases will take place.

A certifiable integration process for standard software must consider the fact that several companies are part of the overall development process. Therefore, the research project

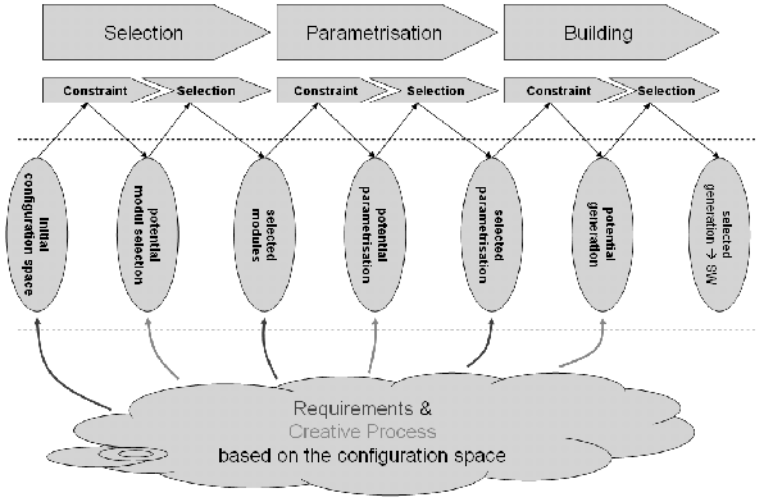

Figure 2: formalised integration process

mobilSoft is the platform to define such a common and certifiable process. This research project is partly funded by the Bavarian Ministry of Economic Affairs, Infrastructure, Transport and Technology.

The increasing number of safety-relevant software functions in modern cars offer a great potential for Bavarian companies to remain the technological leader of the automotive industry world wide, but a certifiable development process is the key for realising this benefit. The work in this research project is one step to face this future challenge.

References

[AG04] DaimlerChrysler AG. OSEK OS Extensions for Protected Applications. Technical report, HIS, http://www.automotive-his.de/download/HIS_ProtectedOSEK10.pdf, July 2004.

[Beu03] Danilo Beuche. *Composition and Construction of Embedded Software Families*. PhD thesis, Otto-von-Guericke Universität Magdeburg, http://ivs.cs.uni-magdeburg.de/ danilo/, 2003.

[CBUE02] Krzysztof Czarnecki, Thomas Bednasch, Peter Unger, and Ulrich W. Eisenecker. Generative Programming for Embedded Software: An Industrial Experience Report. In *GPCE '02: The ACM SIGPLAN/SIGSOFT Conference on Generative Programming and Component Engineering*, pages 156–172, London, UK, 2002. Springer-Verlag.

[KCH+90] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, November 1990.

[Sch03] Dr. Jochen Schoof. OSEK trifft Middleware. *Elektronik Automotive*, 6:28–33, 2003.