Freedom from Interference for AUTOSAR-based ECUs: a partitioned AUTOSAR stack

David Haworth, Tobias Jordan, Alexander Mattausch, Alexander Much

Elektrobit Automotive GmbH, Am Wolfsmantel 46, 91058 Erlangen, GERMANY

Abstract: AUTOSAR[1] is a standard for the development of software for embedded devices, primarily created for the automotive domain. It specifies a software architecture with more than 80 software modules that provide services to one or more software components.

With the trend towards integrating safety-relevant systems into embedded devices, conformance with standards such as ISO 26262 [ISO11] or ISO/IEC 61508 [IEC10] becomes increasingly important.

This article presents an approach to providing freedom from interference between software components by using the MPU[2] available on many modern microcontrollers. Each software component gets its own dedicated memory area, a so-called memory partition. This concept is well known in other industries like the aerospace industry, where the IMA[3] architecture is now well established.

The memory partitioning mechanism is implemented by a microkernel, which integrates seamlessly into the architecture specified by AUTOSAR. The development has been performed as SEooC[4] as described in ISO 26262, which is a new development approach. We describe the procedure for developing an SEooC.

---

[1] AUTOSAR: AUTomotive Open System ARchitecture, see [ASR12].
[2] MPU: Memory Protection Unit.
[3] IMA: Integrated Modular Avionics, see [RTCA11].
[4] SEooC: Safety Element out of Context, see [ISO11].

Introduction

The complexity of the functions that the software of an embedded device performs in a car increases significantly with every new car generation. AUTOSAR is an approach to manage the complexity by providing a standardized software architecture.

A further trend is the integration of several different functions on a single device, which has been simplified significantly by the AUTOSAR concept, as it defines a standard configuration and runtime environment for all suppliers of software components. It defines XML-based data exchange formats that the various suppliers can use to exchange their software components and data.

In this scenario, several software components are provided by different parties involved in the development. During integration, it is necessary to ensure that the different components do not interfere with each other. Interference between software components is not only a problem during development; it can also have legal consequences. This problem becomes especially important when software components with different criticality need to be integrated, for example with different safety integrity levels (SIL[5]/ASIL[6]). These highly integrated embedded devices are often referred to as *domain controller* – a central device that is responsible for all computation of a certain car domain.

We present an approach that integrates seamlessly into the AUTOSAR software architecture and provides freedom from interference in the spatial domain based on the highest (A)SIL levels.

[5] SIL: Safety Integrity Level, see [IEC10].
[6] ASIL: Automotive Safety Integrity Level, see [ISO11].

The AUTOSAR architecture

The AUTOSAR environment is a layered software architecture with about 80 separate modules. The general architecture is shown in Figure 1. The lowest layer contains the hardware-dependent modules, the microcontroller abstraction layer (MCAL) together with the operating system (OS). The basic software (BSW) modules are placed on top of the MCAL and are hardware-independent. The interface to the application layer is provided by the runtime environment (RTE).

The application layer is comprised of several independent entities, the *software components* (SWC). The SWCs themselves contain the tasks and ISRs that provide the functionality of the ECU. The communication between SWCs and with the underlying BSW is handled by the RTE, providing a complete abstraction from the drivers and the hardware and allowing completely hardware-independent application development. The protection measures for safety-relevant functionality, for which a certain safety integrity level is required, are in general also contained within the application.

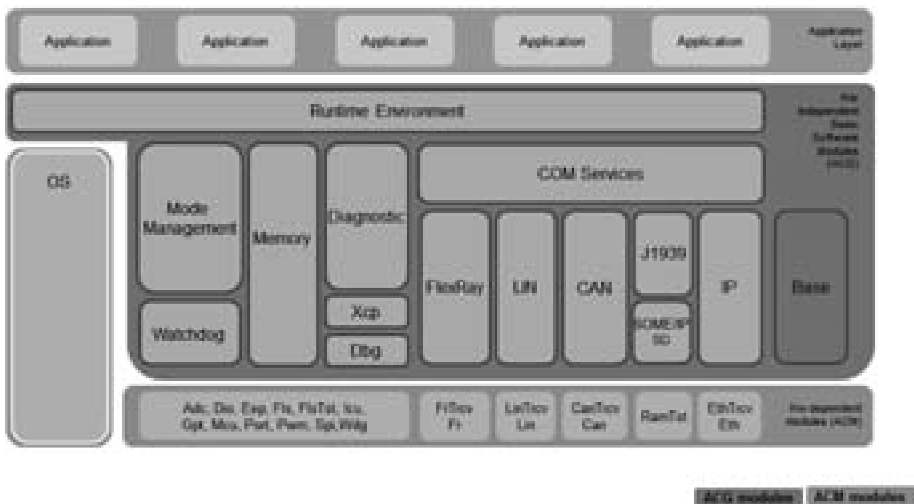

Figure 1: The AUTOSAR software architecture (Release 4.0)

However, although the SWCs are decoupled and independent of each other, the standard AUTOSAR architecture assumes that the entire basic software, including OS and RTE, runs within a single privileged context. Thus, although the safety mechanism of the device may reside within a single SWC, this SWC depends on the correct execution of the entire AUTOSAR stack. Freedom from interference is therefore not guaranteed, and as a consequence not only the entire OS but also the entire AUTOSAR stack inherits the highest integrity level.

A solution to this dilemma would be to establish freedom from interference between the SWCs and the BSW using the OS. The following sections describe such an approach.

A partitioned AUTOSAR stack

A common approach to separating software components from each other is partitioning. The software components run in isolated areas and any interference is inhibited by a memory protection mechanism. In safety-critical environments, partitioning allows the integration of software components of different (A)SIL levels on a single system.

The partitioning mechanism is best placed in the operating system, which will be described in further detail in Section 0. However, from the application's point of view, the operating system is the entire AUTOSAR stack, comprising all modules including and below the AUTOSAR runtime environment (RTE). The AUTOSAR stack is very complex software with several thousand configuration parameters and functionality that exceeds by far the typical necessities for the safety mechanism that is implemented in the system. It is thus neither technically nor commercially feasible to develop the entire AUTOSAR basic software (BSW) according to the highest safety integrity level which is required for the entire system.

Instead, we extend the partitioning approach to the AUTOSAR stack itself. The stack is split into two parts:

- The operating system, which is assumed to provide the context-specific memory protection and to have been developed according to the highest integrity level
- The AUTOSAR BSW, which is encapsulated in a separate memory partition with limited rights and cannot interfere with the remaining parts of the system.

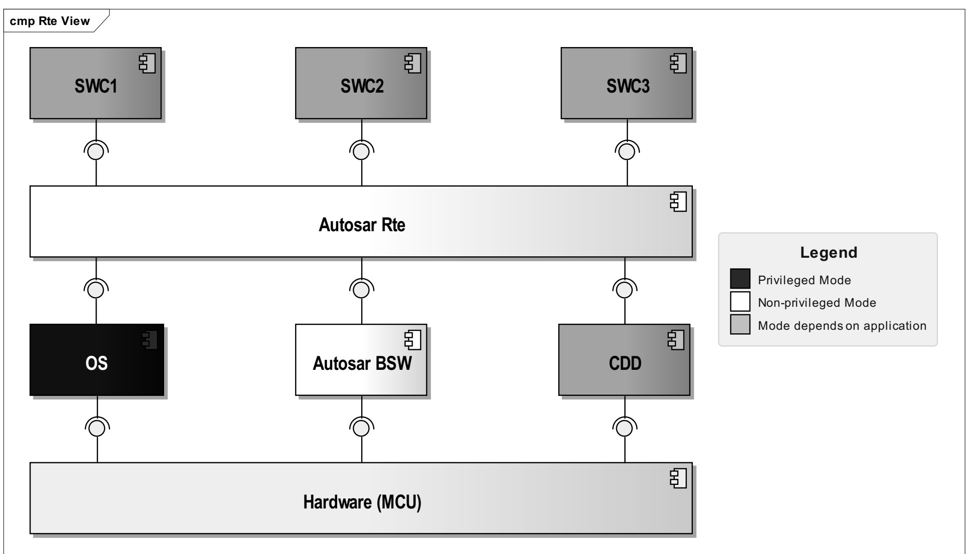

Figure 2: Architectural overview of AUTOSAR modules and software components

This approach allows the assignment of only a few safety-related requirements onto the operating system, namely the provision of safe task and interrupt scheduling and a safe memory protection mechanism, while for all other components usually no safety requirements apply. Therefore, these modules can be developed with usual QM approaches.

Timing and communication mechanisms are not part of the partitioning concept. If the safety concept requires time and control flow monitoring or protection of the communication, they must be part of the application. This can be achieved with standard modules provided by the AUTOSAR stack: the watchdog manager interface provides mechanisms for time and control flow monitoring in conjunction with an external watchdog, while the end-to-end protection library protects the communication channels.

The architecture of such a partitioned AUTOSAR stack is shown in Figure 2. In contrast to Figure 1, the entire basic software including the MCAL is contained in the component *Autosar BSW*. In this scenario, only the operating system runs in the privileged mode, while all other components run in the non-privileged user-mode or have at least the possibility to do so. In addition to three software components, Figure 2 shows also a complex device driver (CDD). All these components can run in the non-privileged mode, although this is the decision taken in the software architecture.

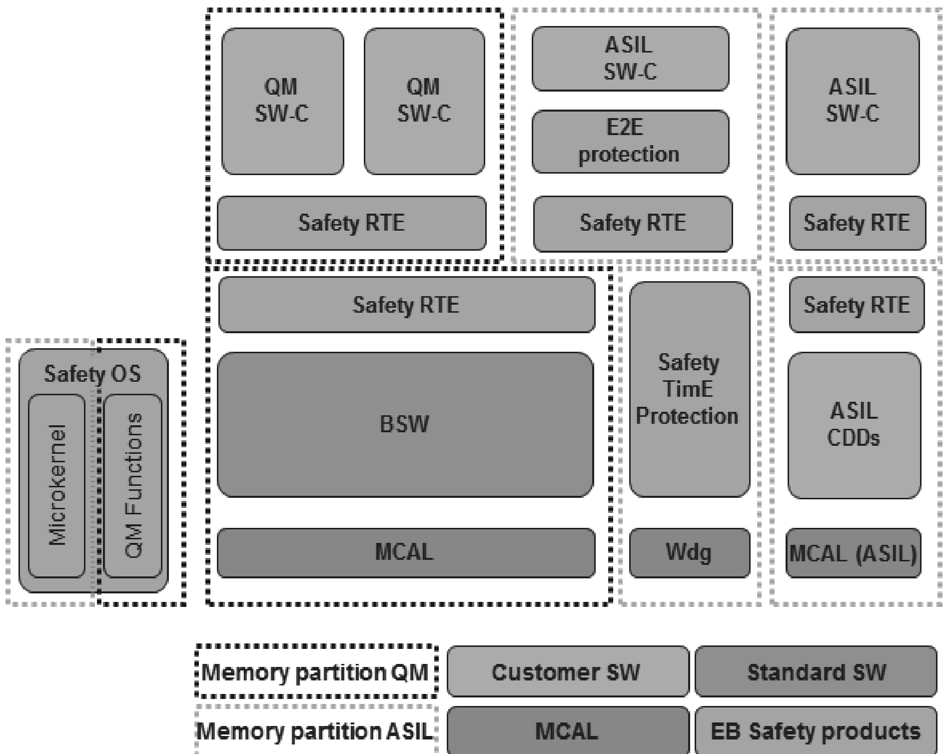

Figure 3: Safety architecture for an AUTOSAR ECU

The approach of putting the AUTOSAR BSW into a separate partition running with reduced rights has the advantage of drastically reducing the code that has to be developed, analyzed and verified according to the highest safety integrity level. Freedom from interference on the data level between the software components and the AUTOSAR BSW is provided by the memory protection which is implemented in the operating system.

Two prerequisites are necessary for this setup:

- Since the MCAL is placed in the AUTOSAR BSW component, which runs in the non-privileged mode, the CPU must grant access to the peripherals in this mode. However, modern CPUs for safety applications have considered this and allow such an operation. Otherwise, the MCAL would have to be adapted to request the elevated rights from the operating system before accessing the peripherals.
- Timing protection and control flow monitoring are still part of the application. The interface to an intelligent watchdog driver can be used by the application.

An exemplary safety architecture detailing a partitioned AUTOSAR ECU including a partitioned RTE, time and execution protection and end-to-end communication protection is shown in Figure 3.

The microkernel concept

A full implementation of the AUTOSAR Os module would provide memory protection for tasks and interrupt service routines (ISR) to the standards required for safety-critical software. However, the OS module is a very large piece of software that would be costly and time-consuming to implement to the high standards required for safety-critical software.

An alternative solution of using a standard OS with additional protection measures has been proposed. The protection features required to ensure freedom-from-interference between tasks and ISRs are listed in Table 1, along with the external protection measures that would be necessary when using a standard OS.

However, it has already been observed [Ha11] that in the presence of an operating system it is not sufficient merely to program the MPU to permit access to the critical data during the computation that uses it. This is because any software that executes while the access is permitted, including interrupt service routines and tasks of higher priority, could modify the critical data without detection.

The design of the standard AUTOSAR Os specifies re-entrant functions and allows the execution of ISR routines while the MPU is programmed. The Safety Os however doesn't allow any interruptions or parallel execution while access to the MPU is allowed, leading to the differences listed in Table 1.

Table 1: Safety properties of the standard AUTOSAR Os and the Safety OS

	AUTOSAR Os	**Safety OS**
Setting the MPU	Yes	Yes
Correctly setting the MPU	No, requires additional measures, e.g. regular value checks	Yes
Safe handling of critical registers (PC, SP, μP mode)	No, requires additional measures, e.g. control flow monitoring	Yes
Safe handling of critical memory regions (stacks, context safe areas, Os states)	No, requires additional measures, e.g. control flow monitoring	Yes
Safe task switch, pre-emption and resume	No, requires additional measures, e.g. control flow monitoring	Yes
Safe interrupt/exception handling	No	Yes
Correct task/interrupt priority based scheduling	No	Yes
Safe deactivation of faulty subsystems	Yes, reset into error mode	Yes, reset into error mode
Safe resource management	No	Yes
Safe event management	No	Yes

Furthermore, there are failure modes in an operating system that can have very subtle effects on the integrity of the tasks that it controls. These effects would be extremely difficult or impossible to detect using the proposed measures. Even the provision of a high-integrity memory protection driver, loosely coupled to a standard operating system, would not provide sufficient protection. This is because during the critical computation the data is not confined to memory but can reside in the processor's registers, which are fully under the control of the operating system.

When other factors such as the integrity of the stack and the critical section control are taken into account, it rapidly becomes clear that the operating system itself must provide the protection.

Fortunately, the OS has many features that could easily be separated from the core of the operating system using the same mechanism that provides the freedom from interference between tasks and ISRs. The minimum core of the OS that must be developed to the high standards for safety critical software is called a microkernel.

In principle, the microkernel should be the only software that runs in the privileged mode of the processor, and is therefore the only software that can modify the contents of the MPU. The main functionality of the ECU, including communication and I/O, along with OS features that provide time-triggered activation, run at a lower privilege level ("user mode") in which access to the MPU is forbidden by the processor. This software need not be developed to the same standards as the microkernel.[7]

The defining feature of the microkernel is that it uses a hardware feature called a *system call*, which is typically a vectored exception, to control transitions between tasks and ISRs and the microkernel. In addition to the system call, all other transitions into privileged mode, such as interrupts and processor exceptions, are handled by the microkernel in a similar way. Using this mechanism, it is not possible to switch from user-mode into privileged mode without simultaneously transferring control to the microkernel.

The microkernel manages all the executable objects in the system. The set of executable objects contains:

- Tasks

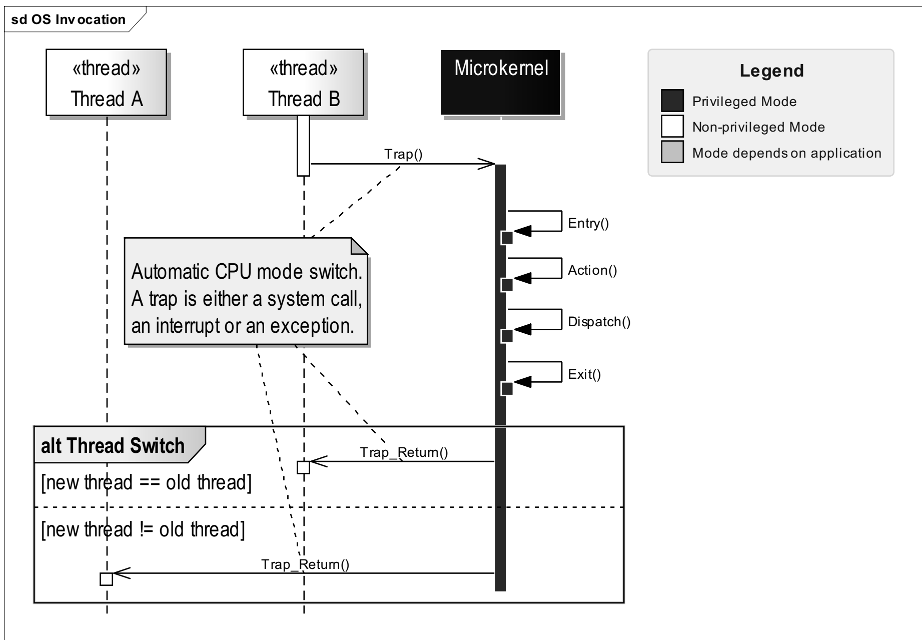

Figure 4: Control flow upon microkernel activation

- ISRs
- Hook functions
- Time-triggering services provided by the standard OS
- Idle loops
- Startup: the `main()` function.

These objects are all managed using a mechanism called a *thread*.

By means of the system call mechanism, it can be demonstrated that the behavior of a user-mode thread with respect to the data of another thread is not merely "acceptably safe", or that interference is "unlikely" or "improbable"; rather, it can be demonstrated that interference cannot occur.

The control flow through the microkernel is depicted in

Figure 4. The kernel function that is called depends on the type of interrupt or exception, and may result in a different thread becoming more eligible to use the CPU than the current thread. The change of thread is performed by the dispatcher, and includes reprogramming the MPU to the correct settings for the incoming thread.

This simplified control flow means that the microkernel can be made non-reentrant, thus further simplifying the design of the microkernel and thus the assurance of correctness.

This also means that the microkernel is only concerned with the execution of thread switches and programming the MCU. All other functions of a standard AUTOSAR Os are outside of the scope of the microkernel and considered QM functions.

Partitioning with an AUTOSAR operating system

From an AUTOSAR operating system point of view, SWCs are just applications using the OS interface, since the RTE mainly encapsulates the OS and BSW APIs within its own layer. For the OS, the SWCs are mapped to so-called *OS Applications*, as shown in Figure 5.

OS Applications are a container for tasks, ISRs and any other OS controlled objects that belong to a certain application. They are also the boundary for the memory partitioning: all objects belonging to the same OS Application can share common data.

Consequently, Figure 5 shows seven different memory partitions: three SWCs, a CDD, the AUTOSAR BSW, the QM part of the operating system and the microkernel itself (although running in the privileged mode, the kernel grants access only to its own data).

The configuration of the OS Application also decides on the CPU mode in which its tasks and ISRs are executed. Depending on an application-specific configuration

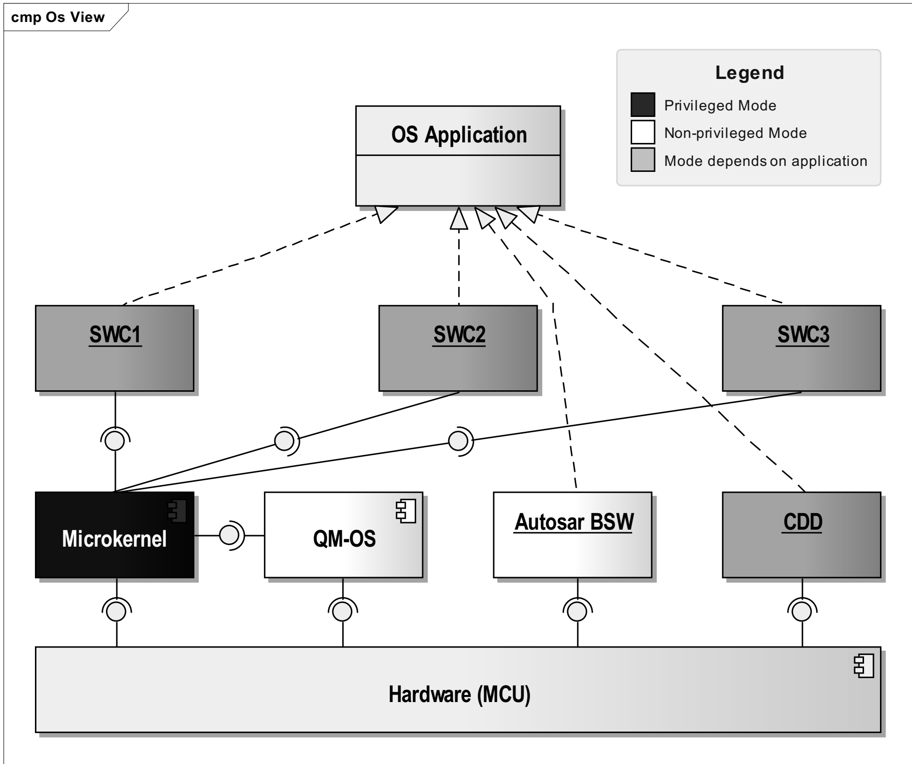

Figure 5: AUTOSAR stack configuration as seen from the OS module

parameter, all application objects either use the CPU's privileged mode or the non-privileged mode.

Running components in the non-privileged mode gives the highest protection against any kind of error: the MPU providing the foundations of the partitioning mechanism is protected against any change, only the microkernel is allowed to reconfigure the memory protection. The microkernel itself is only entered via defined access points – namely the interrupt and exception interface.

An important goal for the design of the Safety Os was to integrate it seamlessly and directly into the AUTOSAR architecture. Given AUTOSAR as a fixed standard this means that all requirements specified by AUTOSAR still need to be fulfilled. This could be achieved and starting with a standard AUTOSAR architecture the Safety Os can be added to an AUTOSAR Os without affecting any AUTOSAR requirement apart from the operating system itself.

The partitioning via software components (SWC) and their OS Application counterpart is a well-established technique in safety-related development. Similar approaches are known in aviation as the IMA architecture [RTCA05].

Implications for the development process

The development of a safety-related operating system introduces challenges into the development process. From the start, the development process was based strongly on Automotive SPICE in the HIS [HIS12] scope. The challenge was to extend the existing processes to cope with the additional requirements arising from safety-related development according to ISO 26262 and IEC 61508.

Since the projects are regularly assessed in project audits, it was decided to extend these audits by introducing essential safety aspects.

Two main lessons have been learnt during the development:

- The HIS scope of SPICE is not sufficient to support safety development
- Internal SPICE assessments have been updated with additional checklists in order to check method definitions and their usage. Such assessments can then be used as functional safety audits according to ISO 26262.

In order to support safety development the process scope has been extended from the HIS scope to cover more processes. In the following, we describe the development of the OS as a safety element out of context and focus then on process extensions which have been implemented to fulfil safety requirements.

Safety Element out of Context (SEooC)

The increasing complexity of modern software projects encourages the use of reusable software components (RSC) with clearly specified functionality. In avionics, the concept is well-established and part of DO-178C [RTCA11]. In ISO 26262, it is referred to as a *Safety Element out of Context* (SEooC). The key element of a SEooC is the set of assumptions that are applied during the development. The approach is shown in Figure 6.

The most important aspect is a clear description of the context in which the SEooC is used and the assumptions that are made. These assumptions define the scope of the SEooC and have impact on its requirements and its design.

Two simple rules have proved to be useful during the development of the OS component:

- The fewer assumptions exist, the clearer is the focus of the component.
- The more precise the assumptions are, the simpler is the implementation.

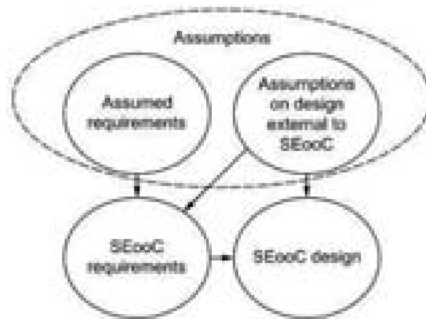

Figure 6: Relationship between assumptions and the development of a SEooC

These rules lead to a small but flexible product which can be used in many different environments. A small set of assumptions also simplifies the usage of the component in the final device. Since the scope of the SEooC is much clearer than with a large assumption set, the safety argument for the entire device becomes much more concise. In addition, this concept allows the re-use of existing safety mechanisms including any certifications that may already be available.

Extended Tracing

In SPICE conformant development processes, the tracing between requirements and their associated test cases is a well-established technique. In a safety-related development, further work products are created. A wide-spread technique to ensure consistency between the content of the various work products is to extend the dependencies between work products to tracing between the content of work products.

For example, to ensure that the correct functionality has been implemented, bi-lateral tracing from requirements through the software architecture and the software unit designs to the source code is already part of Automotive SPICE.

In safety-relevant projects, the number of documents that is required to fully specify and document the implementation as well as its verification can increase significantly: for example, a safety-case providing the safety argument, a hardware-software interface specification and a safety manual that details the correct and safe use of the software product are the most prominent work products.

The foundation of the content of these documents is the software safety requirements specification and each of these documents can contain the fulfilment of a requirement:

- Safety case: by an argument, e.g. a result of a safety analysis
- Hardware-software specification: e.g. does the hardware behave as documented?

- Safety manual: e.g. are all assumptions being taken into account by the software configuration?

As these documents can become an end-point for software safety requirements, from a tracing point of view they need to be treated just like a software unit design or the implementation itself. The tracing does not have to be bi-lateral as for example a safety manual also contains information that is not directly motivated by a software safety requirement.

The tracing model of Automotive SPICE needs to be extended to include such safety work products in order to construct an argument for the completeness of the fulfilment of software safety requirements. See

Figure 7 for an example of a tracing model.

The extension of the tracing model to include safety work products is especially important in SEooC development wherever the final context in which the software is expected to be used is assumed and documented.

Summary and Outlook

In this paper we presented a microkernel based approach for a partitioned AUTOSAR system. In this concept, only the microkernel runs in the CPU's privileged mode, providing freedom from interference in the spatial domain between the AUTOSAR basic software, the integrated software components and complex device drivers.

In order to reduce the size and complexity of the code running in the privileged mode

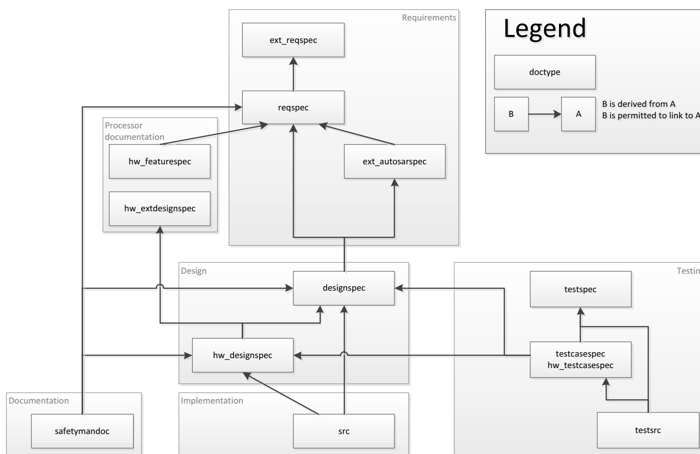

Figure 7: Tracing model of the Safety OS

even further, the operating system has been split into a safety-relevant part providing the basic scheduling features and the memory protection, and a part containing the non-safety relevant functionality which runs in its own non-privileged partition. This reduces to a minimum the size and complexity of the software that needs to be developed according to the highest safety integrity level.

In addition, the development processes of the standard product development have been extended to support the creation and the tracing of the additional work products as required by ISO 26262. They are executed for the development of the OS as a Safety Element out of Context (SEooC), allowing for the reuse of the component in various different environments and systems.The development of the Safety Os including all safety related tasks is already finished and it is now going through a certification process with an external party to have a certification according to ISO 26262 (up to ASIL-D) and to IEC 61508 (up to SIL-3).

With the emergence of more and more powerful multi-core CPUs, partitioning will become even more relevant in the future. The additional computational power will be used to integrate software from various different suppliers often with different quality levels, requiring strict supervision and encapsulation of the supplied components. In addition, domain controllers combining several different functions of an entire car domain are currently being introduced. These devices integrate a multitude of software components, where partitioning is a key concept to ensure the non-interference between the modules. With the development of highly integrated many-core devices this trend will accelerate further.

Bibliography

[ASR12] AUTOSAR, http://www.autosar.org.

[ASPICE10] Automotive SPICE® Process Assessment Model v2.5, May 2010.

[ISO11] ISO 26262, Road vehicles - Functional safety, ISO copyright office, 2011-11-15.

[IEC10] ISO/IEC 61508, Functional safety of electrical/electronic/programmable electronic safety-related Systems, IEC central office, 2010.

[MIRA04] MIRA Ltd., MISRA-C:2004 - Guidelines for the use of the C language in critical systems, ISBN 0952415623, October 2004.

[RTCA11] DO-178C, Software Considerations in Airborne Systems and Equipment Certification, RTCA Inc., 2011-12-13.

[RTCA05] DO-297, Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations, RTCA Inc., 2005-08-11.

[Ha11] Haworth, David: An Autosar-compatible microkernel. In Herausforderungen durch Echtzeitbetrieb. Berlin: Springer-Verlag, November 2011

[HIS12] Hersteller-Initiative Software, http://www.automotive-his.de/