Classic and Adaptive AUTOSAR in MILS terms

Holger Blasum and Sergey Tverdyshev, SYSGO AG

Abstract: AUTOSAR (AUTomotive Open System ARchitecture) supports modular software development for automotive software. MILS (Multiple Independent Levels of Safety and Security) also is also inspired from modular systems such as integrated modular avionics. There are differences though: automotive electronic control units are under much more cost pressure than their avionics counterparts, and Classic AUTOSAR was targeting rather simple systems, with an initial focus on runnables that are compiled together, and we will highlight the difference as well as the evolution of AUTOSAR Adaptive that is much closer to the avionic model. On the other hand, AUTOSAR has a very good standardization momentum, resulting in hundreds of available documents, whereas the smaller MILS community has been less effusive. We map the AUTOSAR standards to MILS, to learn about (1) how well MILS systems can be used for AUTOSAR and vice-versa and (2) what other aspects the communities could mutually learn from.

1 Introduction

AUTOSAR is a standard for the modular development of automotive software. AUTOSAR aims at software reuse and platform independence by decoupling the development of applications from the development of platform drivers (basic software). Our motivation is to understand to what extent MILS systems can be used for implementing AUTOSAR systems, and, the other way round, whether MILS can learn from AUTOSAR. In related work, MILS has been mapped to and used for other standards such as IEC 61508, IEC 62443, EDSA, ISO 62443, ISO 26262, J3061, DO-178, Common Criteria [1] [2] [3]. Concepts for implementing AUTOSAR have been e.g. discussed for Linux [4] and for hypervisors [5] [6]. It has been described how to use, from an implementation perspective, MILS for automotive in previous MILS workshops [7] [8]. However, to the best of our knowledge there has not yet been a direct juxtaposition of how MILS provides AUTOSAR concepts so far.

The structure is as follows: we describe AUTOSAR roles (Section 2); then AUTOSAR Classic, including mixed-criticality concepts and mapping to MILS (Section 3); and AUTOSAR Adaptive, including mixed-criticality concepts and mapping to MILS (Section 4); and we discuss what the communities could learn from each other (Section 5).

2 AUTOSAR roles

AUTOSAR allows to decouple, amongst others, the following roles (roles mostly implicit in AUTOSAR documentation, the following are based on [9]):

- Function engineer: engineers functions need for car operations (e.g., chassis control)
- Software component developer: the software component developer writes application software, delivered as source or object code.
- Driver developer: the driver developer develops low-level drivers for microcontroller-independent hardware components, called “basic software” in AUTOSAR, e.g. CAN driver.
- ECU integrator: the person who integrates the drivers, application software and hardware, for an ECU (electronic control unit). Here the term ECU is used to describe the physical box

containing the electronics which may contain one or several processing units with one or several AUTOSAR Stack instances running ([10] Section 3.1.1).

2.1 Mapping AUTOSAR roles to MILS

AUTOSAR	MILS
Function engineer	System-level engineer / function engineer
Software component developer	Application developer
Driver developer	Driver developer
ECU integrator	System integrator

Table 1: Mapping AUTOSAR roles to MILS

Table 1 shows that AUTOSAR roles can be straightforwardly mapped to MILS roles. The MILS roles are based on ARINC 653 and the MILS architecture document [11]. The main common point is that both AUTOSAR and MILS decouple the developers of low-level components (software components / applications and drivers in Table 1) from the unit/system integration and function engineering.

3 AUTOSAR Classic

The overall workflow in AUTOSAR classic is that tools called RTE generators and BSW generators compile a binary out of the following three ingredients (1) a set of software component source code (e.g., C language) or binary files or (2) BSW component source code (e.g., C language) or binary files and a (3) description of the ECU in XML format.

3.1 Specifying an AUTOSAR Classic system

A runnable provides an entry point for the scheduler. A runnable typically is a short control routine (often 1-5 milliseconds) that is repeatedly (often: cyclically) scheduled.

Software components provide and/or require interfaces for services. Software components are connected to each other through a Virtual Function Bus (VFB), which represents the *specification* of software component independencies, to fulfill architectural responsibilities of runnables, see Figure 1.

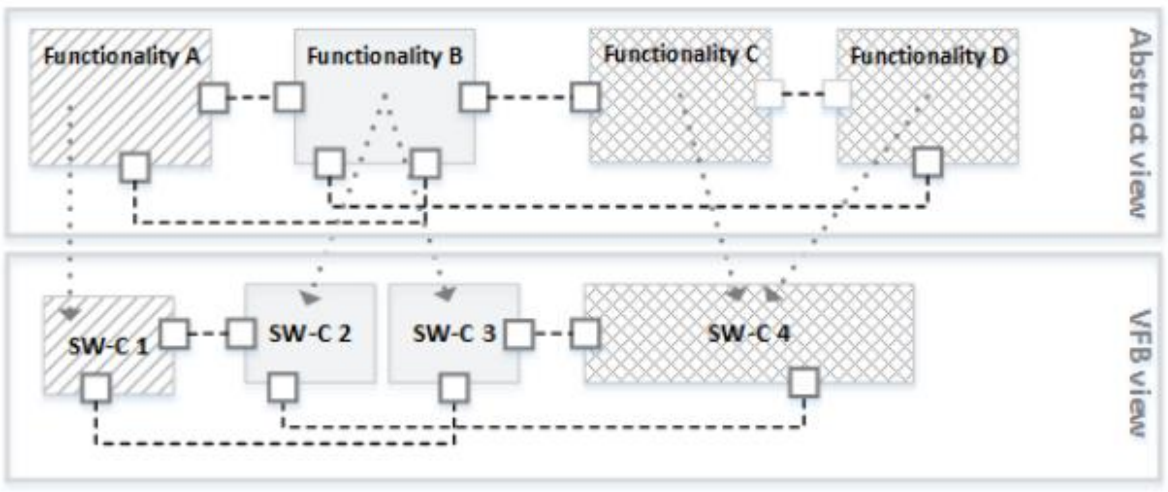

Figure 1: Functionality to software components mapping, from [12] Methodology, Section 2.1.2.2

Workflow-wise, at the very highest level, the function engineer defines the functions. Next, the integrator creates the design, including the runnables, events, interrunnable variables, etc, and groups the runnables into software components. After this step is complete, the contract header files are created and the software component developer develops the runnable implementations ([12] TR_METH_01060).

3.2 Compiling an AUTOSAR Classic system

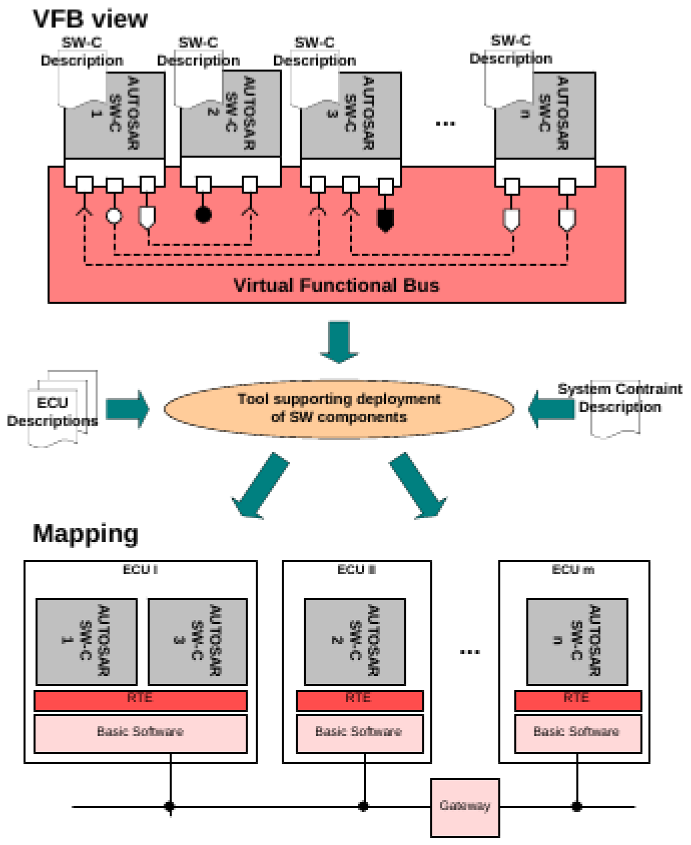

Figure 2: Mapping the VFB to ECUs, from [13], Section 2.1.1

The ECU integrator now uses the source code and/or binaries of the atomic software components and drivers to generate a basic software (BSW), including an RTOS, and an RTE (Run-Time Engine) that satisfies the specifications, see Figure 2. The unit of scheduling in the RTOS is the task [14].

AUTOSAR Classic does allow for some kind of application separation: As the RTE and BSW are compiled statically, it is possible to implement space separation between applications by appropriate management of addresses at the compilation and linking level: The requirement RS_SWCT_00210 in [15] specifies that software components shall be protected from illegal access (change of inner variables), that is only their interfaces are used. Alternatively, it is also possible to distribute software between ECUs: With today's multicore systems, assigning entire CPUs to applications is an alternative to scheduling-based separation in time. AUTOSAR Classic allows assigning applications to different ECUs. That is, the ECU integrator could set up the AUTOSAR system so that untrusted applications can run on different ECUs. As each ECU gets its own CPU and RTE, that would allow pursuing a resource separation approach. While not all RTE compilers do support allocation to different CPUs, in principle AUTOSAR Classic would support to specify a separation of applications by "ComponentClustering" ([16] Section 5.1.4.1) and "ComponentSeparation" ([16] Section 5.1.4.2) constraints. Conversely, a software component can be labeled as "atomic" to indicate that its runnables cannot be distributed among CPUs. Similarly communication is specified in a communication matrix, and it is possible to rule out unwanted signal paths, e.g. there is a ForbiddenSignalPath ([16] Section 5.2.2.2) and SeparateSignalPath ([16] Section 5.2.2.2).

3.3 Mixed-criticality in AUTOSAR Classic functional safety methodology

3.3.1 Mixed-criticality in AUTOSAR Classic traces its origins to ISO 26262

The automotive functional safety norm ISO 26262 [17] has defined separation in space and time: Separation in space can be realized by memory separation and use of CPU supervisor mode, and separation in time is avoiding deadlocks, livelocks, incorrect allocation of execution time and incorrect synchronization between elements. In AUTOSAR Classic, separation in space and time are reflected in the AUTOSAR Functional Safety Measures document [18]. In detail, ([18] Section 2.1.6) points to ISO 26262 Part 6 Sections 7.4.11, 7.4.12 and appendices D.2.1and D2.3, and Part 9 Section 6.2, 6.4.4, and 6.4.5 for memory partitioning, i.e. separation in space. For separation in time, ([18] Section 2.2.6) points to ISO 26262 Part 6 Section 7.4.14, Table 4 and appendices D2.1 and D2.2. The identifications of mixed-criticality aspects in ISO 26262 made by the AUTOSAR Functional Safety Measures document are consistent with our own independently made observations [1].

3.3.2 Separation in space

Memory partitioning is defined as execution of applications in separate memory regions in [18]. That is a memory partition consists of a set of memory regions plus the application(s) that have access to it. For memory protection, the use of hardware mechanisms is recommended. According to [19] Section 3 "an implementer of a SWC shall be aware whether there is a memory protection (e.g. by ECC/EDC/MMU/MPU) supported by the underlying processor architecture in order to correctly implement the handling of safety related data." Moreover, [19] gives an example of a mixed-criticality system implemented by partitioning.

In [20] Section 5.3.2, the MMU/MPU is used as interference protection against COM / RTE module interference and combined with proprietary OS safety extensions to grant freedom from interference also on BSW.

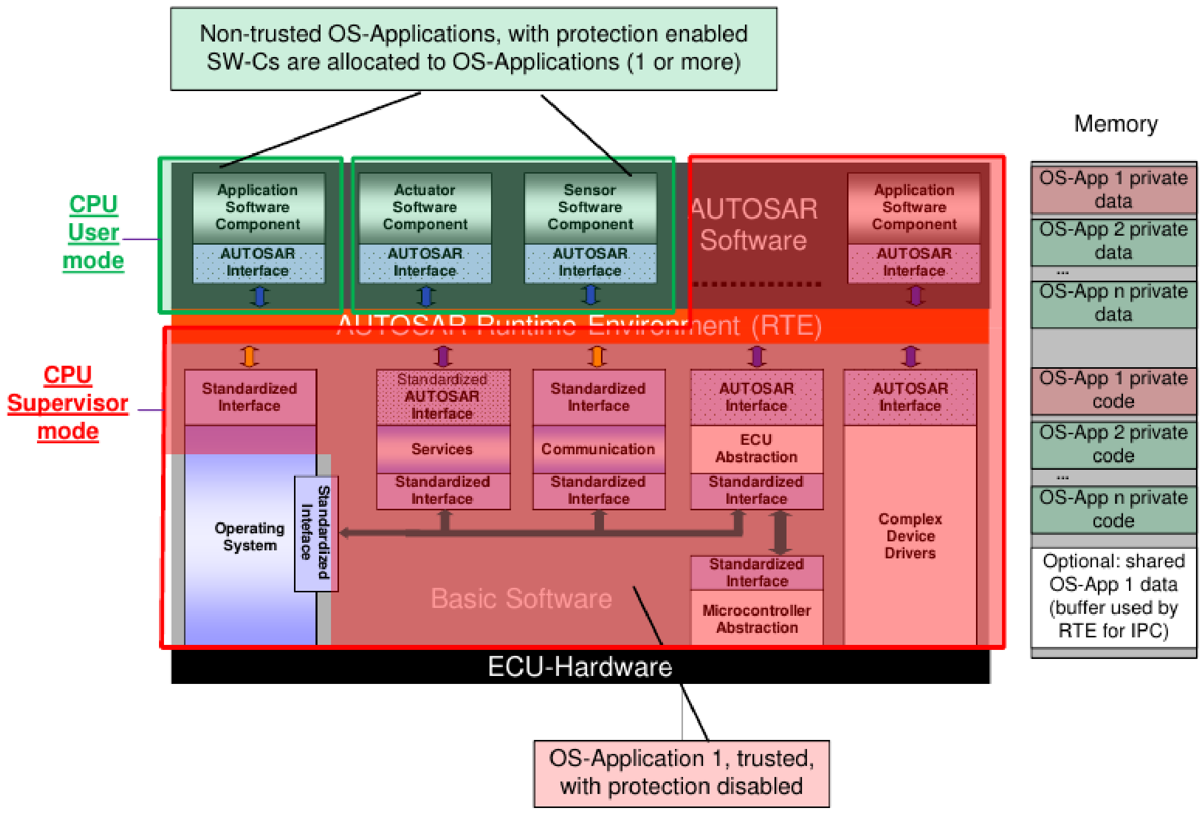

Figure 3: Combination of memory protection and CPU supervisor mode for protection, [21] Section 1.1.6

As shown in Figure 3, memory production is usually combined with the use of CPU user mode / CPU supervisor mode.

3.3.3 Separation in time

[18], Section 2.2.2 proposes timing protection using the operating system or, alternatively temporal program flow monitoring using the watchdog manager. For interrupts, [18] discusses execution time protection, locking time protection, inter-arrival time protection.

3.3.4 Current limitations

In [21] it is observed that when such a grouping of runnables to OS applications is done, in the (classic) AUTOSAR standard “no communication mechanism between OS-Applications offered. The OS itself does not provide the communication between OS-Applications - instead, OS clearly delegates the communication between partitions (i.e. basic techniques for transferring data between protected memory regions) to RTE. RTE assumes its role, but does not provide these mechanisms yet. Therefore, inter OS-Application communication (i.e. communication between different OS-Applications within the same ECU) is the major missing functionality.”

3.4 Mapping AUTOSAR Classic to MILS

	AUTOSAR Classic	MILS
Production model	Multi-role	Multi-role
Partition	A partition is a set of applications in AUTOSAR Classic. The functional safety concept adds the concept of protected memory regions (Section 3.3.2). On MPU or MMU.	A partition is a set of applications and memory regions. Typically uses address spaces, but it can be realized on MPU too.
Interfaces	Standardized interfaces (XML) for driver, hardware abstraction layer.	Standardized interface for run-time e.g. POSIX PSE51.
CPU pinning	Can be realized by assigning runnables to components (see Section 3.2). Not supported by all RTE generation engines.	CPU affinity
Scheduling	Runnables, tasks. Execution time protection in functional safety concept (Section 3.3.3).	Threads, tasks, time partitioning.
Address spaces	Typically not available	Typically available
Communications	Ports at run-time of components, no framework for communications across address spaces yet (Section 3.3.4).	Separation kernels usually provide communication mechanisms

Table 2: Mapping AUTOSAR Classic to MILS

Table 2 shows a mapping of AUTOSAR Classic to MILS. We see that application separation in AUTOSAR Classic originally assumed that all runnables are compiled and controlled by the ECU integrator, and that separation is enforced by the RTE generation tool. Such an approach only works for static systems where all components are trusted. It is also possible to distribute the system among different ECUs, however not all RTE generation engines support that, moreover different ECUs means hardware overhead. If a strong separation on the same ECU is needed, the AUTOSAR Classic functional safety concept adds protected memory regions (memory partitions) and execution time protection. These concepts can be very well supplied by a MILS platform with multiple partitions. The MILS platform can run an RTE as guest in each partition, in order to provide the familiar AUTOSAR Classic interface (e.g. AUTOSAR Classic ports and scheduling of runnables within a partition). In case of communication between memory partitions, one has to implement communication between runnables by communication protocols by means provided by the separation kernel. Depending on the hardware used for the AUTOSAR Classic system, a MILS separation kernel may need to have MPU support.

4 AUTOSAR Adaptive

Technology drivers for AUTOSAR adaptive have been the availability of multicore CPUs, FPGAs, general-purpose GPUs, the increasing use of Ethernet and the emergence of networks on a chip [22]. One motivation for AUTOSAR Adaptive is that it shall be possible to download software on an ECU without configuring and generating the BSW and RTE again. This requires an approach that decouples the ECU configuration from the software components to be deployed ([23] Section 1.1).

Workflow-wise AUTOSAR Adaptive supports a distributed workflow and the development of SEooC (Safety Element out of Context), where the safety requirements of the integrated system are not fully known at development time.

Other features are that the languages C++ and C are supported, and it possible to have not only local services but also to have remote services (service-oriented architecture). ECUs running AUTOSAR Adaptive can interact with ECUs running AUTOSAR Classic as well as non-AUTOSAR systems.

4.1 AUTOSAR Adaptive as a layered architecture

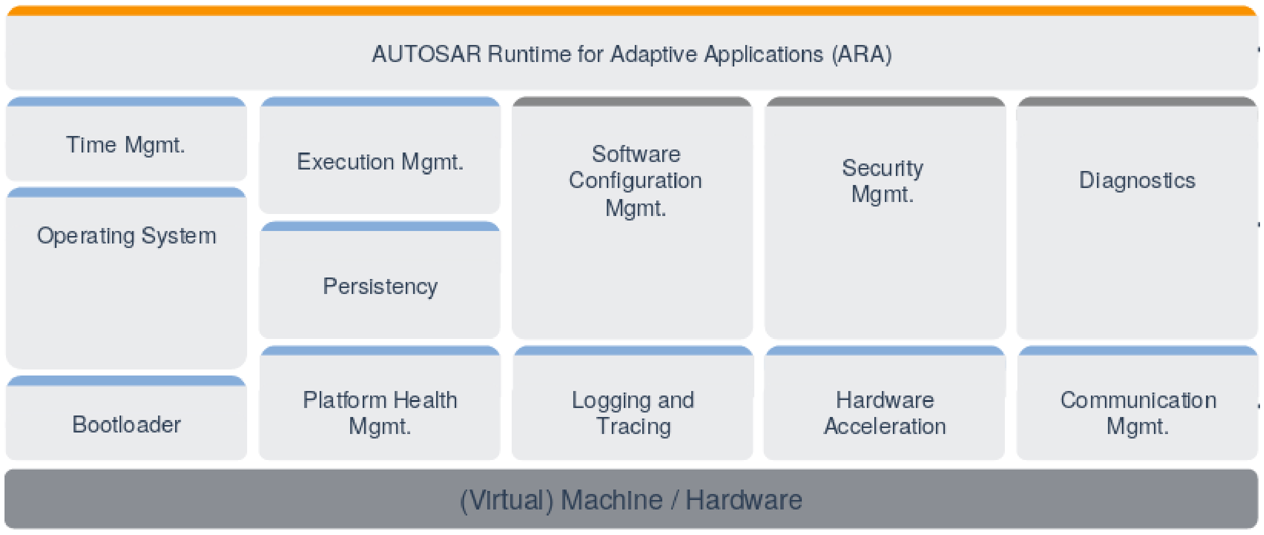

Figure 4: AUTOSAR Adaptive conceptual overview, from [24]

Figure 4 shows the layers of that are part of an AUTOSAR Adaptive system. On the top layer of the figure there is the AUTOSAR Runtime for Adaptive Applications (ARA), which provides APIs for the services shown in the middle-layer boxes, which for example include communications. The ARA API to Adaptive Applications is based on POSIX PSE51, but allows for extensions. Services marked with a gray tab are services provided via the network, services marked with a blue tab are services provided by local libraries. Services can communicate with each other via interprocess communication.

At the bottom of the layered architecture (dark gray box) there is the machine, which explicitly not only can be realized by hardware, but also by a virtual machine. Topmost (not shown in Figure 4), on top of the ARA there are Adaptive Applications, which use the services provided by the ARA. Adaptive Applications consist of one or more processes, in turn each process consists of one or more threads [22].

4.2 Mixed-criticality concepts in AUTOSAR Adaptive

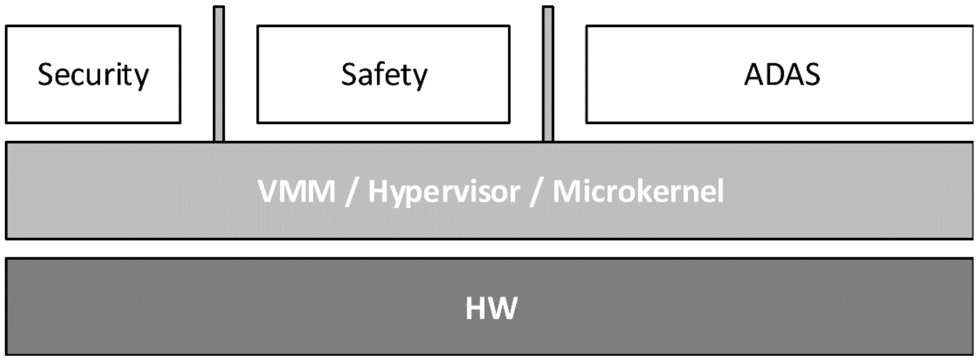

Figure 5: Mixed-criticality example in AUTOSAR Adaptive, from [23] Section 2.3.3.1

AUTOSAR Adaptive has memory partitioning "built-in" from the beginning. Figure 5 shows an example deployment of AUTOSAR Adaptive with support for memory partitions on top of a hypervisor, described in [23]. In the example the ADAS (advanced driver assistance system) is the normal operation partition, the partition labeled "safety" serves as a fallback, and the partition labeled "security" provides security functionality (e.g. firewall).

Execution management allows to start processes in a particular order at initialization (like e.g. Linux init) and to maintain resource groups: a resource group consists of several adaptive applications which are assigned a maximum quota of memory / CPU time. In terms of requirements the resource management is put onto the operating system in [25], RS_OSI_00201 "*The Operating System shall provide mechanisms for system memory budgeting.*" and RS_OSI_00202 "*The Operating System shall provide mechanisms for CPU time budgeting*".

AUTOSAR Adaptive subsumes under "planned dynamics" such mechanisms to restrict dynamism [23]. Planned dynamics also include pre-determination of service discovery process, restriction of dynamic memory allocation to startup phase only, fair scheduling policy instead of priority-based scheduling, fixed allocation of processes to CPU cores, access to pre-existing files in the file-system only, constraints for API usage by applications and execution of authenticated code only. Applications carry their own manifest file to indicate resource needs.

4.3 Mapping AUTOSAR Adaptive to MILS

	AUTOSAR Adaptive	MILS
Production model	Multi-role	Multi-role
Partition	Memory partition; planned dynamics (e.g. resource groups)	Partition + resource assignment
Interfaces	PSE51 + ARA	Standardized interface for run-time: Subset of Posix (PSE51-like) is being standardized by the Open Group
CPU pinning	Yes	CPU affinity
Scheduling	Threads, Applications	Threads, Tasks, Time partitioning
Address spaces	Yes	Yes

Table 3: Mapping AUTOSAR Adaptive to MILS

Table 3 shows a mapping of AUTOSAR Adaptive to MILS. Again, MILS partitions are a natural representation of memory partitions. “Planned dynamics” can be implemented on a MILS system by specifying these limits in each the configuration of each partition. We see that for interfaces AUTOSAR Adaptive proposed PSE51, which is also being standardized by the Open Group for MILS systems. Both approaches allow CPU pinning of applications. Architecturally, both approaches put different applications into different address spaces.

5 Results and discussion

We have the following results: especially when separation in space and time and good predictability as stipulated by ISO 26262 are important, MILS is a good choice to implement AUTOSAR Classic. Stronger separation requirements and predictability such as memory partitions are built into AUTOSAR Adaptive, and MILS appears to be generally a good choice to implement AUTOSAR Adaptive. Still, judging from the proposed scheduling mechanisms, many AUTOSAR Adaptive deployments may have less strict timing requirements than time partitioning provided by MILS systems, especially where CPU pinning is used instead of traditional cyclical time windows. Modern MILS separation kernels also support CPU pinning.

Some of the resource allocation and planned dynamics concepts of AUTOSAR Adaptive are also found in Linux systems (e.g. cgroups, [4]). However, in mixed-criticality systems the ISO 26262 assurance level (ASIL) of the foundation must be equal or larger than the ASIL of the most critical application, and thus a small hypervisor is probably better suited for the task [6], although especially in paravirtualization the overhead as to be evaluated carefully, this scenario is expected to improve with automotive CPUs that have full virtualization support [5].

Currently, MILS providers and users are starting a modular protection profile at the Common Criteria Users Forum [26]. AUTOSAR has more than a decade of history and produced a panacea of specification documents as well as templates that describe concepts that are also found in MILS systems, but not yet standardized there. Therefore some AUTOSAR work could be inspiring to MILS providers and users, e.g. the rich set of XML templates, ECU managers distributed among multiple cores ([27] Section 2.4.4), or different trust-levels within the BSW ([27] Section 3.2).

Coming from security, MILS already has worked out threat analyses and models in the form of Common Criteria protection profiles. AUTOSAR has included many security-for-safety concepts implicitly under the name of functional safety, without mentioning security explicitly. It might be worth to explore how the upcoming MILS protection profile can be applied to AUTOSAR Adaptive systems.

Acknowledgment

This work has received partial funding from the European Union’s Horizon H2020 research and innovation program under grant agreement No 731456 (certMILS, https://certmils.eu/).

Bibliography

[1] S. Nordhoff and H. Blasum, "Ease Standard Compliance by Technical Means via MILS," 2017. [Online]. Available: https://doi.org/10.5281/zenodo.571175.

[2] S. Tverdyshev, "Security-Zertifizierung im IoT-Kontext: Effiziente Evaluierung durch komponentenbasiertes Software-Design," 2017. [Online].

[3] J. E. Rico, M. Banon, A. Ortega, R. Hametner, H. Blasum and M. Hager, "certMILS - D1.3 Compositional security certification methodology," [Online]. Available: http://www.certmils.eu/.

[4] M. Massoud, *Evaluation of an Adaptive AUTOSAR System in Context of Functional Safety Environments,* 2017.

[5] D. Reinhardt and G. Morgan, "An embedded hypervisor for safety-relevant automotive E/E-systems," June 2014. [Online]. Available: https://doi.org/10.1109/SIES.2014.6871203.

[6] C. Gouma, "Autonomous Driving needs Safety & Security," 2018. [Online]. Available: https://www.sysgo.com/fileadmin/user_upload/www.sysgo.com/redaktion/downloads/pdf/presentations/Autonomous_Driving_Needs_Safety_and_Security_EW18.pdf.

[7] D. Adam, C. Rolfes, S. Tverdyshev and T. Sandmann, "Two Architecture Approaches for MILS Systems in Mobility Domains (Automobile, Railway and Avionik)," 20 Jan 2015. [Online]. Available: https://doi.org/10.5281/zenodo.47991.

[8] A. Much, R. Grave, R. Leibinger, M. Böhner and E. Waitz, "Current Trends and Solutions in Securing Automotive Software," 2017. [Online]. Available: https://zenodo.org/record/571167.

[9] AUTOSAR GbR, "AUTOSAR TR TimingAnalysis," 2017. [Online]. Available: https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_TR_TimingAnalysis.pdf.

[10] AUTOSAR GbR, "AUTOSAR TPS ECUResourceTemplate," 2017. [Online]. Available: https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_TPS_ECUResourceTemplate.pdf.

[1 S. Tverdyshev, H. Blasum, B. Langenstein, J. Maebe, B. De Sutter, B. Leconte, B. Triquet, K. Müller, M.

1] Paulitsch, A. Söding-Freiherr von Blomberg and A. Tillequin, "MILS Architecture," 2014. [Online]. Available: https://doi.org/10.5281/zenodo.45164.

[1 AUTOSAR GbR, "AUTOSAR TR Methodology," 2017. [Online]. Available:
2] https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_TR_Methodology.pdf.

[1 AUTOSAR GbR, "AUTOSAR TechnicalOverview," 2008. [Online]. Available:
3] https://www.autosar.org/fileadmin/user_upload/standards/classic/3-1/AUTOSAR_TechnicalOverview.pdf.

[1 AUTOSAR GbR, "AUTOSAR SWS RTE," 2017. [Online]. Available:
4] https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_SWS_RTE.pdf.

[1 AUTOSAR GbR, "AUTOSAR RS SoftwareComponentTemplate," 2017. [Online]. Available:
5] https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_RS_SoftwareComponentTemplate.pdf.

[1 AUTOSAR GbR, "AUTOSAR TPS SystemTemplate," 2017. [Online]. Available:
6] https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_TPS_SystemTemplate.pdf.

[1 ISO/TC 22, Road vehicles, Subcommittee SC 3, Electrical and electronic equipment, "ISO 26262:2011
7] Road vehicles - Functional Safety," 2011. [Online]. Available: http://www.iso.org/.

[1 AUTOSAR GbR, "AUTOSAR EXP FunctionalSafetyMeasures," 2017. [Online]. Available:
8] https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_EXP_FunctionalSafetyMeasures.pdf.

[1 AUTOSAR GbR, "AUTOSAR TPS SafetyExtensions," 2017. [Online]. Available:
9] https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_TPS_SafetyExtensions.pdf.

[2 AUTOSAR GbR, "AUTOSAR EXP SafetyUseCase," 2017. [Online]. Available:
0] https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_EXP_SafetyUseCase.pdf.

[2 AUTOSAR GbR, "AUTOSAR TR SafetyConceptStatusReport," 2017. [Online]. Available:
1] https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_TR_SafetyConceptStatusReport.pdf.

[2 AUTOSAR GbR, "AUTOSAR_EXP_PlatformDesign.pdf," 2018. [Online]. Available:
2] https://www.autosar.org/fileadmin/Releases_TEMP/Adaptive_Platform_18-03/General.zip.

[2 AUTOSAR GbR, "AUTOSAR_EXP_SafetyOverview.pdf," 2018. [Online]. Available:
3] https://www.autosar.org/fileadmin/Releases_TEMP/Adaptive_Platform_18-03/General.zip.

[2 AUTOSAR Web Team, "AUTOSAR Introduction," 2017. [Online]. Available:

4] https://pdfs.semanticscholar.org/presentation/bab9/5dd0cbe6f70cf06aadd5a7100fd8abedfc6e.pdf.

[2 AUTOSAR GbR, "Requirements on Operating System Interface: AUTOSAR AP," 2017. [Online].
5] Available: https://www.autosar.org/fileadmin/user_upload/standards/adaptive/17-03/AUTOSAR_RS_OperatingSystemInterface.pdf.

[2 Common Criteria Users Forum (CCUF), "The Common Criteria Users Forum," 2018. [Online]. Available:
6] http://www.ccusersforum.org/.

[2 AUTOSAR GbR, "AUTOSAR EXP BSWDistributionGuide," 2017. [Online]. Available:
7] https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_EXP_BSWDistributionGuide.pdf.