Methods and tools for reducing certification costs of mixed-criticality applications on multi-core platforms: the RECOMP approach

Paul Pop[1], Leonidas Tsiopoulos[2], Sebastian Voss[3], Oscar Slotosch[4], Christoph Ficek[5], Ulrik Nyman[6], Alejandra Ruiz Lopez[7]

Abstract—**The current trends in embedded systems are driving the integration of more functions in one processing element. At the same time, multicore architectures are increasingly used to improve performance, reduce costs, power consumption and size. Safety-critical applications are developed according to certification standards, which, depending on the criticality level, dictate the development processes and certification procedures that have to be followed. Functions of different criticalities have to be separated, both temporally and spatially, otherwise a low-criticality function can corrupt a high-criticality function. Separation is difficult to achieve in multicores, which leads to very high certification costs: all the functions on a multicore have to be developed as if they are of highest criticality. The RECOMP (Reduced certification cost for trusted multicore platforms) project aims at reducing the certification costs of mixed-criticality applications on multi-core platforms, and addresses several industries, including industrial automation, automotive and avionics. The reduction in certification costs is achieved through (1) the use of separation solutions among mixed-criticality functions, through a combination of communication infrastructure, hardware, operating systems and middleware, methods and tools and component models, such that the required independence is achieved; (2) reducing the need for tool qualification, which is very expensive, by performing a tool-chain analysis of the tools used to develop a particular system; (3) methods and tools which reduce development costs and assist the designers in the certification lifecycle. This paper presents the RECOMP approach to safety-critical multicore systems, with a focus on methods and tools.**

I. Introduction

Safety is a property of a system that will not endanger human life or the environment. Many safety-critical systems are also real-time: in a *hard real-time system* the "correctness of the system behavior depends not only on the logical results of the computations, but also on the physical instant at which these results are produced" [1]. A *hazard* is a situation in which there is actual or potential danger to people or to the environment. *Risk* is a combination of the frequency or probability of a specified hazardous event, and its consequence. If, after performing an initial hazard and risk analysis, a system is deemed safety-related, it has to be certified [2]. Certification is a "conformity of assessment" performed by a third party. The current certification practice is "standards-based" [3], and requires that prescribed certification standards be followed, depending on the application area. For example, IEC 61508 [4] is used in industrial applications, ISO 26262 [5] is for the automotive area, whereas DO-178B/C [6] refers to software for airborne systems.

1. pop@imm.dtu.dk, *Technical University of Denmark, Denmark*
2. *Åbo Akademi University, Turku, Finland*
3. *fortiss GmbH, Munich, Germany*
4. *Validas AG, Munich, Germany*
5. *Symtavision GmbH, Braunschweig, Germany*
6. *Aalborg University, Aalborg, Denmark*
7. *Tecnalia, San Sebastián, Spain*

During the engineering of a safety-critical system, the hazards are identified and their severity is analyzed, the risks are assessed and the appropriate risk control measures are introduced to reduce the risk to an acceptable level. A Safety-Integrity Level (SIL) captures the required level of risk reduction. SIL allocation is typically a manual process, which is done after performing hazard and risk analysis [2], but researchers have proposed automatic approaches for SIL allocation. Although SILs differ slightly among areas (for example, the avionics area [6] uses five "Design Assurance Levels" (DAL), from DAL A to DAL E; in the automotive area there is the Automotive SIL, or ASIL), the approaches presented in this paper are applicable to all safety-critical areas, regardless of the standard. SILs are assigned to safety functions, from SIL 4 (most critical) to SIL 0 (non-critical).

The SIL assigned to a function will dictate the development processes and certification procedures that have to be followed. SIL 0 functions are non-critical and can be developed using any methods. For SIL 1, a more systematic approach is needed, to the level required by quality management standards such as ISO 9001. SIL 2 is quite similar to SIL 1, but typically involves more reviewing and testing. SIL 3 is significantly more difficult. Certification standards will suggest specific methods to be followed, and provide a checklist of techniques that are recommended to be applied. If "semi-formal" methods are acceptable in lower SILs, SIL 4 often requires formal methods, increasing further the difficulty and development costs associated to building safety-critical systems. SIL 4 is not possible to implement on a single chip due to redundancy requirements.

Many embedded applications, following physical, modularity or safety constraints, are implemented using distributed architectures, composed of several different types of hardware components (called *nodes*), interconnected in a network. The application software running on such distributed architectures is composed of several functions. The way the functions have been distributed on the architecture has evolved over time. Initially, in automotive and aerospace applications, for example, each function was running on a dedicated hardware node, allowing the system integrators to purchase nodes implementing

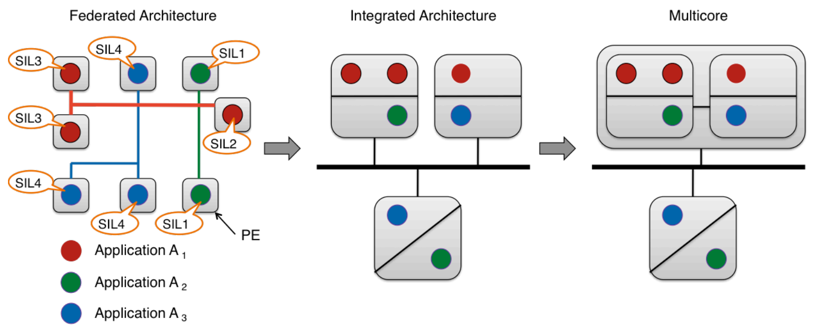


Figure 1. From federated to integrated and multicore architectures

required functions from different vendors, and to integrate them together into their system (this approach is also called a “federated architecture”). However, the number of such nodes in the architecture has exploded, reaching over one hundred in an airplane or a high-end car, leading to increased wiring, increased costs, size, weight and power consumption.

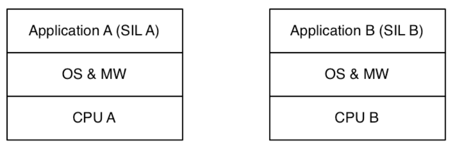
These trends have created a huge pressure to reduce the number of nodes, use the resources available more efficiently, and thus reduce costs. This is achieved through the integration of several functions in one node (also called an “integrated architecture”), see Figure 1. In addition, the nodes themselves can be integrated into a single chip, as is the case with the trend towards using multicore architectures, where several processing cores can be integrated onto a single chip, decreasing the costs, power consumption, size, and increasing the performance through parallelization [7]. The same trends are driving the integration of several levels of safety-criticality, together with non safety-critical functions. The “Research Agenda for Mixed-Criticality Systems” [9] defines a *mixed-criticality* system as “an integrated suite of hardware, operating system and middleware services and application software that supports the execution of safety-critical, mission-critical, and non-critical software within a single, secure computing platform”.

Such problems are faced in many other industries. At the European level, multi-cores are addressed through EU projects such as RECOMP (Reduced certification cost for trusted multicore platforms), which has the goal to “define a European standard reference technology for mixed- criticality multi-core systems supported by the European tool vendors” [10]. This paper highlights the approach taken by RECOMP, at the level of methods and tools, to reduce certification and re-certification costs. Section II presents the requirements from standards on separation, component models and methods and tools. The RECOMP approach is detailed in Section III, and the conclusions are presented in Section IV.

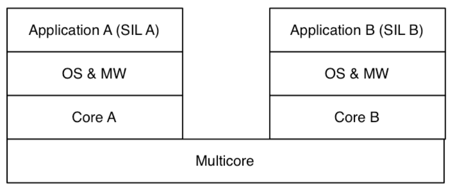
II. Requirements from Standards

A. *Separation Requirements from Safety Standards*

Functions of different SILs have to be separated. Otherwise, for example, a lower-criticality function could write in the code or data area of a higher-criticality function, leading thus to a failure. The current practice to mixed-criticality systems is to physically separate the different criticality functions in different hardware components, so they cannot influence each other, see Figure 2.a. The figure shows two applications, A with an SIL A, and B of SIL B. The applications are running on top of and operating system (OS) and also, possibly, a middleware (MW). If such an approach has worked in the past, the advent of multicores is, however, commonly regarded as a design challenge in the safety-critical area, as there are no established approaches to achieve certification. When several safety functions of different SILs share the same multicore, the standards require that the hardware and software be developed at the highest SIL among the SILs of the safety functions (for more details, see the standards), which is very expensive, see Figure 2.b. This is because the current multicore architectures do not offer enough separation between the safety functions [8]. For example, two tasks on separate cores might compete over the bus for access to the memory, increasing the worst-case execution times (WCETs), on which important timing guarantees are based. At the same time, there is an increased need for flexibility in the systems used in the safety-critical market. This need for flexibility puts new requirements on the customization and the upgradability of both the non-safety and safety-critical part. However, currently, any small change in the critical or non-critical functions, leads to an expensive recertification, at the highest SIL.



(a) Physical separation: each application is certified at it's own SIL lowest (re) certification cost, highest hardware cost



(a) Unconstrained integration: apps. certified at the highest SIL Highest (re)certification cost, lowest hardware cost

Figure 2. Multicores: increased certification costs

On a multicore, the safety functions have to be developed and certified at the highest SIL, unless, it can be shown that the implementation of the safety functions is sufficiently independent, i.e., there is both spatial and temporal separation among the safety functions. If the safety functions are sufficient independent, then the certification can be done at their own individual SIL level, and this would reduce costs, as shown in Figure 3, in Section III.A. This has to be done first by using a method for achieving independence (both temporal and spatial), and secondly, there has to be a justification of the method. Section III.A discusses the approach taken in RECOMP to provide sufficient independence.

B. *Tool Requirements from Safety Standards*

Tools are becoming more and more important in current software and system development, and tool usage can significantly reduce development costs, e.g., due to the automation of manual tasks. However, this increases also the threats that the tools can introduce errors into the products or fail to detect them. Therefore, current safety standards require analyzing the tools in the development and verification process. The analysis covers all tools and is done in three steps:

1) Does the tool have any impact on the safety of the product? This is true if the tool's output is part of the product, or if the tool is used in some development / verification steps required by the standards.

2) If the tool has such an impact, it needs to be classified. The classification depends on the standards.

3) If the classification result is "critical" the tool has to be qualified. Qualifications can be certified. There are different qualification methods possible, according to the standards.

There are three important standards to be considered: DO-178C with DO-330 for tool qualification, IEC 61508 and ISO 26262. For example, the ISO 26262 standard has three classes for the tools according to the required tool confidence level (TCL). TCL1 means no confidence required, while TCL3 denotes high Confidence requirement that has to be provided by qualifying the tool. The computation of the tool qualification levels depends on an analysis of the use cases for tools that have impact. Tool impact TI1 means no impact, and TI2 means that the potential tool failures can have impact on the safety of the developed system. The analysis considers all potential errors of the tool in the use cases and their detection (Tool error detection, TD) or prevention probabilities by checks and restrictions. These are classified as: TD 1—Tools with high detection probabilities for all potential errors; TD 2—Tools with some medium detection probabilities for some potential errors; TD 3—Tools with unknown or low detection probabilities for some potential errors. Based on this, the tool confidence level according to ISO 26262 is computed according to the following table:

TABLE I. DETERMINATION OF TOOL CONFIDENCE LEVEL (TCL) ACCORDING TO ISO 26262 [5]

		Tool error detection		
		TD1	TD2	TD3
Tool impact	TI1	TCL1	TCL1	TCL1
	TI2	TCL1	TCL2	TCL3

A simple example is the well-known tool *make*, which builds executable programs (and libraries) from source code, based on an input file called a *makefile*. In a general use-case, there is the danger of not considering some dependencies (tool faults or errors in *makefiles*), which means that the result would be a wrong (old) executable, which is not detectable when make is called without intensive analysis. Therefore *make* would have TCL 3 in such use-cases. However, if the process is changed a bit by requiring that *make* with the argument "*clean*" is executed before *make* is called to build the executable, then the probability of detecting non-considered dependencies is high and in this case *make* has a reduced TCL of 1.

Tool qualification is the next step. Only critical tools need to be qualified. The three main standards differ in the proposed qualification methods. While the IEC 61508 requires a proven in use argumentation, or a validation of the critical tools to be qualified, the ISO 26262 has the following methods selections available for TCL3 (and similar also for TCL2):

TABLE II. TOOL QUALIFICATION ACCORDING TO ISO 26262 [5]

	Methods	ASIL			
		A	B	C	D
1a	Increased confidence from use in accordance with 11.4.7	++	++	+	+
1b	Evaluation of the tool development process in accordance with 11.4.8	++	++	+	+
1c	Validation of the software tool in accordance with 11.4.9	+	+	++	++
1d	Development in accordance with a safety standard[a]	+	+	++	++

This table shows that the tool qualification according to ISO 26262 can be done in several ways; for higher tool confidence levels the recommended methods are development according to a safety standard and validation. Regardless of the approach taken, tool qualification is very costly, and should be avoided, if possible.

Section III.B presents RECOMP's approach to reducing tool qualification costs to a minimum.

C. *Component Model Requirements from Safety Standards*

To address the complexity of today's systems, engineers use component-based development approaches, which encourage modularity and reuse. Applying component-based methodology and component-based mindset during the Safety Development Life Cycle is believed to improve the reuse and testability of the developed system. It is expected that a component-based methodology will help reduce the development as well as certification and re-certification costs, using several aspects in certification standards (e.g., IEC 26262) that are relevant to support a modular certification of the system under consideration (e.g., a dedicated component model or the notion of a "Safety-Element out of Context").

There are differences among standards on this issue. In the avionics domain, DO-297 refers to "Integrated Modular Avionics", which provides separation among modules. In IEC 61508 the reuse is addressed with the concept of the compliance item. In this paper we will use ISO 26262 as an example. Thus, ISO 26262 (Part 10, Key Concepts) provides a component model that contains several elements: Function, System, Component, Element, Item, Part/Units. A *system* is a set of elements, at least sensor, controller, and actuator, in relation with each other in accordance with a design. Note that an element of a system can be another system at the same time. *Components* can be seen as non-system level, non-elementary, logically and technically separable elements; a component is a part of a system and consists of software units or hardware parts. An *element* is a system or part of a system, including components, hardware part, and software units. An *Item* can be seen as the system or the systems under consideration. Systems can be hierarchically structured and consist out of a set of *functions*. Each system is composed out of a set of components that in turn can be hierarchically structured as well. Systems are elements or are composed of elements. Non-atomic elements are components and atomic elements are either *software parts* or *hardware units*. The ISO 26262 component model supports a modular construction of the item under consideration, including suitable generic levels of abstraction for the corresponding elements.

Furthermore, it supports modular certification by providing means for system decomposition, and corresponding constructive and analytic methods of quality assurance for the specific type of elements, e.g. "Criteria for co-existence of different ASILs within the same element", see Part 9, Clause 6 in [5].

A distributed and modular development of reusable elements is also supported in ISO 26262, Part 10, Clause 10 by the "Safety-element out of context" (SEooC). A SEooC is defined as follows: a SEooC is never an item; a SEooC can either be a subsystem, a hardware component, or a software component; typically, requirements at higher levels remain in the status "assumed" (see ISO 26262, Part 8, Clause 5) and will be confirmed when the SEooC is used in an item development. The correct implementation of the assumed requirements will be verified during the SEooC development, but the validation takes place during the item development. The development of a SEooC starts at a certain level of requirements and design, and all information on requirements or design prerequisites are predetermined in the status "assumed". The use of assumed safety goals and functional safety requirements provided by the SEooC leads to an ahead-construction of modularly applicable sub-systems that will lead to a reduction of certification costs, due to reuse.

III. The RECOMP Approach

A. *Sufficient Separation*

One approach, taken by other EU projects, e.g., ACROSS [11], is to provide such a separation at the platform level, using architectural mechanisms, see Figure 3.a. For example, if the communication on the network is predictable, e.g., using a time-triggered policy, each core has a particular time slot when it's allowed to transmit, then the WCETs will not be affected by the shared network. However, semiconductor vendors operating in mass markets are unlikely to adopt and develop completely new architectures. RECOMP addresses several industries, including mass markets such as automotive and industrial automation, and has to deliver solutions applicable in these areas. The approach taken in RECOMP is to provide the separation among mixed-criticality functions using a combination of communication infrastructure ("Comm." In Figure 3.b), hardware (HW), operating systems (OS) and middleware (MW), methods and tools and component models, such that the required independence is achieved, see Figure 3.b.

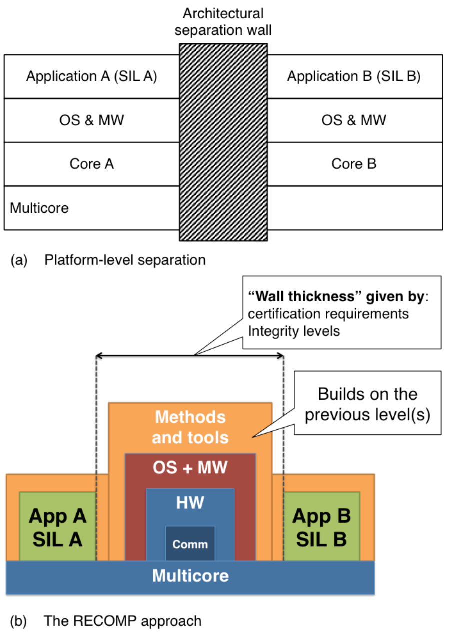


Figure 3. Multicores: two approaches to reduced certification costs

1) *The Infineon platform*

The complete description of hardware platforms developed in the project is available in [12]. One example platform is the Trusted Compute Platform (TCP), which is a development board designed by Infineon Technologies, AG, to support automotive and industrial applications. The board is designed to provide most of the conventional capability of an Automotive ECU such as, CAN, FlexRay and SPI networking connections, 5V ADCs with multiple channels, and an automotive quality safety watchdog. In addition, it provides more established industrial interfaces such as Ethernet and USB, a robust debug infrastructure which supports debug not only of the "virtual microcontroller" but of the TCP board itself, and an LCD screen for more immediate feedback than in a traditional closed Automotive ECU.

The main interest in the virtual microcontroller (from a RECOMP perspective) is the IP (see Figure 4), which has been enhanced to contain mechanisms developed to support the RECOMP requirements. The enhanced IP modules include: the CPU itself, main interconnect for code and data and interrupt infrastructure. The CPU has been modified to provide more support for isolation of tasks and applications. As well, the interconnect and all peripherals and memory modules are all enhanced to provide more isolation from tasks and processors, in addition this also provides protection against hardware faults. The interrupt mechanism, which supports some core-to-core communication requirements, is also enhanced to become more robust against faults. The TCP will be deployed with an AUTOSAR [13] OS and PharOS [14], which have been extended to take advantage of this platform. The hardware platform and the OS/middleware provide isolation services, thus supporting mixed-criticality applications.

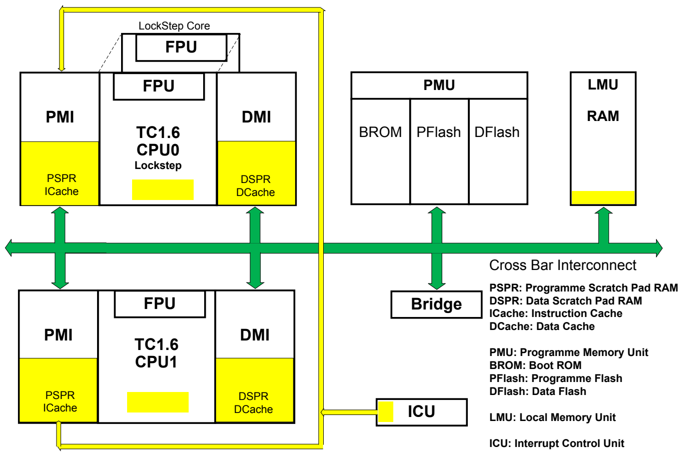


Figure 4. Subset of virtual microcontroller with RECOMP (enhanced IP in yellow)

2) *SymTA/S analysis tool*

The complete list of methods and tools developed in the project, which span the whole life-cycle and address multiple areas, is presented in [15][16][17][18]. The deployment of components has an impact on certification. The configuration relies on synthesis tools, which generate the configuration parameters such that separation is guaranteed, in terms of functionality, fault-containment and timing. Examples of configuration are the decision on which time-slots to use for a given component, if a time-based separation is used (e.g., in the case of static temporal partitions), or on which core and partition to place a function, which is a space-based separation. Configuration techniques can guarantee timing compositionality, which might be impossible to provide only relying on separation mechanisms from the platform. In our context, *partitioning* means deciding the borders of a component and/or how to group components into larger ones, and *mapping* means deciding the assignment of components to the different cores of a multi-core system, or an assignment to time slots—configuration decisions related to deployment. Both partitioning and mapping provide a way of separating the mixed-criticality functions such that non-interference is guaranteed.

In this context, one example tool is the SymTA/S tool [19] from Symtavision GmbH, which has been extended in RECOMP to be able to analyze different multi core configurations. Worst-case scheduling and performance analysis is used for to characterize the worst-case timing behavior of the software. This provides upper bounds for the system timing. In reality these bounds should never be reached but the system, especially a safety critical one, should be designed to handle such worst case scenarios. The whole technology is model based and abstracts from details of the systems and considers only relevant parts for the scheduling and performance analysis. It can be used for modeling and planning of upcoming systems but also for validation of already existing and finished systems.

With multi cores new challenges appears for developers but also for the scheduling analysis. One identified issue in multi cores are communication overheads resulting from core cross-communication. Communication means here the exchange of data between software functions running on different cores. This data is written by one core into a memory and is read by the other core from the memory. In AUTOSAR, this is covered by the IOC (Inter-OS-Application communication). Depending on the structure of the different cores and memories and also on the implementation of the IOC by an operating system, different amounts of overheads are introduced. For heterogeneous memory structures, the overhead depends on where the data is stored. This could be in local fast memories for one core or in global shared memories connected by crossbars or other bus technologies.

Industrial experience, and the experience in RECOMP, has shown that these overheads can get significant and the overhead dominates the system. In RECOMP, Symtavision added an explicit consideration of memory access overheads to their SymTA/S tool. This allows to analyse how much data is accessed by which core and software function and how much overhead does it introduce. With this information, optimizations are possible to reduce the overhead in the system; less overhead means more performance and reduces bottleneck situation.

B. *Minimizing Tool Qualification*

The Tool Chain Analysis (TCA) [20] method has been developed and applied within the RECOMP project. It automatically computes the tool confidence level (TCL) according to ISO 26262 and can also reduce the tool qualification level. The method is based on a formal model of tools, use cases, artifacts, potential errors, checks and restrictions etc. and an algorithm to compute the TCL. Furthermore, it has an error model to systematically derive potential errors using attributes that characterize the tools (black-box and white-box). An important aspect of the method is that it allows formalizing assumptions in order to express that certain checks for potential errors during the development process have to be applied by the developers.

The tool chain analysis method has been implemented in a “Tool Chain Analyzer” software tool. The tool also contains a report generation feature and has a documentation explaining the model and features. The Tool Chain Analyzer has TCL 1 with the assumption that the confirmation review of the TCLs and the qualification measures are done on the generated report. Since this review is required by ISO 26262, the tool is uncritical and requires no further qualification.

The tool has been applied in three large industrial projects. One result was a tool chain with 39 tools, which could be extended by small process extensions and some redundancy, such that only the qualification of one tool was necessary. Compared with the results of a fixed tool confidence level classification as proposed in literature, this has dramatically reduced the tool qualification costs by reducing the number of tools to be qualified from 13 to 1. Therefore, the tool chain analysis method can substantially reduce certification costs by minimizing costly tool qualification.

C. *Component Models for Certification*

From a development process point of view, significant cost savings can be achieved by streamlining the management of the needed information for the certification authorities. Traceability of information through the development cycle requires significant effort. In order to effectively use components, they need to carry enough information with them in order to be useful in a development process for certified safety critical systems. What information is needed is domain specific (standard specific) and depends on the safety integrity level of the system where the component will be used. Verification and validation is also a time and resource consuming task. Automated test case generation, other automated verification techniques and re-use of components could bring savings.

We have surveyed the existing component models and proposed a component model specification for safety-critical systems to support certification. The component model specification has been initially developed as an extension to the component model of AutoFocus 3 (AF3) [21]. AF3 system model is divided into several models that provide different levels of abstraction. This enables component-based modeling at the functional, logical, and technical design levels. The AF3 component model is inspired from the Esterel language currently used in state-of-the-art development of safety-critical systems, and uses a message-based, discrete-time communication scheme as its core semantic model. To support a wide range of development processes, RECOMP is currently extending the CESAR metamodel [22] with the essential parameters of the component model specification for certification. The CESAR metamodel is part of the ARTEMIS tool reference platform [23] addressing the complete development lifecycle of safety critical embedded systems, without focusing on certification.

In order to use a component model specification for certification, we have identified certain basic parameters to be associated, with respect to the component meta-model that might be necessary for certification purposes. These parameters are: *Unique ID*–To ease identification within a component library; *Version number*; *Name*–Name of the component; *Description*–Description of the functionality provided by the component; *References* to requirements fulfilled by the component; *References to test cases*–Tests applied to verify the component; *Safety Integrity Level* (SIL)–The SIL capability of the component; *Functionality–Safety related*–the component implements functionality related to the safety function; *Diagnostic*–the component implements a diagnostic function; *Non-safety related*–the functionality provided by the component is non-safety related; *Rules/Restrictions*–Specifies rules/restrictions that have to be followed in order to reuse the component;

The proposed component model has been used to model an industrial case study (emergency stop function) from Danfoss Drives [15].

D. Verification and Validation vs. Certification

Assume-guarantee methods have been used to support compositional *verification*. However, these methods are not applicable to *certification* [3], across all industrial areas. Certification for multicore architectures can use compositional verification only if separation mechanisms are provided, which guarantee that, if something goes wrong with a component, will not influence indirectly (through shared memory corruption for example) other components. Component *validation* checks the components in isolation, before they are put together. This is done offline, using testing and/or verification to determine the correctness of the implementation and online using runtime monitoring to check the separation between the safety and non safety-critical parts.

IV. Conclusions

This paper has presented the RECOMP approach to implementing mixed-criticality applications on multi-core platforms aiming at reducing certification and re-certification costs. The increased certification costs are due to the nature of multicores, which do not provide sufficient separation among mixed-criticality functions. Thus, the designers are forced to develop at the highest criticality level, which is prohibitively expensive, and to re-certify the system for any small upgrade, even of the non-critical functions. RECOMP reduces the certification costs by offering sufficient separation, through mechanisms at platform, OS, middleware and component levels, and through the carefully planned use of methods and tools such that this separation is guaranteed. In addition, the methods and tools used in a tool-chain to develop a particular system can be analyzed, such that costly tool qualifications are reduced and minimized.

References

[1] Kopetz, H. *Real-time systems: design principles for distributed embedded applications*, Springer, 2011.

[2] Storey, N. *Safety critical computer systems.* Addison-Wesley, 1996.

[3] Rushby, J. Just-in-time certification. 2007.

[4] Standard IEC 61508: Functional Safety of Electrical/ Electronic/ Programmable Electronic Safety-related Systems. International Electrotechnical Commission, 2010.

[5] ISO/DIS 26262:. International Standard Road vehicles— Functional safety, International Organization for Standardization, 2012.

[6] RTCA, DO-178C Software Considerations in Airborne Systems and Equipment Certification, 2011

[7] Ernst, R. Certification of trusted mpsoc platforms. 10th International Forum on Embedded MPSoC and Multicore, 2010.

[8] Nowotsch, J., Paulitsch. M. Leveraging Multi-core Computing Architectures in Avionics. In *European Dependable Computing Conference*, pp.132-143, 2012.

[9] Barhorst, J. et al. A research agenda for mixed-criticality systems. In *Cyber-Physical Systems Week,* April 2009.

[10] RECOMP project, Reduced certification cost for trusted multi-core platforms, www.recomp.eu

[11] ACROSS project, ARTEMIS CROSS-Domain Architecture, www.across-project.eu

[12] Deliverable D3.4, HW support for operating systems, applications and monitoring, RECOMP, 2012

[13] AUTOSAR, AUTomotive Open System ARchitecture, www.autosar.org

[14] Lemerre, M. et al. Method and tools for mixed-criticality real-time applications within PharOS. In *Object/Component/Service-Oriented Real-Time Distributed Computing Workshop*, pp. 41-48, 2011.

[15] Deliverable 2.1, Component model specification, RECOMP, 2012

[16] Deliverable 2.2, Report on component validation methods and techniques, RECOMP, 2012

[17] Deliverable 2.3, Report on methods and techniques for verification, validation and timing analysis, RECOMP, 2012

[18] Deliverable 2.4, Report on configuration methods and techniques, RECOMP, 2012

[19] SymTA/S, www.symtavision.com

[20] Tool Chain Analyzer, www.validas.de

[21] AutoFOCUS3, af3.fortiss.org

[22] D_SP1_R3.2_A_M2, Meta-Model Specification for RTP 2.0, www.cesarproject.eu

[23] Artemis tool platform, www.artemis-ia.eu/tool_platform