Towards safe mixed critical embedded multi-core systems in dynamic and changeable environments

Christoph Dropmann, Tiago Amorim, Alejandra Ruiz, Daniel Schneider
Fraunhofer IESE, Kaiserslautern GERMANY
ICT-European Software Institute Division, TECNALIA. Derio, SPAIN
{christoph.dropmann, tiago.amorim, alejandra.ruiz, daniel.schneider}@tecnalia.com / @iese.fraunhofer.de

***Abstract*— Nowadays, embedded systems are evolving from closed, rather static single-application systems towards open, flexible, multi-application systems of systems. From a safety engineering perspective, this trend certainly is a curse as it detriments the base assumptions of established engineering methodologies. Combinatorial complexity and the amount of uncertainty encountered in the analysis of such systems are the reason why we investigate the possibility of an integrated contract-based approach. Our solution approach is covering vertical dependencies (between platform and application) and horizontal dependencies (between applications) in order to efficiently assure the safety of the whole system of systems through modularization. Moreover, the contract-based nature of our solution helps engineers to systematically and unambiguously define potential interferences as well as validating the compatibility of the components (platform and applications) behavior ensuring safety will not be compromised.**

Keywords— safety, assurance, contracts, multi-core, ConSerts

I. INTRODUCTION

In recent years, developers of distributed embedded systems design their products in accordance with the principles of integrated architectures. These principles are applied for regular products as well as for those that have to be safe. Standards such as ARINC 653 in the avionics and AUTOSAR in the automotive domain constitute the corresponding foundations in their respective domains. The envisioned shift towards multi-core platforms, which are relatively cheap and yet powerful, rises additional challenges – in particular with respect to safety assurance. Taking full advantage of the multi-core technology implies the coexistence of applications on the same electrical control unit with several integrity levels (mixed criticality). In parallel, the trend towards open systems, such as dynamic compositions of different cars, traffic infrastructure, and Internet-based services, manifests in new computing paradigms, such as cyber-physical system and Internet of thinks. However, it is clear that many of the envisioned services and applications require very high integrity, as we are talking about safety critical systems.

In common practice, safety is ensured through diligent engineering that is guided by domain-specific safety standards such as the ISO 262626 for automotive. Such engineering is typically a complex and laborious endeavor, demanding detailed domain knowledge and experience from the engineers. Base assumption of any established safety engineering approach is that the system, as well as its anticipated environment, is known (Safety Element out of Context of ISO 26262 being one notable example to soften this base assumption) and that they can thus be analyzed comprehensively. Integration tasks are exclusively executed by human engineers at development time and in accordance with the aforementioned regulations. Combining different applications on a single ECU, maybe even supporting dynamic download and update, allowing dynamic integration of different systems in the field – these integration scenarios go clearly beyond what is possible based on today's engineering approaches. As a consequence, insufficient safety assurance might become a show stopper for some of the most promising and ground breaking aspects of the new computing paradigms.

Recognizing this problem, some research has been done with respect to formalizing and modularizing safety in an adequate way to support the different types of integration scenarios. In the recent nSafeCer project [1], for instance, it has been investigated how component-based integration could efficiently be realized based on Safety Contracts specified in OCL (i.e. based on the well-known UML) [2]. The approach focuses on early safety verification support, whereas our work builds on two approaches that are similar in the utilization of contracts, but that have been specifically designed to enable automated integration. On the one hand, the VerSaI (Vertical Safety Interface) approach for dealing with safety dependencies between applications and the execution platform on which the applications get deployed [3]. And on the other hand, the ConSerts (Conditional Safety Certificates) approach focusing on safety assurance of open systems. ConSerts are basically a means to establish a variable safety case with formalized demands (i.e. assumptions) to be resolved when systems compose (i.e. at runtime). Based on the fulfillment of assumptions, guarantees are determined and propagated through the dynamically formed composition hierarchy (i.e. through a series of guarantee-demand relationships) [4].

A first combination of these approaches has been proposed in M2C2. (Multidirectional Modular Conditional Safety Certificates) [5]. M2C2 is a solution for a dynamic runtime safety assessment for a mixed critical multicore-system of systems. The approach can support the development to focus on emerging safety incompatibilities and provide suggestions for a valid alternative strategy for integration, reducing the effort needed for safety assessment and thus time to market. The extension of M2C2 presented here is the linkage between the

dependencies and the discussion of the potential interactions. The next section describes the different types of safety-related dependencies considered by M2C2. We will then briefly explain the M2C2 approach and the newest extensions before we finally conclude in Section IV.

II. Horizontal and Verical Dependencies

In order to modularize safety on the level of system components, understanding the boundaries and the interaction schemes between components is key. And since it is our aim to explicitly reflect safety-related requirements between application software and the platform it is running on, we divide systems in two corresponding types of elements: platform elements and application elements. Platform elements are components (hardware and software) which provide function-independent platform services. Such services can provide hardware resources, but also other OS or middleware services (e.g. IO or communication). Applications are software components that provide system-level functions directly to human end users or (as services) to other applications. Based on this distinction between application and platform, we encounter two types of safety relevant dependencies that are first introduced by [6]. On the one hand, the horizontal dependencies, which are dependencies between applications that might either be running on the same or on different platforms. On the other hand, vertical dependencies, which are dependencies between a platform and an application. Both kinds of dependencies shall be addressed by corresponding safety interfaces, i.e. horizontal and vertical safety interfaces (cf. Figure 1).

The vertical interfaces (as introduced by the VerSaI approach) consequently describe the safety-relevant relations between an application and a platform service. Platform services, e.g., libraries, communication protocols, or operating system services, tend to be developed with reuse in mind from the early design phases on. Platform developers do not consider in advance the possible systems in which the platform service will be allocated and they do not know in which way the service will be part of the application functionality. The horizontal interfaces (as introduced by the ConSerts approach) describe safety-relevant relations between applications that might either be located on the same (safety-related independence then to be proven separately) or on different platforms. These relations are characterized by formalized safety guarantees and demands that are associated with required and provided services of applications. Different applications (and thus systems/devices) join together to render higher level services that could not be rendered by a single system alone. Conceptually, dynamic composition hierarchies are spanned up in this process, where each composition element possesses its own ConSert. ConSerts are evaluated along the service-based dependencies in the composition, finally determining the safety guarantees of the root element. The root application ConSert has been engineered with the whole application in mind and it also holds the overall safety responsibility. Accordingly, any ConSert has a certification scope and responsibility over anything that is below itself in the composition hierarchy. In relation to Figure 1 and the notion of horizontal interfaces we state that even though we have certain composition hierarchies between applications, the dependencies are still denoted as horizontal safety dependencies.

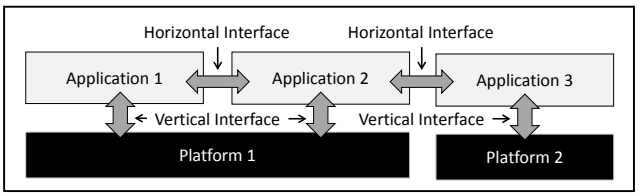

Fig. 1. Horizontal and vertical dependencies

III. Multidirectional Modular Conditional Safety Certificates

The main idea of the solution is to shift the final safety evaluation of an arbitrary constellation of applications and platforms from development time to runtime. The safety related information of the smallest units (e.g. applications, components, modules) is defined at development time in contracts. At runtime integration, the systems shall assemble the safety critical information and evaluate the safety of the overall composition. In addition, the system checks the validity of the contracts at runtime to be able to deal with runtime adaptations. Shifting safety evaluations into runtime significantly reduces development time complexity and thus the time to market of new improvements. We denote these contracts as Multidirectional Modular Conditional Certificates (M2C2), since they are multidirectional in covering both, horizontal and vertical interfaces. They are modular due to their composability characteristics and they are conditional because the contracts can have open ends (demands) that need to be fulfilled by the environment. In the following subsections, we will give an overview of the M2C2 solution and explain the different sub contributions as well as their interactions.

A. Solution Overview

In M2C2 each considered unit of composition has its own runtime representation of a contract. A contract consists of a demand and guarantee interface. Demands are requirements outside the boundaries of a module. They represent safety requirements wrt. the system environment that cannot be verified at design time already due to a lack of information. The environment is either related to the horizontal interfaces (i.e. other systems or properties of the physical environment) or to the vertical interfaces (i.e. safety related characteristics of the underlying platform). Due to the variability of the environment, there will typically be different constellations of demands associated with different levels of guarantees. Figure 2 illustrates the basic components of M2C2, the demand and guarantee based contract languages VerSaI (Vertical Safety Interface) for the vertical safety related dependencies, ConSerts (Conditional Safety Certification for Open Adaptive Systems) for the horizontal safety related dependencies, and, means for their integration to be explained later.

B. Horizontal and Vertical Interaction

M2C2 guarantee and demand relations can be represented as a tree like graph as it has been introduced by the ConSerts approach. Each sub-tree of this graph is a contract, a ConSert thus being a set of contracts or a “variable” contract. The relations between the elements of this tree are binary logic, specifying which demands must be fulfilled to provide certain guarantees. The demands and guarantees are defined using

service-specific safety properties in combination with an integrity level, stating the assurance level delivered by a guarantee or requested by a demand (cf. [4] for more details).

Moreover, each application has its vertical interfaces with a platform and if the application is required to provide any service guarantee, the vertical safety interface has to be fulfilled. To specify such relationships, the VerSaI language uses dependency classes which are platform service failures (focus on the detection or avoidance of platform failure), health monitoring (focus on trapping and encapsulating of application failures), service diversity (also called dissimilarity or independence) and resource protection (to ensure partitioning and segregation). The platform developer can define platform guaranties based on the dependency classes, defined failure modes for a generic application (e.g. execution time deviation) and platform elements (e.g. scheduling jitter failure) [3]. Based on the dependency classes and the associated failure modes, combined with the horizontal demands and guarantees of an application, the vertical application demands could be automatically derived.

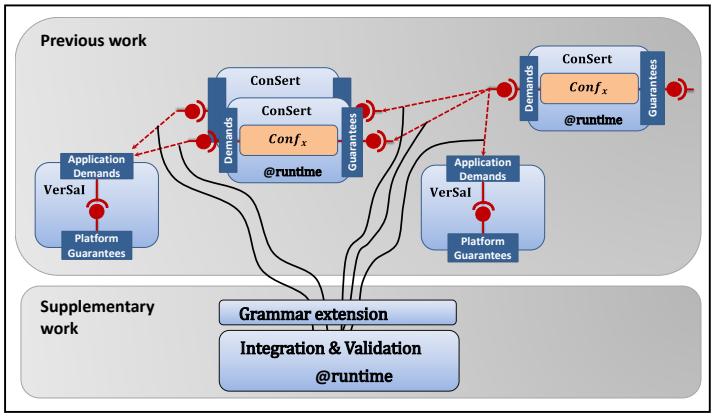

Fig. 2. M2C2 Solution Overview

The M2C2 contracts must be available at runtime in a format appropriate for automated composition and evaluation. For this reason, we describe them in XML format. A grammar extension allows combining the vertical VerSaI-based contracts with the horizontal ConSerts based contracts. The main idea is that within a ConSerts conditional certificate of a configuration (i.e. application variants that can be switched dynamically; *Confx* in Figure 2) of an application, the Vertical Demands are integrated. A M2C2 certificate can technically be represented as an optimized BDD (Binary Decision Diagrams) for maximizing evaluation performance. The BDD contains the information that an application guarantee can be granted e.g. if different demands to other applications are satisfied at runtime. In addition, Vertical demands are logically connected within the BDD and the horizontal demands. E.g., a monitor application fulfills its guarantee to set a fail-safe signal (horizontal) 1) if it receives a fail-safe signal from an extern application (horizontal) and dedicated message corruption, loss and transmission demands are fulfilled (vertical) or 2) if the application is able to determine the fail-safe signal itself. In case 2), the application demands to additional sensor applications (horizontal) and analog output value and delay failures (vertical) need to be fulfilled at runtime.

The example shows that the holistic consideration of horizontal and vertical dependencies offers additional levels of variety and flexibility. The solution idea is to identify based on the initial resolution of the horizontal conditional certificates the corresponding vertical demands. We assume that in many situations, especially in case of a multi-core system, several and different combinations of horizontal and vertical demands may fulfill an applications guarantee. That implies an additional level of adaptation within M2C2 compared with just ConSerts, for instance in the area of fail operational to support commercial-of-the-shelf multicore platforms for e.g. autonomous driving.

IV. Conclusion

In summary, M2C2 is a solution concept to guarantee safety for open and adaptive multi-core systems. The main idea is to split the safety relevant dependencies within a system or system of systems in a horizontal and vertical part. Demand and guarantee based conditional certificates are used as means of specification. At runtime, the M2C2 framework autonomously evaluates if the safety relevant interfaces are valid and compatible and what top-level guarantees can be offered for a given composition. Same as ConSerts, M2C2 does provide support to pre-configured adaptive systems, where the system is self-adaptive at operational time based on a range of already defined configurations.

The feasibility and limitations of M2C2 remains to be evaluated, since we are still in an early stage of work. We are currently applying M2C2 in a case study with a hybrid powertrain scenario and an eclipse-based tool implementation to create and evaluate the contracts. Further ongoing work is focusing on a method to identify and address platform-induced interferences automatically to ease the definitions of the vertical contracts.

Acknowledgment

This research has received funding from the EMC2 project. This is an ARTEMIS Joint Undertaking project in the Innovation Pilot Programme ‘Computing platforms for embedded systems’ (AIPP5) under grant agreement n° 621429.

References

[1] nSafeCer project: Safety Certication of Software-Intensive Systems with Reusable Components. Project Grant Agreement no 295373. More information at: http://safecer.eu/, [Online].

[2] Gomez-Martinez et al., „Model-Based Verification of Safety Contracts,“ in *Springer International Publishing*, 2015.

[3] B. Zimmer et al., „Vertical Safety Interfaces -- Improving the Efficiency of Modular Certification,“ in *SAFECOMP*, 2011.

[4] D. Schneider et al., „Conditional Safety Certification of Open Adaptive Systems,“ in *ACM Trans. Auton. Adapt. Syst. 8, p. Article 8*, 2013.

[5] T. Amorim, A. Ruiz, C. Dropmann, D. Schneider, „Multidirectional Modular Conditional Safety Certificates,“ in *SAFECOMP, ASSURE*, 2015.

[6] B. Zimmer, S. Bürklen, M. I. Knoop, J. Höfflinger and M. Trapp, „Vertical Safety Interfaces - Improving the Efficiency of Modular Certification,“ in *SAFECOMP*, 2011.