Technology-preserving transition from single-core to multi-core in modelling vehicular systems

Alessio Bucaioni[1,2], Saad Mubeen[1,2], Federico Ciccozzi[1], Antonio Cicchetti[1], and Mikael Sjödin[1]

[1] School of Innovation, Design and Engineering
Mälardalen University, Sweden
{name.surname}@mdh.se
[2] Arcticus Systems AB, Järfälla, Sweden
{name.surname}@arcticus.se

Abstract. The vehicular industry has exploited model-based engineering for design, analysis, and develop of single-core vehicular systems. Next generation of autonomous vehicles will require higher computational power, which can only be provided by multi-core platforms. Current model-based solutions and related modelling languages, originally conceived for single-core, can not effectively deal with multi-core specific challenges, such as core-interdependency and allocation of software to hardware. In this paper, we propose an extension to the Rubus Component Model, core of the Rubus model-based approach, for the modelling, analysis, and development of vehicular systems on multi-core. Our goal is to provide a lightweight transition of a model-based approach from single-core to multi-core, without disrupting the current technological assets in the vehicular domain.

Keywords: Model-based engineering, multi-core, vehicular domain, embedded, real-time, metamodelling.

1 Introduction

Software is ubiquitous in our society. In automotive, vehicles have transitioned from being mechanics-intensive to software-intensive systems [11]. For instance, the throttle control system of a modern vehicle is realised by means of Electronic Control Units (ECUs), sensors, and actuators, connected by several networks, and run by software, which replace the mechanical linkage between the accelerator pedal and the throttle. The current trend in the vehicular domain is to go towards vehicles capable of autonomously driving. While most of the current vehicular systems still employ single-core ECUs, the tendency is to switch to multi-core ECUs. In fact, next generation vehicles, particularly autonomous ones, are expected to require higher computational power, which can only be provided by multi-core solutions.

On the one hand, the shift to multi-core impacts the way vehicular software is designed, analysed and developed. Current model-based solutions, specifically tailored to single-core, are not as effective when dealing with multi-core specific challenges, such as core-interdependency and allocation of software to hardware. On the other hand, the vehicular industry can not prescind from the current technological assets for many reasons, among which:

Legacy. It has been estimated that up to 90% of the software of a new vehicle release can be reused from previous releases when using model-based engineering [37].

Organisation. Original Equipment Manufacturer (OEM) companies define their technological assets based on decennial contracts with Tier-1 and Tier-2 companies. Changes to these assets shall not affect these contracts.

Certified run-time support. Functional safety [35] is paramount for the safety criticality of vehicles [36]. Current model-based solutions rely on certified development environments and real-time operating systems [1]. Typically, the certification process adds a development cost overhead between 25 and 100%, and it lasts several years [28].

We have investigated the extension of Rubus [3], a commercial model-based approach for vehicular single-core systems, to multi-core with the intent of not disrupting the current vehicular technological assets related to it. Our hypothesis is two-fold. (H1) Abstraction provided by models and automation provided by model transformations can be a game changer in the development of multi-core applications. Abstraction permits to detach software functional modelling from multi-core hardware modelling and software/hardware allocation modelling. Automation can support the developer in taking important decisions, such as how to allocate tasks to available cores in order to maximise a specific quality aspect [12]. (H2) A lightweight transition of a model-based approach from single-core to multi-core which does not affect critical aspects such as certified run-time support and lastingness of legacy applications is possible.

In model-based engineering, metamodels play a pivotal role as they define the set of available modelling entities and relationships for representing the software architecture and its quality attributes. Moreover, they enable automation via model transformations. However, it is essential that metamodels effectively prescribe the type system, the structure, and the behaviour of domain-specific applications [33]. In [23] we have discussed some modelling languages (among which Rubus Component Model) used for single-core vehicular applications and highlighted the issues arising when using them for modelling multi-core applications. In particular, existing structural hierarchies lack concepts for representing multi-core aspects (e.g., cores and partitions) and do not provide explicit support for core-interdependency and allocation of software to hardware.

In this paper, we propose an extension to the Rubus Component Model (RCM) [17], core of the Rubus approach, to support multi-core. This represents the first crucial step in the transition from single-core to multi-core. The contribution of the proposed extension is two-fold. We provide a modelling language able to prescribe type system, structure, and behaviour of multi-core applications (H1). In particular, the proposed extension comprises modelling elements for representing the software architecture, the hardware platform, and the software to hardware allocation. We ensure backward compatibility with legacy single-core applications modelled with RCM and do not entail any modification to the Rubus run-time layer, the Rubus Kernel (H2).

The remainder of the paper is structured as follows. Section 2 introduces RCM and motivates its selection as well as its extension. Section 3 presents a comparison between existing related approaches documented in the literature and our solution. Section 4 describes the proposed solution in all its constituents. Section 5 describes the application of the proposed solution to an industrial vehicular application. Section 6 and Section 7 discuss the benefits and limitations of our solution and conclude the paper, respectively.

2 The Rubus Component Model

There are several modelling languages used in the vehicular domain, such as RCM, AUTOSAR [2], ProCom [34], COMDES [20], AADL [15], to name a few. These languages were not conceived to deal with the complexity of predictable vehicle software specifically developed to run on multi-core platforms.

We focus on RCM and its extension for multi-core due to the following reasons. RCM is a good candidate to overcome the issues related to predictability thanks to its statically synthesised communication as well as its predictable and fine-grained execution model [24]. RCM uses pipe-and-filter communication and distinguishes between the control and data flows among its software components. In [25], we showed that these two features are central for providing early timing verification of the modelled system, e.g., by supporting end-to-end timing analysis [14]. Another reason for focusing on RCM is the small run-time footprint of the developed software (automatically generated from RCM models) as compared to other languages [25].

RCM is developed by Arcticus Systems AB[3] in collaboration with Mälardalen University. Through the years, RCM has been adopted by several OEM, Tier-1 and Tier-2 companies (e.g., Volvo Construction Equipment, BAE Systems Hägglunds, Hoerbiger and Knorr Bremse) for the development of embedded real-time software. RCM provides the Rubus Kernel, a dedicate real-time operating system, which is available for different processor architectures and certified according to the ISO 26262 [1] standard ASIL D (Road vehicle – Functional Safety).

RCM was originally thought for providing modelling purposes, but it did not feature model-based mechanisms, i.e. automation in terms of model transformation. In order to achieve a full-fledge model-based approach, in [8] we reverse-engineered the RCM specification in order to express it in a more canonical form, a metamodel, which we called RubusMM. RubusMM included concepts for expressing software architectures and concepts for describing timing information of vehicular single-core applications. In this paper, we extend RubusMM to enable modelling of software applications for multi-core.

3 Related Work

AUTOSAR [2] is an industrial initiative to provide standardised software architecture for the development of vehicular software systems. Since the emergence of AUTOSAR 4.0, multi-core support is part of the standard. From a modelling point of view, AUTOSAR makes use of a structural hierarchy similar to the one leveraged by RCM and consisting of system, node, and software component elements. However, it does not

[3] `https://www.arcticus-systems.com`

distinguish between the control and the data flows at the application software level. In [25], we discussed how this feature is central for providing early timing verification of the modelled system. The AUTOSAR modelling language does not provide clear separation of concerns between the functional (software) and architectural (platform) models [32]. Recently, several works on the use of AUTOSAR for multi-core have been proposed both from industry and academia. However, their main focus is on the adaptation of the AUTOSAR run-time support rather than on specific modelling challenges such as, e.g., allocation of the software components. In [29], the authors investigated the use of AUTOSAR for virtualised architecture and they identified some challenges on the use of AUTOSAR for multi-core. They concluded that additional features for the dynamic allocation of the software were needed. In [22] and [6], the authors evaluated AUTOSAR systems realised with a centralised architecture where the layered architecture was entirely allocated to one of the available cores only. Both the approaches were able to demonstrate that the behaviour of the multi-core software system and its footprint did not significantly vary from the corresponding single-core configuration. However, in both approaches, the uneven distribution of the workload among the cores led to performance and timing verification issues. In [30] and [31] the authors described AUTOSAR systems based on virtualised architectures where hypervisors coordinate multiple software systems with same or different real-time operating system(s). The use of hypervisors complicates early timing verification as it introduces additional complexity. From a footprint point of view, the virtualised architecture may lose its efficiency as each software system can carry a different real-time operating system. Both approaches rely on certified versions of AUTOSAR systems.

Besides technologies specific to the vehicular domain, several works have discussed the use of UML and the UML profile for MARTE [4]. Being general-purpose, these technologies are often used as complementary to domain-specific languages as, e.g., AUTOSAR and RCM. In [21], the authors present the VERTAF/Multi-core UML-based framework for the development of multi-core software. Within VERTAF/Multi-core, the software system is described by means of UML class diagrams, timed state machines and sequence diagrams. Model transformations are used for generating extensions to these models for checking the viability of the design with respect to schedulability and conformance to the specifications. In [13] and [26] MARTE is used for representing the high-level architecture of the software system and as enabler for code-generation. In the first approach, UML is used for modelling the software components while MARTE is used for modelling hardware and software to hardware allocations. Starting from these models, code is automatically generated and timing verification through simulation is run. The second approach focuses on the system deployment of component-based systems. MARTE is used for modelling high level description models from which different models representing allocations of components are generated by means of code generation. In [12], MARTE is used for describing a task model and the allocations of tasks to cores for combined simulation- and execution-based task allocation optimisation. In [16] the authors introduce a MARTE-based framework, named GASPARD, for the design of parallel embedded systems. Herrera et al. [18] discuss a framework for the design space exploration of embedded systems based on MARTE.

The framework, called COMPLEX, uses MARTE for describing the different architecture solutions composing the design space.

AADL [15] is an architecture description language developed for the avionic domain, but currently used for modelling embedded systems in general. AADL provides multi-core support and a clear separation of concerns between software and hardware elements.

4 Extending Rubus Component Model for Multi-core

In this section, we describe the extension to RCM for modelling vehicle software on multi-core. The extension is formalised by means of metamodelling. We compare the extended RCM with its previous definition, given in [8], thus highlighting differences and commonalities. The extension comprises the addition of modelling packages, classifiers, features, and relations as well as the modification of some hierarchical structures.

With respect to the previous definition, we have introduced packages for ensuring a better separation of concerns, improving the understandability of the metamodel, and simplifying future extensions. The RubusMM packages involved in the extension are *RCM_COMMON*, *RCM_HW* and *RCM_SW*[4]. *RCM_HW* contains the elements for modelling the hardware platform: *Node*, *Target*, *Allocator*, *Core*, and *Partition*. *RCM_SW* contains the elements for modelling the software architecture: *Allocatable*, *Mode*, *Assembly*, and *Software Circuit*. *RCM_COMMON* contains elements which are common to different packages as, for instance, *System* and port elements. Fig. 1 shows a fragment of RubusMM containing elements from *RCM_HW* for modelling the hardware platform. *System* represents the system under development. As all the elements in RubusMM, it inherits from the abstract metaclass *NamedElement* which provides two attributes: *name* and *ID*. We extended *System* with the reference *timingConstraint* for enabling the specification of timing constraints, occurrences and events which are used for timing verification.[5] These constraints are used for running timing analysis, but we employed them for automatically generating the set of RCM models satisfying a given set of timing requirements too [7].

A *System* contains one *Network*, one or more *Node* elements, and one or more *Mode* elements. A *Network* element models all the messages exchanged among the *Node* elements. It has two attributes, *protocol* and *speed*, which specify the protocol (e.g., Controlled Area Network (CAN) [19]) and the speed of the network in Kbit/s, respectively. A *Node* is an abstraction of a *Target*, which models an ECU.

A *Node* can be concretised by means of different *Target* elements (e.g., Infineon XC167-32 and MPC555 from NXP). Its reference *activeTarget* defines which *Target* is active in each given execution. The definition of *Node* has been extended with the references *timingConstraint*, *portIO*, and *portNetwork*. *portIO* and *portNetwork* model the peripherals and the inter-node communication, respectively.

[4] The complete explanation of RubusMM is not in the scope of this work. The interested reader may refer to [8].

[5] *TimingConstraint* and other elements from different RCM packages are not part of this extension. However, they are put in relation to the extension as they contribute to a holistic view of the language and its peculiarities. .

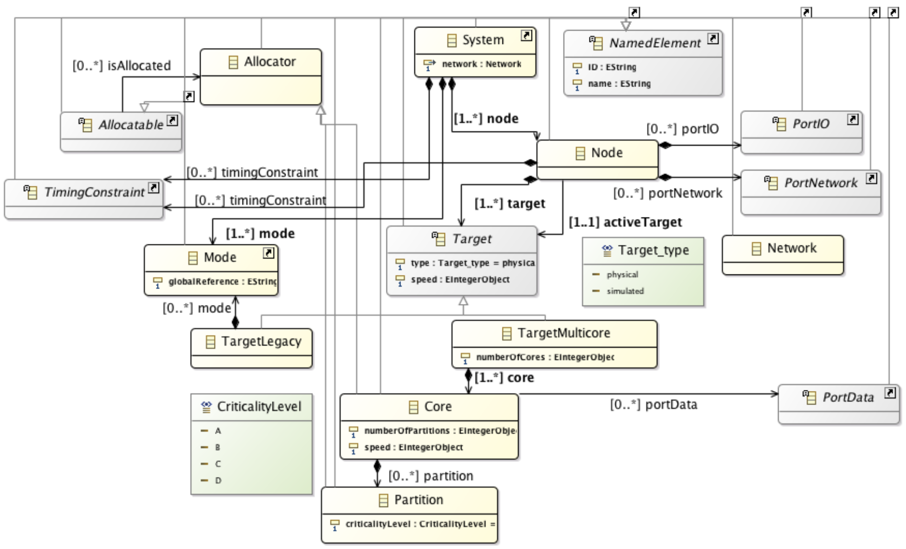


Fig. 1: Fragment of the *RCM_HW* package for modelling the hardware platform.

In the previous definition of RubusMM, *Node* contained *Target*, which in turns contained *Mode*, representing the software application. However, the containment relation between *Target* and *Mode* was too restrictive for modelling multi-core applications. Such a containment prescribed in fact that *Mode* elements, representing software, were structurally contained by hardware, represented by *Target* elements. Although not providing a clear separation between software and hardware, this structural containment suited the single-core case, since allocation of software to hardware was not variably splitted across different cores. Modelling for multi-core demanded more flexibility, since allocation of software to hardware is a variability point, which can hardly be represented by a structural containment.

In order to provide such a flexibility, while ensuring backward compatibility with legacy RubusMM models, we have modified the existing hierarchy as follows. We have added the metaclasses *TargetLegacy* and *TargetMulticore*, both inheriting from the abstract metaclass *Target*. *TargetLegacy* represents a single-core ECU and it contains one or more *Mode* elements. This containment is specified through the reference *mode*. *TargetMulticore* represents a multi-core ECU and contains one or more *Core* elements, which in turn can contain *Partition* elements. Both *Core* and *Partition* elements inherit from the abstract metaclass *Allocator*, representing hardware elements to which software elements, represented by the metaclass *Allocatable*), can be allocated. The metaclasses *Allocator* and *Allocatable*, together with the reference *isAllocated*, provide the flexible mechanism for the allocation of software to hardware that we needed, without any structural containment.

The metaclass *Target* provides the following attributes: *speed*, which specifies its speed in MHz, and *type*, which specifies whether it is a *physical* or a *simulated* target and it is used for timing verification purposes. For instance, in case of a simulated *Target*, timing analysis would not make sense as the speed of the *Target* would be equal to the speed of the hosting machine. In this case, timing is verified by simulation rather than by timing analysis.

Both *TargetLegacy* and *TargetMulticore* inherit *speed* and *type*. Moreover, *TargetMulticore* provides additional multi-core specific attributes. *numberOfCores* specifies the number of cores composing the *TargetMulticore* and it is used by the model-based timing analysis and to automatically allocate software to hardware. The reference *core* links *Core* elements to their respective *TargetMulticore*. *Core* may contain *Partition* elements. The attribute *numberOfPartitions* specifies the number of partitions within a *Core* and the reference *partition* links them to the *Core*. The attribute *criticalityLevel* specifies the safety criticality level according to the ISO 26262 standard. There are four criticality levels (A to D) in this standard. A is the lowest criticality level, whereas D is the highest criticality level (the Rubus Kernel supports and is certified for all of them). *Target*, *TargetLegacy*, *TargetMulticore*, *Core*, *Partition*, *Allocator*, *Allocatable*, as well as their attributes and related references were not part of the previous RubusMM definition.

Fig. 2 shows a fragment of the RubusMM containing elements from the *RCM_SW* and the *RCM_COMMON* packages for modelling the software architecture. In RCM a

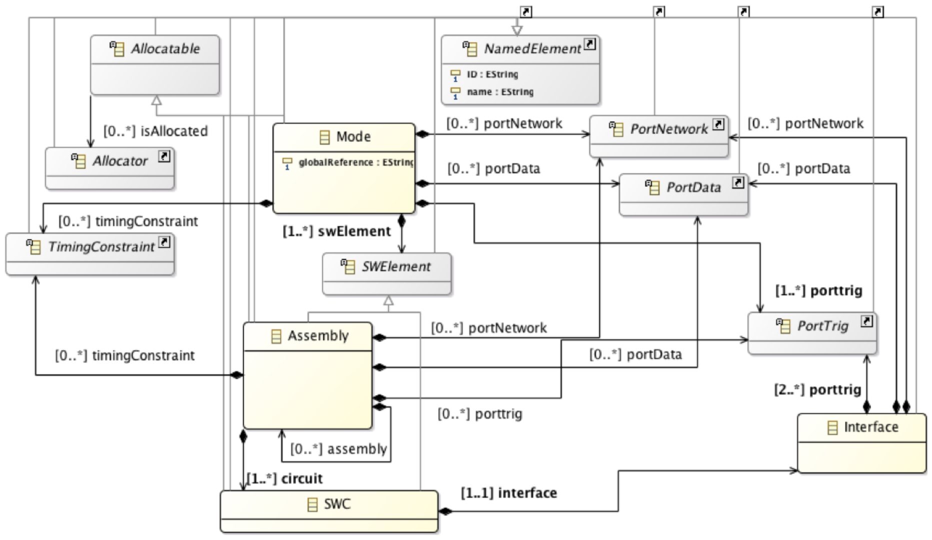


Fig. 2: Fragment of the *RCM_SW* package for modelling the software architecture.

software circuit, represented in RubusMM by *SWC*, is the lowest-level hierarchical element that encapsulates basic software functions. A *SWC* contains one *Interface* which groups all its ports. As RubusMM distinguishes between the data and control flows, an *Interface* containsPortData and *PortTrig* elements. The *PortData* elements manage the data communication among *SWC* deployed on the same *Target*. The *PortTrig* elements manage the activation of the *SWC* elements.

A *PortNetwork* is a port for the data communication of *SWC* elements deployed on different *Target* elements. The *PortData* elements of a *Core* are referenced to the *PortData* elements of the *SWC*s allocated on that *Core*. Similarly, the *PortNetwork* elements of a *Node* are referenced to the *PortNetwork* elements at *SWC* level. An *Assembly* groups *SWC* and *Assembly* elements in a hierarchical fashion.

Its reference *timingConstraint* enables the specification of timing constraints, occurrences and events which are used for timing verification. With respect to the previous definition, *SWC* and *Assembly* have been extended with the inheritance relation from the abstract metaclass *Allocatable*. A *Mode* groups *Assembly* and *SWC* elements and it is used for modelling a specific application of the software architecture (e.g., start-up or error mode). The attribute *globalReference* serves for creating a reference among all the *Mode* elements contributing to the same application. With respect to its previous definition, *Mode* has been extended with the inheritance relation from the abstract metaclass *Allocatable*. The metaclasses *Allocatable* and *Allocator* together with the reference *isAllocated* enable the specification of the allocation of software to hardware. More precisely, an *Allocatable* element can be deployed to an *Allocator* element by setting the *isAllocated* reference. *Allocatable*, *Allocator*, and related references were not part of the previous RubusMM definition.

5 Modelling the Brake-by-wire System

In this section, we leverage the extended RubusMM for modelling the Brake-by-wire (BBW) vehicular application. The BBW system is a stand-alone braking system equipped with an anti-lock braking (ABS) function, which allows to control the brakes through electronic means. To this end, it does not employ any mechanical connection between the brake pedal and the brake actuators. Fig. 3 depicts the block diagram of the BBW system.

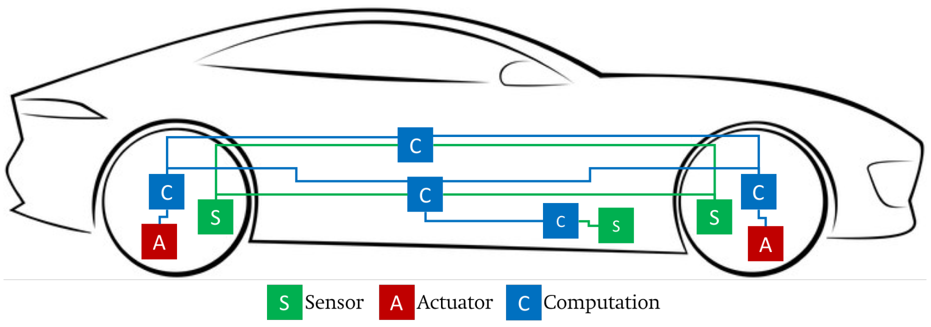


Fig. 3: Block diagram of the BBW system.

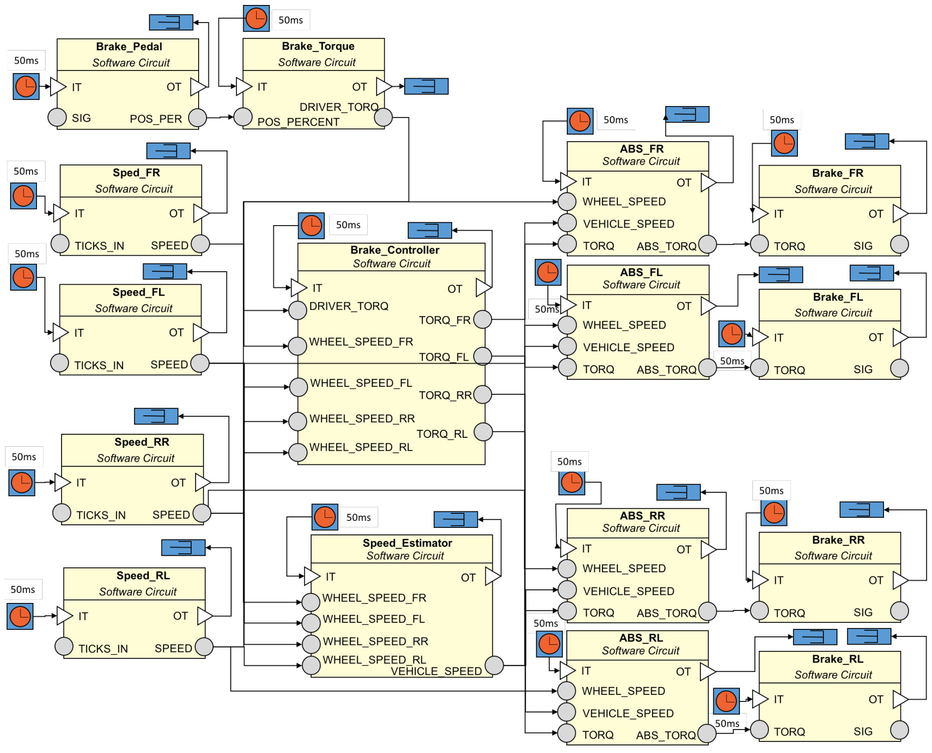
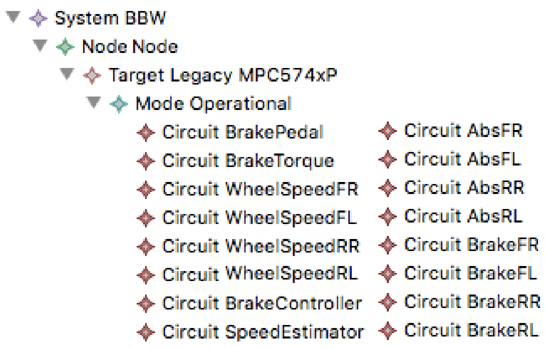


Fig. 4: RubusMM model representing the software architecture of the BBW system.

A sensor, attached to the brake pedal, acquires the signal expressing the position of the pedal. The signal is sent to a computational unit which translates it into a brake torque. A sensor on each wheel acquires the signal expressing the speed of the wheel. The speed of each wheel, together with the computed brake torque, is sent to a computational unit which calculates the brake torque for each wheel. Also, the speed of each wheel is sent to a computational unit which calculates the speed of the vehicle. The speed of the vehicle and the brake torque of each wheel are used from the ABS units for calculating the optimal brake torque for each wheel for avoiding locking the brakes. Finally, the actuators on the wheels produce the actual brake. Fig. 4 shows a RubusMM model depicting the software architecture of the BBW system.

The model consist of 16 software circuits where i) *Brake_Pedal* models the sensor on the brake pedal, ii) *Speed_FR*, *Speed_FL*, *Speed_RR*, and *Speed_RL* model the speed sensors on the wheels, iii) *Brake_Torque*, *Brake_Controller*, *Speed_Estimator*, *ABS_FR*, *ABS_FL*, *ABS_RR*, and *ABS_RL* model computational units and iv) *Brake_FR*, *Brake_FL*, *Brake_RR*, and *Brake_RL* model the actuators on the wheels.

In order to show how the extended RubusMM supports the modelling of multi-core applications (H1), while ensuring backward compatibility with legacy single-core applications (H2), we propose two different deployment configurations. In the first configuration, the BBW system is deployed to a MPC5744Pmicrocontroller, which is a 32-bit unicore microcontroller designed for vehicular applications.

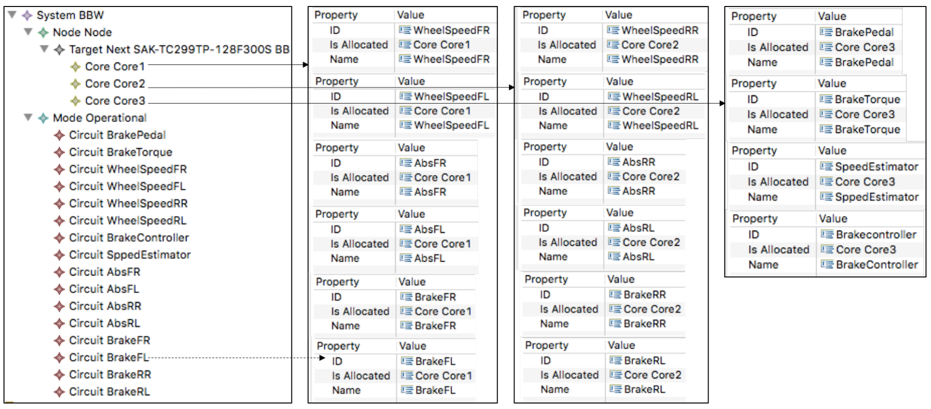


System BBW
Node Node
Target Legacy MPC574xP
Mode Operational
Circuit BrakePedal
Circuit BrakeTorque
Circuit WheelSpeedFR
Circuit WheelSpeedFL
Circuit WheelSpeedRR
Circuit WheelSpeedRL
Circuit BrakeController
Circuit SpeedEstimator
Circuit AbsFR
Circuit AbsFL
Circuit AbsRR
Circuit AbsRL
Circuit BrakeFR
Circuit BrakeFL
Circuit BrakeRR
Circuit BrakeRL

Fig. 5: Serialisation of the BBW system deployed to a unicore microcontroller.

Fig. 5 shows an Ecore serialisation of such a configuration. Note that, according to what described in Section 4 regarding the modelling of legacy applications, the deployment on single-core is expressed leveraging the containment relation between the 'TargetLegacy' *MPC574xP* and the 'Mode' element *Operational*.

In the second configuration, the BBW system is deployed to an Infineon SAK-TC299TP-128F300S BBmicrocontroller, which is a tricore microcontroller developed for applications with high demands of performance and safety.



System BBW
Node Node
Target Next SAK-TC299TP-128F300S BB
Core Core1
Core Core2
Core Core3
Mode Operational
Circuit BrakePedal
Circuit BrakeTorque
Circuit WheelSpeedFR
Circuit WheelSpeedFL
Circuit WheelSpeedRR
Circuit WheelSpeedRL
Circuit BrakeController
Circuit SppedEstimator
Circuit AbsFR
Circuit AbsFL
Circuit AbsRR
Circuit AbsRL
Circuit BrakeFR
Circuit BrakeFL
Circuit BrakeRR
Circuit BrakeRL

Property	Value
ID	WheelSpeedFR
Is Allocated	Core Core1
Name	WheelSpeedFR

Property	Value
ID	WheelSpeedFL
Is Allocated	Core Core1
Name	WheelSpeedFL

Property	Value
ID	AbsFR
Is Allocated	Core Core1
Name	AbsFR

Property	Value
ID	AbsFL
Is Allocated	Core Core1
Name	AbsFL

Property	Value
ID	BrakeFR
Is Allocated	Core Core1
Name	BrakeFR

Property	Value
ID	BrakeFL
Is Allocated	Core Core1
Name	BrakeFL

Property	Value
ID	WheelSpeedRR
Is Allocated	Core Core2
Name	WheelSpeedRR

Property	Value
ID	WheelSpeedRL
Is Allocated	Core Core2
Name	WheelSpeedRL

Property	Value
ID	AbsRR
Is Allocated	Core Core2
Name	AbsRR

Property	Value
ID	AbsRL
Is Allocated	Core Core2
Name	AbsRL

Property	Value
ID	BrakeRR
Is Allocated	Core Core2
Name	BrakeRR

Property	Value
ID	BrakeRL
Is Allocated	Core Core2
Name	BrakeRL

Property	Value
ID	BrakePedal
Is Allocated	Core Core3
Name	BrakePedal

Property	Value
ID	BrakeTorque
Is Allocated	Core Core3
Name	BrakeTorque

Property	Value
ID	SpeedEstimator
Is Allocated	Core Core3
Name	SppedEstimator

Property	Value
ID	Brakecontroller
Is Allocated	Core Core3
Name	BrakeController

Fig. 6: Serialisation of the BBW system deployed to a tricore microcontroller.

Fig. 6 shows an Ecore serialisation of this configuration. In this case, the deployment information is modelled by means of the 'isAllocated' reference expressed between 'Allocatable' and 'Allocator' elements. More precisely, the software circuits modelling the sensors, the computation units and the actuators of the two front wheels (*Wheel-Speed_FR*, *WheelSpeed_FR*, *Abs_FR*, *Abs_FL*, *Brake_FR*, *Brake_FL*) are allocated to *Core 1* of the *SAK-TC299TP-128F300S BB* target, as shown by the arrow in the top-right corner of Fig. 6. Similarly, the SWCs modelling the sensors, the computation units and the actuators of the two rear wheels (*WheelSpeed_RR*, *WheelSpeed_RR*, *Abs_RR*, *Abs_RL*, *Brake_RR*, *Brake_RL*) are allocated to *Core 2* of the *SAK-TC299TP-128F300S BB* target. The remaining SWCs modelling the computational units are allocated to *Core 3* of the *SAK-TC299TP-128F300S BB* target. As discussed in Section 4, the extended RubusMM leverages a clearer separation of concerns between software and hardware elements as well as an explicit and more flexible allocation mechanism. Let us suppose that the allocation specified in Fig. 6 does not satisfy a given set of fault-tolerance requirements. One way of addressing this would be to model a lockstep [27] configuration of the BBW system where each core runs the whole software, in parallel. In order to model such an allocation with the extended RubusMM, it is sufficient to allocate all software circuits composing a 'Mode' to each single 'Core'.

6 Lesson Learned

In this paper, we have proposed an extension to RCM for modelling next generation of vehicular multi-core systems (H1). The main challenge faced during the extension of RCM was how to introduce the new modelling elements without affecting the lastingness of legacy RCM applications (H2). In the first definition of RCM, pragmatic choices for more efficient modelling and analysis of single-core applications were made when defining the language. In addition to not providing clear separation of concerns between hardware and software, these choices complicated the extension of RCM, as in the case of the containment relation between *Target* and *Mode* discussed in Section 4. In fact, that structural containment, although dramatically simplifying model navigation for analysis and code generation purposes in case of single-core applications, did not suit variability of software to hardware allocation in the multi-core case. In this respect, the proposed extension prescribes an allocation mechanism which is more flexible and apt to be automated by means of model transformations. Please note that, we have previously provided RubusMM with support for variability modelling [9]. This feature can be very valuable for representing sets of allocations of software components to multiple cores, all in a single model with variability points representing allocations.

To maximise backward compatibility, we introduced the new modelling elements as leaves in the metamodel hierarchy, as in the case of, e.g., *Core* and *Partition*. This choice could demand additional modelling effort as the engineer can be required to model the entire hierarchy in order to design valid models from scratch. This can be mitigated by tooling features, allowing the modeller to directly model a leaf, while automatically generating the path to the model root populated with a set of default values.

In Section 2, we have pointed out early timing verification as one of the main reasons which made RCM very appreciated in the vehicular domain and its extension for multi-core compelling. In this respect, when extending RCM, we have explicitly addressed timing verification by allowing the specification of timing constraints, occurrences and events at several levels of the structural hierarchy by means of the references *timing-*

Constraint. This ensures full compatibility with the existing model-based timing analysis provided by Rubus. Moreover, it enables the use of the most recent timing analysis for vehicular embedded systems on multi-core [10]. Without the extension provided in this paper, the timing analysis for multi-core would not have been possible in Rubus due to missing structural and timing information.

Functional safety is paramount for the safety criticality of vehicular systems. For being adopted in the vehicular domain, model-based solutions must provided certified run-time support, e.g., real-time operating system, along with modelling languages able to capture all the characteristics of a vehicular application. The Rubus Kernel is certified according to the ISO 26262 standard ASIL D while Rubus ICE (i.e., the development environment supporting Rubus) is undergoing the same certification. In this respect, we have extended RCM according to the virtualisation design option, as described in [5], which enables the reuse of the certified Rubus Kernel. On the one hand, the reuse of the Rubus Kernel makes also the explicit modelling of the memory not necessary since the mapping of data ports to physical memory is handled by the Rubus Kernel itself. On the other had, this makes the current definition of RCM not suited for approaches where explicit modelling of the memory is pivotal. Moreover, despite the Rubus Kernel footprint is significantly small, the virtualised design option increases the overall footprint of the developed vehicular application since each core or partition can host a separate instance of the Rubus Kernel.

7 Conclusion and Future Work

In this paper, we have discussed the extension of the Rubus Component Model for modelling vehicular multi-core applications while ensuring backward compatibility with legacy single-core applications. We have leveraged an industrial vehicular application to validate the proposed extension, also in terms of backward compatibility.

One line of future work will investigate how to support the analysis and verification of vehicular embedded systems with multi-criticality levels on multi-core with respect to predictable timing behaviour. Moreover, we will investigate how to adapt the certified Rubus Kernel for providing run-time support to these systems on multi-core. Another line of future work will investigate how to provide automatic support for the allocation of software to hardware. In particular, we are developing model transformations that, starting from a model with no modelled allocations and a set of timing constraints, produce a set of models featuring the set of different allocations of software to hardware optimised for satisfying the set of timing constraints. We are planning to represent the set of generated models by means of the compact notation presented in [9]. Such a notation uses modelling with variability for representing a multitude of models with one single model with variability points.

Acknowledgments. The work in this paper is supported by the Swedish Knowledge Foundation (KKS) through the PreView and MOMENTUM projects, and by the Swedish Research Council (VR) through the SynthSoft project. We thank our industrial partners Arcticus Systems, Volvo Construction Equipment and BAE Systems Hägglunds, Sweden.

References

1. ISO 26262-1:2011: Road Vehicles in Functional Safety. http://www.iso.org/
2. AUTOSAR Techincal Overview, Version 4.3, The AUTOSAR Consortium, Dec., 2016. http://autosar.org
3. Rubus ICE-Integrated Development Environment, http://www.arcticus-systems.com
4. The UML Profile for MARTE: Modeling and Analysis of Real-Time and Embedded Systems, 2010. OMG Group (January 2010)
5. Becker, M., Dasari, D., Nélis, V., Behnam, M., Miguel, P.L., Nolte, T.: Investigation on AUTOSAR-Compliant Solutions for Many-Core Architectures. In: 18th Euromicro Conference on Digital System Design. vol. 18 (August 2015)
6. Böhm, N., Lohmann, D., Schröder-Preikschat, W.: A Comparison of Pragmatic Multi-core adaptations of the AUTOSAR system. In: 7th annual Workshop on Operating System Platforms for Embedded Real-Time Applications (OSPERT). pp. 16–22 (2011)
7. Bucaioni, A., Cicchetti, A., Ciccozzi, F., Eramo, R., Mubeen, S., Sjödin, M.: Anticipating Implementation-Level Timing Analysis for Driving Design-Level Decisions in EAST-ADL. In: International Workshop on Modelling in Automotive Software Engineering (September 2015)
8. Bucaioni, A., Cicchetti, A., Ciccozzi, F., Mubeen, S., Sjödin, M.: A Metamodel for the Rubus Component Model: Extensions for Timing and Model Transformation from EAST-ADL. Journal of IEEE Access 5(1) (December 2016)
9. Bucaioni, A., Cicchetti, A., Ciccozzi, F., Mubeen, S., Sjödin, M., Pierantonio, A.: Handling Uncertainty in Automatically Generated Implementation Models in the Automotive Domain. In: 42nd Euromicro Conference series on Software Engineering and Advanced Applications (September 2016)
10. Burns, A., Davis, R.: Mixed Criticality Systems - A Review, eighth edition. Tech. rep., Dept. of Computer Science, University of York (2016), https://www-users.cs.york.ac.uk/burns/review.pdf
11. Charette, R.N.: This car runs on code. IEEE Spectrum 46(3), 3 (2009)
12. Ciccozzi, F., Feljan, J., Carlson, J., Crnković, I.: Architecture optimization: speed or accuracy? both! Software Quality Journal pp. 1–24 (2016)
13. Ciccozzi, F., Seceleanu, T., Corcoran, D., Scholle, D.: UML-Based Development of Embedded Real-Time Software on Multi-Core in Practice: Lessons Learned and Future Perspectives. IEEE Access 4, 6528–6540 (2016)
14. Feiertag, N., Richter, K., Nordlander, J., Jonsson, J.: A Compositional Framework for End-to-End Path Delay Calculation of Automotive Systems under Different Path Semantics. In: Proceedings of the IEEE Real-Time System Symposium ? Workshop on Compositional Theory and Technology for Real-Time Embedded Systems, (2008)
15. Feiler, P.H., Gluch, D.P., Hudak, J.J.: The architecture analysis & design language (AADL): An introduction. Tech. rep., DTIC Document (2006)
16. Gamatié, A., Le Beux, S., Piel, É., Ben Atitallah, R., Etien, A., Marquet, P., Dekeyser, J.L.: A model-driven design framework for massively parallel embedded systems. ACM Transactions on Embedded Computing Systems (TECS) 10(4), 39 (2011)
17. Hänninen, K., Mäki-Turja, J., Sjödin, M., Lindberg, M., Lundbäck, J., Lundbäck, K.L.: The Rubus Component Model for Resource Constrained Real-Time Systems. In: 3rd IEEE International Symposium on Industrial Embedded Systems (June 2008)
18. Herrera, F., Posadas, H., Peñil, P., Villar, E., Ferrero, F., Valencia, R., Palermo, G.: The COMPLEX methodology for UML/MARTE Modeling and design space exploration of embedded systems. Journal of Systems Architecture 60(1), 55–78 (2014)

19. ISO 11898-1: Road Vehicles Interchange of Digital Information Controller Area Network (CAN) for high-speed communication, ISO Standard-11898, Nov. 1993.
20. Ke, X., Sierszecki, K., Angelov, C.: COMDES-II: A Component-Based Framework for Generative Development of Distributed Real-Time Control Systems. In: 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), 2007. pp. 199 –208 (August 2007)
21. Lin, C.S., Hsiung, P.A., Chang, C.H., Hsueh, N.L., Koong, C.S., Shih, C.H., Yang, C.T., Chu, W.C.C.: Model-Driven Multi-core Embedded Software Design (2011)
22. Morgan, G., Borg, A.: Multi-core automotive ECUs: Software and hardware implications. Tech. rep., ETAS Group, Tech. Rep (2009)
23. Mubeen, S., Bucaioni, A.: Modeling of Vehicular Distributed Embedded Systems: Transition from Single-core to Multi-core. In: 14th International Conference on Information Technology : New Generations. Springer (April 2017)
24. Mubeen, S., Mäki-Turja, J., Sjödin, M.: Communications-Oriented Development of Component- Based Vehicular Distributed Real-Time Embedded Systems. Journal of Systems Architecture 60(2), 207–220 (2014)
25. Mubeen, S., Nolte, T., Sjödin, M., Lundbäck, J., Lundbäck, K.L.: Supporting timing analysis of vehicular embedded systems through the refinement of timing constraints. Software & Systems Modeling pp. 1–31 (2017)
26. Nicolas, A., Posadas, H., Peñil, P., Villar, E.: Automatic deployment of component-based embedded systems from UML/MARTE models using MCAPI. In: Design of Circuits and Integrated Circuits (DCIS), 2014 Conference on. pp. 1–6. IEEE (2014)
27. Poledna, S.: Fault-tolerant real-time systems: The problem of replica determinism, vol. 345. Springer Science & Business Media (2007)
28. Pop, P., Scholle, D., Hansson, H., Widforss, G., Rosqvist, M.: The SafeCOP ECSEL Project: Safe Cooperating Cyber-Physical Systems Using Wireless Communication. In: Digital System Design (DSD), 2016 Euromicro Conference on. pp. 532–538. IEEE (2016)
29. Reinhardt, D., Kaule, D., Kucera, M.: Achieving a scalable e/e-architecture using autosar and virtualization. SAE International Journal of Passenger Cars-Electronic and Electrical Systems 6(2013-01-1399), 489–497 (2013)
30. Reinhardt, D., Kucera, M.: Domain Controlled Architecture-A New Approach for Large Scale Software Integrated Automotive Systems. PECCS 13, 221–226 (2013)
31. Reinhardt, D., Morgan, G.: An embedded hypervisor for safety-relevant automotive E/E-systems. In: Proceedings of the 9th IEEE International Symposium on Industrial Embedded Systems (SIES 2014). pp. 189–198. IEEE (2014)
32. Sangiovanni-Vincentelli, A., Di Natale, M.: Embedded system design for automotive applications. Computer 40(10) (2007)
33. Schmidt, D.C.: Guest Editor's Introduction: Model-Driven Engineering. Computer 39(2), 25–31 (Feb 2006)
34. Sentilles, S., Vulgarakis, A., Bures, T., Carlson, J., Crnkovic, I.: A Component Model for Control-Intensive Distributed Embedded Systems. In: 11th International Symposium on Component Based Software Engineering (CBSE), 2008. pp. 310–317. Springer (October 2008)
35. Smith, D., Simpson, K.: Functional safety. Routledge (2004)
36. Storey, N.R.: Safety critical computer systems. Addison-Wesley Longman Publishing Co., Inc. (1996)
37. Thorngren, P.: keynote talk: Experiences from east-adl use. In: EAST-ADL Open Workshop, Gothenberg (2013)