Enabling Automated Integration of Architectural Languages: an Experience Report from the Automotive Domain

Alessio Bucaioni[a], Matthias Becker[b]

[a] *Mälardalen University, Västerås, Sweden*
[b] *KTH Royal Instituite of Technology, Stockholm, Sweden*

Abstract

Modern automotive software systems consist of hundreds of heterogeneous software applications, belonging to separated function domains and often developed within distributed automotive ecosystems consisting of original equipment manufactures, tier-1 and tier-2 companies. Hence, the development of modern automotive software systems is a formidable challenge. A well-known instrument for coping with the tremendous heterogeneity and complexity of modern automotive software systems is the use of architectural languages as a way of enabling different and specific views over these systems. However, the use of different architectural languages might come with the cost of reduced interoperability and automation as different languages might have weak to no integration. In this article, we tackle the challenge of integrating two architectural languages heavily used in the automotive domain for the design and timing analysis of automotive software systems: AMALTHEA and Rubus Component Model. The main contributions of this paper are i) a mapping scheme for the translation of an AMALTHEA architecture into a Rubus Component Model architecture where high-precision timing analysis can be run, and the back annotation of the analysis results on the starting AMALTHEA architecture; ii) the implementation of the proposed scheme, which uses the concept of model transformations for enabling a full-fledged automated integration; iii) the application of such automation on three industrial automotive systems being the brake-by-wire, the full blown engine management system and the engine management system. We discuss and evaluate the proposed contributions using an online, experts survey and the above-mentioned use cases. Based on the evaluation results, we conclude that the proposed automation mechanism is correct and applicable in industrial contexts. Besides, we observe that the performance of the automation mechanism does not degrade when translating large models with several thousands of elements. Eventually, we conclude that experts in this field find the proposed contribution industrially relevant.

Keywords: Architecture languages, Model-based development, timing verification.

1. Introduction

Automotive software was born less than 45 years ago when General Motors employed a single function controller for the electronic spark in their Oldsmobile Toronado. In less than five decades, the complexity of automotive software and its development has grown remarkably [1]. Modern automotive software systems consist of hundreds of heterogeneous applications, belonging to different functional domains and developed in parallel within huge ecosystems involving tens of automotive Original Equipment Manufactures (OEM), tier-1 and tier-2 companies [2] [3]. In addition, these applications are integrated into heterogeneous computing platforms and need to comply with stringent quality attributes such as performance and timing [2] [3].

To tackle the tremendous complexity of modern automotive software systems, both practitioners and researchers have promoted the use of multiple views for the development of these systems [4]. Views should be linked using suitable relationships, which are fundamental to understand the impact of architectural decisions. Each view can be described by one or more Architectural Languages (ALs) [5] [6] for easing the design, communication and analysis of automotive systems [7]. Functional[1] ALs are typically used for designing automotive software systems as they focus on the system decomposition in components and interactions among components [8]. Technical ALs, instead, describe automotive software systems from the realisation perspectives (e.g., execution environment, hardware platform and software to hardware allocation) hence they are mostly employed for enabling model-based non-functional analyses of the automotive software systems [9]. In this context, interoperability among different ALs is not only desirable but pivotal for benefiting from the advantages introduced from the use of ALs such as increased abstraction,

Email addresses: alessio.bucaioni@mdh.se (Alessio Bucaioni), mabecker@kth.se (Matthias Becker)

[1] We assume that the reader is familiar with the concepts of functional, logical and technical architectures defined in [8].

separation of concerns and reduced overall complexity [10]. In addition, the lack of mature tools support [10] and proper integration mechanisms among different ALs represent one of the main factors hampering the full-fledged adoption of these languages in industry [7].

An example of this is the joint use of so-called functional, logical and technical ALs. While the former is typically used for designing purposes (as they reflect the decomposition of software systems into sets of components and interactions among them), the latter is commonly used for non-functional analyses of software systems (as they describe software systems from the realisation-perspective, e.g., execution environment, hardware platform and allocation). According to the empirical study by Malavolta et al., one of the main reasons why practitioners use ALs is for (automated) non-functional analysis such as timing analysis [7]. However, this is hampered by the lack of integration among the languages as different ALs have weak or no integration with each other [7]. As a consequence, these languages are often integrated manually using engineers educated guesses. Malavolta et al. have identified this as one of the major challenges slowing down the full-fledged adoption of ALs in industry [7].

In this work, we tackle the challenge of integrating two industrial ALs heavily used within the automotive domain: AMALTHEA [11] and Rubus Component Model (RCM) [12]. AMALTHEA is an automotive AL, which has been defined in 2011 within the European project AMALTHEA [13]. Since 2011, AMALTHEA has been constantly refined through several European projects including AMALTHEA4public [14] and PANORAMA [15]. Currently, AMALTHEA is used by several automotive OEMs and suppliers in the value chain mostly as industrial exchange format. RCM has been defined in 2008 within the collaborative research project MultEx [16] and builds on the concepts presented by Hansson et al. [17]. RCM support the timing analysis and synthesis of distributed real-time embedded systems and it is mostly used within the automotive value chain from international OEMs and suppliers such as Volvo AB, BAE Systems plc, Knorr-Bremse AG. The interest for and relevance of such integration has emerged within several research projects [18] including the European project PANOARAMA, which involves more than 20 partners among research institutions and companies from 5 different countries [15] [19]. Within PANORAMA, we have observed that several automotive OEMs and suppliers employ a development process, which leverages AMALTHEA, for the automotive system and software design, and RCM for high-precision timing analysis (e.g., response-time analysis [20], distributed end-to-end response-time and delay analyses [21]), as graphically described by Figure 1. There are several factors, which justify the interplay between AMALTHEA and RCM. AMALTHEA is a large and flexible AL, which provides dozens of elements for the representation of the software system, the hardware abstraction, the software to hardware allocation, the operating system abstraction, and different kinds of events and constraints.

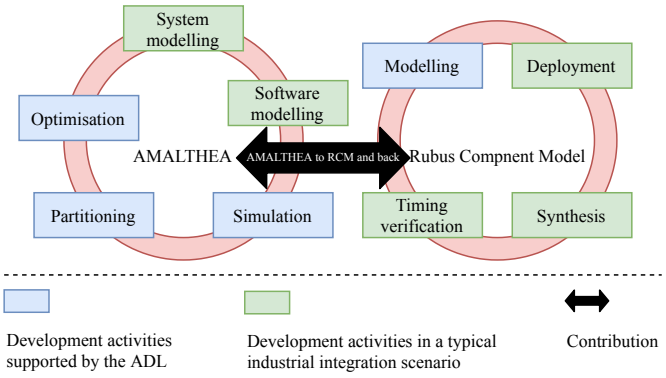

Figure 1: Architecture languages, development process and contribution of this work.

At times, such a wealth of elements makes AMALTHEA less suited for performing timing verification of automotive software systems. To have meaningful results, the engineer would be required to have an extensive knowledge of AMALTHEA and to describe several low-level details such as, e.g., operating system. RCM is a rather concise and pragmatical AL, which has been purposely defined for supporting early high-precision timing analysis. RCM features an accompanying real-time operating system (RTOS) and development environment certified according to the ISO 26262 international standard for functional safety of electrical and/or electronic systems in production automobiles [22]. Using RCM, the engineer does not need to explicitly represent low-level details as memory hierarchies or the operating system as this information is abstracted by the accompanying RTOS and development environment.

In this work, we extend our earlier work that described an initial mapping from Amalthea to RCM [23]. Specifically, we report on our experience in integrating AMALTHEA and RCM as a way for improving the design and timing verification of automotive software systems. The main contributions of this work are:

- a mapping scheme enabling i) the translation of an AMALTHEA architecture into an RCM compliant architecture where high-precision timing analysis can be run, and ii) the back-annotation of the analysis results in the AMALTHEA architecture,
- a realisation of the mapping scheme for automatically performing the translations using the concept of model transformations [24]
- the application of the automation mechanism to the Brake-By-Wire (BBW), the Full Blown Engine Management system (FBEM), and the Engine Management System (EMS) industrial use cases.

The work described in this paper has been carried out within the research project PANORAMA [15] in close collaboration with automotive OEMs and suppliers. We have used a research methodology, which is an adaptation of the model by Gorschek et al. for industry-relevant research in software engineering [25]. The proposed mapping has

been defined building on the knowledge and best practices acquired during PANORAMA and the engine control system use case presented by Frey et al. [26]. The automation mechanism is realised as proof-of-concept implementation and allows to translate and back-annotate large scale industrial-sized applications.

We have assessed the applicability of the mapping scheme and the automation mechanism using the BBW use case. We have assessed the correctness of the mapping scheme and automation mechanism by comparing the automatically generated RCM model for the BBW use case with an RCM model built manually from an engineer having more than 10 years of experience with RCM. We have discussed the scalability and the performance of the automation mechanism using the FBEM and the EMS use cases. The FBEM use case was proposed by Robert Bosch GmbH within the 2017 WATERS Industrial Challenge of the Euromicro Technical Committee on Real-Time Systems [27][2], while the EMS use case is described by Frey in [26]. Eventually, we have assessed the industrial relevance of this research using expert interviews and online workshops.

The remainder of this paper is structured as follows. Section 2 describes the background for this work and its relation with our previous works. Section 4 presents the mapping between AMALTHEA and RCM and its realisation. Section 6 describes the application of the proposed mapping scheme on the BBW use case. Section 7 presents a discussion on the correctness, scalability, performance and industrial relevance of this work. Section 8 presents related work documented in the literature. Finally, Section 9 concludes the paper with final remarks and future work.

2. Background

In this section, we describe the background of this research. In particular, we discuss the main architectural elements of AMALTHEA and RCM. Besides, we describe the importance of timing verification for the development of automotive software systems and how timing verification is supported by current ALs.

2.1. AMALTHEA

AMALTHEA is an automotive-specific language, which focuses on the design, implementation and optimisation of automotive software for multi-core systems. AMALTHEA has been defined and extended through several European research projects including AMALTHEA, AMALTHEA4public and PANORAMA, and it is developed and maintained by a wide consortium of automotive companies and research institutes. AMALTHEA supports the design, implementation and optimisation of automotive software employing two main data models, which are the system model and the trace model. In turn, the system model is organised in different data models. In this work, we focus

[2] https://www.ecrts.org/archives/index4bb2.html?id=277

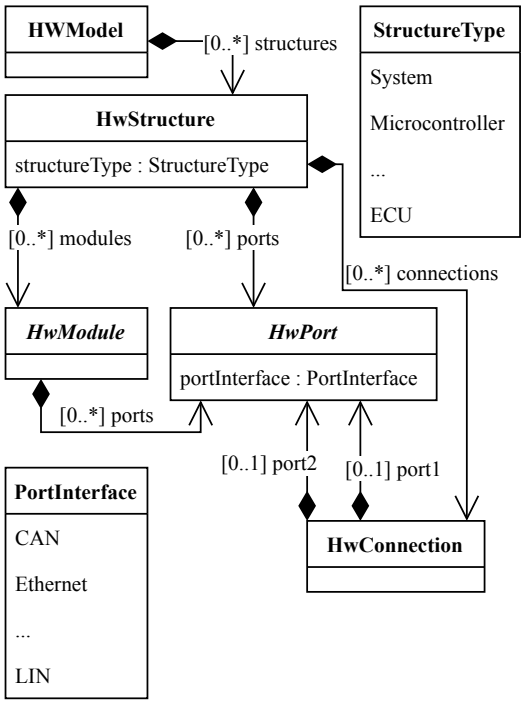

Figure 2: Simplified representation of the AMALTHEA hardware model.

on hardware, mapping, operating system, software, events, stimuli, constraints and measurement data models.

The hardware data model describes the hardware platform of the systems using `HwModule` and `HwConnection` elements. The former is used for representing electronic control unit (ECU), processing units, etc. while the latter represent communication means. In AMALTHEA, `HwModule` and `HwConnection` elements communicate via ports. AMALTHEA supports different kind of communication protocols including Ethernet [28], CAN [29, 30], etc. Figure 2 shows a simplified representation of the main hardware elements and their relationships. The software data model describes the automotive software in terms `Runnable` and `Task` elements and connections among them. `Runnable` elements represent basic software units. `Task` elements represent executable units that can be managed by an operating system scheduler. Hence, `Runnable` elements are refined into one or more `Task` elements. The communication among software units is represented using the `Label` and `Channel` elements. Both represent data in the memory, but `Channel` elements can hold multiple data elements where `Label` elements hold only the most recent data (last-is-best semantic). Figure 3 shows a simplified representation of the relationships among `Runnable`, `Channel` and `Label` elements. Software units can be activated periodically or sporadically. Such information is specified using the stimuli data model. The main elements of this data model are `Stimulus` and `Clock`. The events data model describes tracing configurations, event chains and some timing constraints. There are different event elements for the

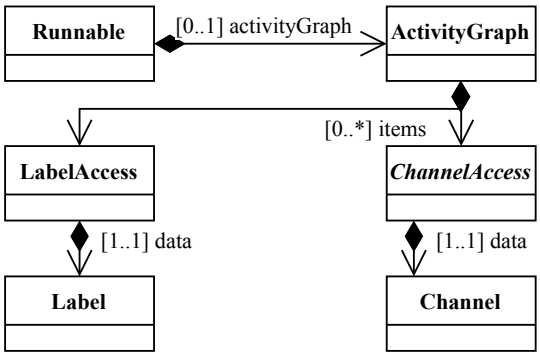

Figure 3: Simplified representation of the relationships among Runnable, Channel and Label elements.

different elements that can be traced. Examples of event elements are: `RunnableEvent`, `LabelEvent`, `ChannelEvent`, and `ProcessEvent`. Hence event elements can be linked to different software and hardware elements. The constraints data model specifies different constraints including runnable-sequencing-constraints, affinity constraints and timing constraints. `RunnableSequencingConstraint` elements define an order for the specified runnable elements. `AffinityConstraint` elements constraint the mapping of runnable and other software elements. Timing constraints restrict the time span between events or the duration of event chains and are expresses using different elements including `DataAgeContraint`, `EventChain` elements and others. The mapping data model describes allocation information of `Runnable`, `Task`, `Interrupt`, `Task scheduler` and other elements. This is achieved using specific elements such as `RunnableAllocation` elements for the allocation of `Runnable` to `Scheduler` elements. In addition, the mapping data model specifies how the software is mapped to the memory using the `MemoryMapping` and `PhysicalSectionMapping` elements. The operating system data model describes the main functionalities of a RTOS as well as the access to system resources. The main elements are `OperatingSystem`, `OsOverhead` and `Semaphore`, where OperatingSystem element contains `TaskScheduler` and `InterruptController` elements.

The measurement model provides the possibility to store measured times. In this research, we use the measurement model for enabling the back annotation of the timing analysis results from Rubus ICE. In particular, we use the `EventChainMeasurament` and `TaskMeasurament` elements for storing the measured time for event chains and task, respectively. For both the elements, we create three custom properties `Worst-Case Reaction Time`, `Worst-Case Response Time` and `Creator` for storing the type of the recorded results and their origin, respectively.

2.2. RCM

RCM is an industrial AL specifically defined for easing and supporting the timing analysis and synthesis of distributed real-time embedded software systems. It is developed and maintained by Arcticus Systems with the collaboration of several research institutes such as the KTH Royal Institute of Technology and Mälardalen University. RCM and its accompanying RTOS and development environment have evolved over the years through several national and European research projects. Currently, RCM is used by several automotive manufacturers like Volvo Group, BAE Systems, Hoerbiger, Knorr Bremse and others for the development of predictable and resource-constrained embedded software systems [31].

The purpose of RCM is to express the infrastructure for software functions. To this end, RCM features four main data models, which are hardware, software, allocation and timing. The hardware data model provides for the description of the hardware platform in terms of its processing units and network busses. The software data model is used for describing software systems in terms of software functions and connections among them. The allocation data model provides for the specification of software-to-hardware allocation constraints. The timing data model allows expressing real-time requirements and properties on the software architecture. As mentioned in Section 1, RCM describes the hardware platform in terms of processing elements abstracting from low-level details such as memory hierarchies or the operating system, which are taken care from the accompanying RTOS and development environment. Hence, the hardware abstraction is described using `Node`, `Core`, `Partition`, and `Network` elements. Processing units are represented using `Node` elements. Their internal structure is described using `Core` and `Partition` elements. `Core` elements represent physical cores while `Partition` elements represent the logical partitions of cores. `Network` elements encapsulate network specification details as well as protocol stack information of different network communication protocols such as the Time Sensitive Networking (TSN) [32], CAN and its higher-level protocols [29, 30].

In RCM, software functions are encapsulated from `Software Circuit` (SWC) elements. `SWCs` are the lowest-level hierarchical elements and contain `Behaviours` and an `Interface` elements. `Behaviour` elements represent the functional behaviour of functions and allow for the specification of real-time properties of the `SWCs` such as worst-average- and best-case execution times, and maximum stack usage. `Interface` elements are used for grouping `SWC` ports. `SWCs` interaction is clearly separated in data and control flows for facilitating the definition of the control specification and interactions, typical of real-time embedded systems. Data and control flows are specified using connectors and ports of SW. This means that interface elements contain both `Data` and `Control` ports. `SWCs` can be activated through their control ports. RCM supports both periodic and sporadic activation. Periodic activation is modelled with `Clock` elements while the most common sporadic activation is activation from a preceding `SWCs`. `Assembly` elements are only used for grouping `SWC` elements in a hierarchical fashion. `Mode` elements group assembly and `SWC` elements and are used for modelling specific configura-

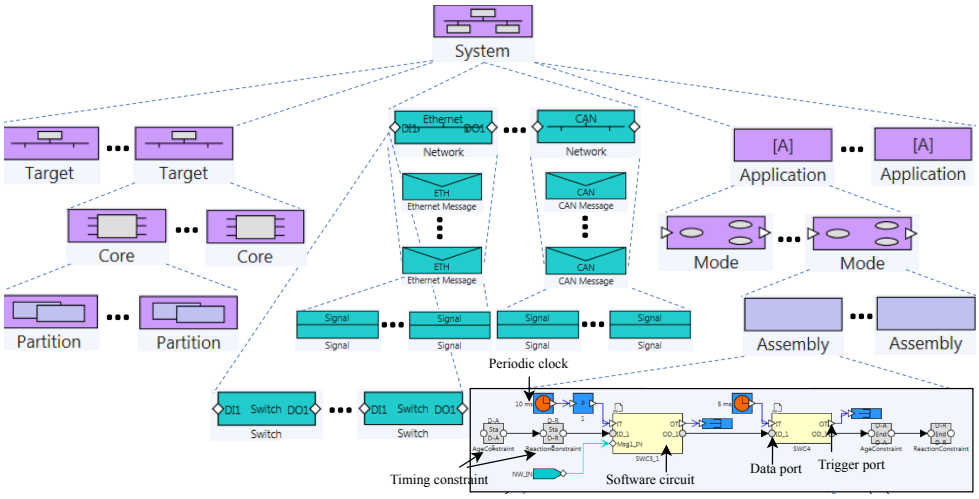

Figure 4: RCM main elements and relationships.

tions of the software system such as start-up or error mode. `Mode` elements are grouped into `Application` elements, where an `Application` element represents an independent software functionality in the system under development such as steer-by-wire application. Figure 4 shows the main RCM elements and their relationships using the concrete syntax of the accompanying RCM development environment. Software to hardware allocation constraints are specified using relations between software and hardware elements. The `isAllocated` reference can be specified for any `SWC`, `Assembly` and `Application` element towards any `Partition`, `Core` and `Node` element.

2.3. Timing

Amalthea, as well as RCM, allows for the specification of timing properties. The timing properties that can be specified in both ALs are rooted in the Timing Augmented Description Language TADL2 [33]. Amalthea allows specifying the timing constraints on events, either for single tasks/runnables or for the data propagation trough an event chain. The event chain concept is a direct result of TADL2, where event chain items are defined by a stimuli and a response event. The chain is then a result of several event chain items that connect over shared events for response and stimuli respectively. The data age constraint or the reaction constraint can then be specified for the event chain [34]. Classical real-time constraints such as deadlines and jitter can be specified as well. Similarly, RCM allows specifying timing constraints on event chains. Here, the `AgeDataEnd` and `AgeDataStart` or `ReactionDataEnd` and `ReactionDataStart` element can be connected to data ports of the `SWC`. The event chain is then implicitly defined by the data path in the model. A detailed discussion of the representation of TADL2 timing constraints in RCM is provided in [35].

The RCM model allows performing high-precision timing analysis. This is possible through the Rubus RTOS and well defined synthesis from RCM model to the final code. On `SWC`-level, the analysis engine then implements tight response time analysis for tasks with offsets [36]. Holistic timing analysis is implemented for task chains that are distributed over several compute nodes [37]. And end-to-end delays of effect chains, such as data age or reaction delay, are available as well [38] based on the analysis of [34].

3. Research method

We have conducted this study using an adaptation of the model for industry-relevant research in software engineering proposed by Gorschek et al. [25]. The adapted model

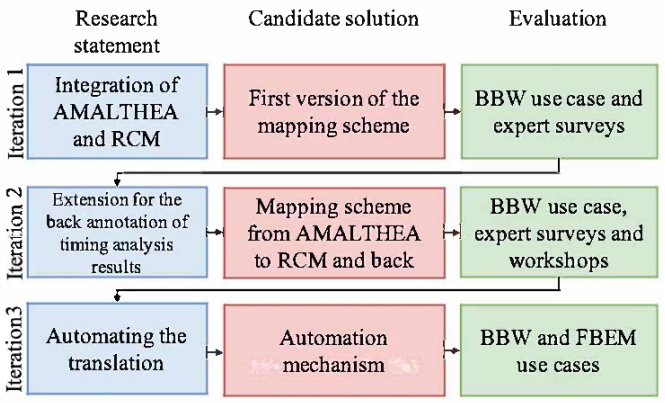

Figure 5: Adopted research method.

consists of 3 main steps carried out iteratively. In three iterations, we have developed and evaluated the mapping scheme and the automation mechanism. Figure 5 provides a graphical representation of the steps and iterations. As a first step, we have observed domain and business settings and assessed current industrial practices with the aim of defining a research statement aligned with industrial needs. Such an observation has happened in the context of the PANORAMA project and involved several national and

international automotive companies. Integrating AMALTHEA and RCM has emerged as a pressing concern that was of interest for both the practitioners and the research community. In the next step, we have developed a first architectural mapping between AMALTHEA and RCM. We have constructed the mapping using relevant sources and use cases. In this step, we have collaborated with two automotive companies being Arcticus Systems in Sweden and Robert Bosch GmbH in Germany. We have evaluated the applicability of the mapping scheme using the BBW use case. Besides, we have assessed the relevance of this work using expert surveys. The survey contains a set of 10 questions drawing on the model for evaluating research rigour and relevance presented in [39]. The pool of interviewed experts includes 10 researchers and practitioners with relevant experience in the fields of cyber-physical systems, software engineering, software architecture, model-driven engineering and real-time embedded systems. More details on the survey are provided in Section 7. The application of the mapping scheme on the BBW use case has proved its applicability. The survey has shown that there was a consensus on the usefulness and relevance of the mapping scheme. It has also pointed some limitations and possible improvements. The limitation that has concerned the most the respondents was that the proposed mapping scheme did not contain relationships for back annotating the timing analysis results on the AMALTHEA architecture. Hence, in the second iteration, we have focused on identifying the architectural elements for enabling the back annotation of the original AMALTHEA architecture with timing analysis results. The result was an extended mapping scheme, which we have validated on the same BBW use case. Another concern we have collected from the expert survey was regarding the size of the use case, which was considered as relevant only by half of the respondents. We have addressed this feedback by introducing other use cases, the FBEM and the EMS use cases, which have helped us in evaluating the scalability and performance of the automation, too. To collect further feedback on the usefulness of this research, we have presented the proposed mapping scheme to the broader community during two online workshops. In the first workshop, we have discussed our effort together with the research community of the Euromicro Conference on Software Engineering and Advanced Applications. The participants were mostly academics with different titles and job positions, from PhDs to Professors. This workshop has lasted for 40 minutes and has involved In the second workshop, we have discussed our research with the partners from the PANORAMA project. The attendees of this workshop were a mix of academics and professionals researching and working in automotive software engineering. Among others, we had representatives from Arcticus Systems, Robert Bosch GmbH, Alten, SAAB AB, KTH Royal Institute of Technology, Siemens AG, University of Gothenburg, University of Rostock, etc. This workshop has lasted for one hour. During both workshops, it was suggested to use the mapping scheme for defining an automation mechanism able to integrate AMALTHEA and RCM. In the third and last iteration, we have focused on developing such an automation mechanism using the concept of model transformation. Besides, we have focused on validating the applicability, correctness, scalability and performance of the proposed automation mechanism. We have validated the applicability and correctness of the automation mechanism using the BBW use case. For assessing the correctness, we have compared the automatically generated RCM model of the BBW system with an RCM model created manually by a software engineer with more than 10 years of experience with these technologies. To evaluate scalability and performance we have used the FBEM and EMS use cases.

3.1. Threats to validity

Hereafter, we discuss and classify potential threats to validity and describe our mitigation strategies according to the scheme proposed by Runeson et al. [40].

3.1.1. Threats to internal validity

Threats to internal validity affect the relation among the observed variables. To mitigate the threat to treatment setup and design, we have carried out this research by following the model for industry-relevant research in software engineering and complemented it with domain knowledge, best practices and experiences acquired during our consolidated collaborative research experience. To mitigate threats related to subjects selection, we have involved only subjects with extensive and proven experience in the field. All the data and use cases in this research have been provided from our industrial partners or taken from openly available industrial use-cases such as the ones provided trough the WATERS Industrial Challenge [27] and the work by Frey [26]. This has helped us in mitigating threats to treatment design and sample selection. Besides, we have mitigated the lack of clear data collection procedure threats by describing all the steps we have followed for the design and validation of this research. We have made publicly available all the artefacts used in this research, i.e., use cases, code of the automation mechanism and the survey. This has helped us in mitigating possible threats to poor parameter settings and lack of discussion on development and testing artefacts. To mitigate the risk that the questions composing the survey might have influenced the answers, we have built the survey on the model for evaluating research rigour and relevance presented by Ivarsson et al. [39]

3.1.2. Threats to construct validity

Threats to construct validity related to the design of the study. To minimise such threats, we have adopted a well-defined research protocol based on the model for industry-relevant research in software engineering. We have mitigated mono-operation bias using different validation means, i.e., use cases, survey and workshops, and subjects.

To minimise threats related to experimenter bias and expectations, we have made an effort in relying on well-known and accepted methods such as the one by Ivarsson et al. [39]. In this research, we have run the proposed mechanism on real-world use cases coming from industry. This has helped us in reducing threats to the appropriateness of data. All researchers involved in this work have prior and established experience in the automotive domain, which has helped in ensuring construct validity. It is worth mentioning that all the authors have longstanding professional collaborations with the experts involved in the evaluation and this has resulted in insightful discussions characterised by mutual trust.

3.1.3. Threats to external validity

Threats to external validity affect the generalisation of the observed results. The main threats to external validity for this research are the representation of the population and settings. To mitigate threats to the representation of the settings, we have made an effort in using use cases and tools representative of the actual domain. In particular, we have leveraged three real-world use cases, which we have manipulated using the official accompanying tools of the languages, being APP4MC and Rubus ICE. To mitigate threats to the representation of the population, we evaluated this research only with profiles having extensive and proven experience in the field of automotive software engineering. Besides, both the authors have a long track record in such a field proven both by the different research projects they run and by their professional experience.

3.1.4. Threats to conclusion validity

Threats to conclusion validity affect the ability to derive conclusions from the observed relationships. One of the main threats to conclusion validity for this research is the reliability of the measures as the research results could have been affected by the quality of the leveraged use cases and survey questions. We have addressed this threat by using real-world industrial use cases and by building the survey on a well-known model for evaluating research rigour and relevance presented by Ivarsson et al. [39]. Another important threat to conclusion validity is the statistical power as the survey was answered by 10 respondents. To mitigate such a threat, we have complemented the expert survey with two online meetings involving more than 40 participants, consisting of researchers and practitioners in software engineering from well-known and reputable universities and companies. We have tried to mitigate other threats to conclusion validity such as lack of expert evaluation and fishing for results by involving only profiles with an extensive and proven experience and by not contaminating the validation activities with our expectations.

4. From AMALTHEA to RCM and back

In this section, we present a mapping scheme for the integration of AMALTHEA and RCM. The proposed mapping scheme allows for the translation of an AMALTHEA architecture into an RCM one, where high-precision timing analysis can be performed. Besides, the mapping allows for the back-annotation of the timing analysis results on the starting AMALTHEA architecture. Table 1 summarises the main relationship composing the proposed mapping. Each relationship relates one RCM element to the corresponding AMALTHEA element(s) from which it can be translated. The wealth of elements of AMALTHEA means that the elicited relationship might not be unique as RCM elements may be translated from different AMALTHEA ones. We believe that the existence of such alternatives does not affect the correctness and effectiveness of the proposed mapping. We discuss this aspect in Section 7.

4.1. From AMALTHEA to RCM

In RCM, a `Node` element represents a computing processor. An RCM `Node` element can be translated from an AMALTHEA `HwStructure` element when its `structureType` is typed to either ECU or micro-controller. Figure 6 shows an example of an AMALTHEA architecture where the `HwStrcuture NXP_MPC5744P` [41] is typed to a microcontroller.

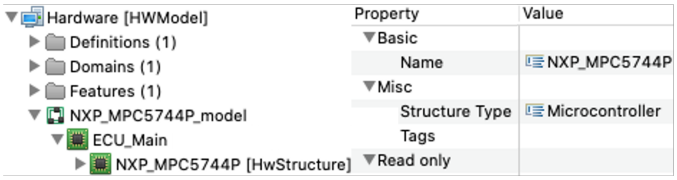

Figure 6: Extract of an AMALTHEA architecture depicting an HwStructure element of type Microcontroller.

An RCM `Core` element describes a processing unit and it can be translated from an AMALTHEA `ProcessingUnit`. In this case, the associated `ProcessingUnitDefinition` must have the attribute `puType` set to CPU. In Figure 7 the element CPU_type is typed to CPU. From a semantic perspective, these two relationships allow preserving the structural containment between `HwStructure` and `ProcessingUnit` elements, and between `Node` and `Core`. Figure 7 shows the containment between `NXP_MPC5744P` and `CPU_type`.

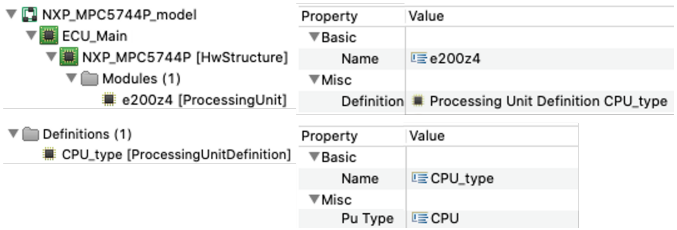

Figure 7: Extract of an AMALTHEA architecture depicting an HwStructure element, a ProcessingUnit element and their relationship.

Generally, RCM does not require to explicitly model memory hierarchies as the allocation of code to data memory is subject to automated synthesis and not entrusted to

Scope	RCM	AMALTHEA
Hardware	Node	HWModel::HwStructure
	Core	HWModel::ProcessingUnit
	Network	HWModel::ConnectionHandler
	Message	SWModel::Channel and SWModel::Label
Operating system	Partition	OSModel::TaskScheduler
Software	Application	SWModel
	Mode	SWModel::Mode
	Software circuit	SWModel::Runnable
	Data port	SWModel::Channel and SWModel::Label
	Clock	StimuliModel::PeriodicStimulus
	Activation from predecessor	StimuliModel::SporadicActivation and SWModel::ActivityGraph
	Priority	MappingModel::TaskAllocation
	Worst-case execution time	SWModel::Ticks and HWModelFrequency
Mapping	Software to hardware allocation	MappingModel::SchedulerAllocation
Timing	DataAge	ConstraintModel::EventChainLatencyConstraint
	DateReaction	ConstraintModel::EventChainLatencyConstraint

Table 1: Mapping relationships.

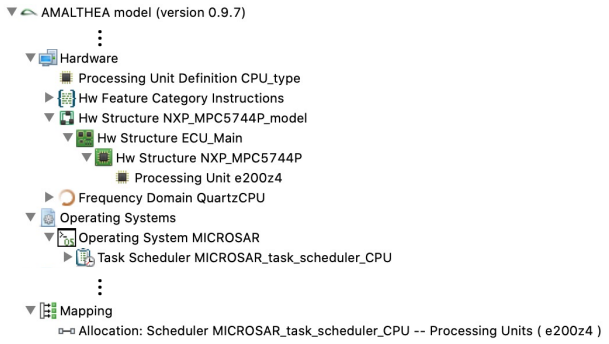

Figure 8: Extract of an AMALTHEA architecture depicting a `ProcessingUnit` element, a `TaskScheduler` element and the `SchedulerAllocation` element.

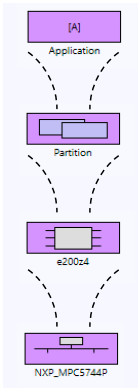

Figure 9: Extract of an RCM architecture depicting a `Partition` element, a `Core` element and a `Node` element.

the engineer. An exception to this is the description of the communication network where RCM requires the explicit modelling of the network and all the messages travelling through it. In RCM, a `Network` element represents the network connecting different nodes of the architecture. A `Network` element can be translated from the AMALTHEA `ConnectionHandler` element. An RCM network `Message` represents data exchanged from software elements residing in different nodes of the architecture. Hence, it can be translated from an AMALTHEA `Label` or `Channel` element, when it refers to `Runnable` elements in different `HwStructure` elements.

In RCM, a `Partition` element represents a logical division of cores and it is used for grouping software functions to schedulable units. It can be translated from an AMALTHEA `TaskScheduler` element. One important aspect to consider when translating `TaskScheduler` elements are the allocation information: in AMALTHEA, such information is specified using the `SchedulerAllocation` element, while in RCM a `Partition` element is structurally contained within a `Core` element. Hence, the information modelled by the `SchedulerAllocation` element must be taken into account for creating `Partition` elements within the proper `Core` elements. An example is shown in Fig. 8. The example includes an e200z4 CPU core that is modelled as `ProcessingUnit` element, a MICROSAR task scheduler modelled as `TaskScheduler` element and the `SchedulerAllocation` element that maps the scheduler to the processing unit. Fig. 9 depicts the RCM elements `Node`, `Core`, `Partition` and `Application` of the AMALTHEA model in Fig 8. The structural containment of the elements is visualised in the figure. An RCM `Application` element is used for grouping software functions and specifying its safety-related properties. It can be translated from an

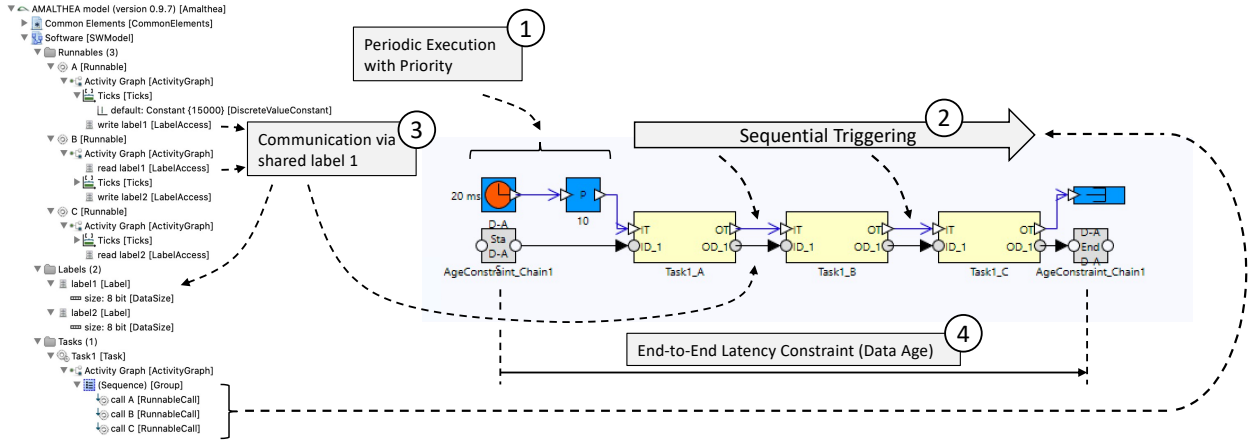

Figure 10: Extract of an AMALTHEA architecture and its RCM representation. Highlighted are the periodic execution with fixed priority (marking 1), mapping of runnable execution from AMALTHEA tasks to sequential execution in the RCM model (marking 2), communication via shared labels in the AMALTHEA model and mapping to communication via data channels in the RCM model (marking 3), and representation of end-to-end timing constraints in the RCM model (marking 4).

AMALTHEA `SWModel` element. RCM `Mode` elements RCM can be translated from the AMALTHEA `Mode` elements. In RCM, an SWC element describes an elementary unit of computation. In AMALTHEA, a `Runnable` element represents an abstraction of an executable entity and a `Task` element represents an executable unit. Each AMALTHEA task needs to specify the sequence of the executable entity using the `Activity Graph` element. Hence, we believe that RCM SWCs should be translated from the `Runnable` elements contained within an `Activity Graph`. An example of this is provided in Figure 10, which shows the example task `Task1`, its `Activity Graph` and the `Runnable` elements `A`, `B` and `C` in the graph, which in turn exchange data via the communication labels `label1` and `label2`. The example task is periodically activated with a period of 10 ms and a fixed priority of 10 (marking 1). The runnables that are executed by `Task1` are instantiated in the RCM model and their trigger ports are connected to form a sequential execution (marking 2). Communication in the AMALTHEA model is realised via shared labels that are accessed from the `Runnables` with either read or write access. These are translated to `data ports` and their connection in the RCM model (highlighted for the example of `label1`, marking 3).

It is important to note that since a single `Runnable` element can be referred from different `Activity Graph` elements, SWCs should be translated with proper identifiers. In RCM, some important properties of a SWC are the `Priority` and the `Worst-Case Execution Time` (WCET). These properties can be translated from an AMALTHEA architecture. The former can be translated from the priority parameter of AMALTHEA `TaskAllocation` elements. The WCET of a runnable can be calculated using the attribute `Ticks` of the `Runnable` elements and the attribute `Frequency` of `FrequencyDomain` elements used to denote the CPU frequency, as follows:

$$WCET = Ticks/Frequency$$

For instance, if the CPU frequency is defined to be 300 MHz, the execution time of the runnable `A` in Figure 10 can be computed by dividing 15000 ticks by 300 MHz.

As mentioned above, RCM `Data port` elements can be generated from AMALTHEA `Label` and `Channel` elements depending on the size of the data. If the AMALTHEA architecture does not contain `Label` or `Channel` elements, RCM `Data port` elements can be derived from `Event Chain` and `Event` elements. In RCM, `Control port` elements are mandatory and are used for specifying the activation of SWC elements, which can be either periodic or sporadic. Periodic activation is modelled using `Clock` elements, while the most common kind of sporadic activation is the activation from a preceding SWC (see Figure 10, marking 1).

An RCM `Clock` element can be derived from an AMALTHEA `PeriodicStimulus` element while a sporadic activation can be translated from an AMALTHEA `SporadicActivation` element and from the order of `Runnable` elements within an `Activity Graph` element.

RCM only express software to hardware allocation information and it does that using the `isAllocated` association, which can be specified between any RCM software and hardware element. Such information can be derived from the AMALTHEA `SchedulerALlocation` and `Task Allocation` elements. The former allows for the linking of `Scheduler` elements to processing unit elements while the latter links `Task` elements to Scheduler elements.

RCM provides for the specification of timing requirements on response-time analysis and end-to-end data propagation delay analysis [38, 42]. These requirements are expressed using `Data Age` and `Data Reaction` elements, which can be translated from AMALTHEA `EventChainLatencyConstraint` elements. If the attribute `LatencyType` of an `EventChainLatencyConstraint` is set to `Age` then the AMALTHEA constraint will be translated into an RCM `Data Age`; if the attribute `LatencyType` of an

`EventChainLatencyConstraint` is set to `Reaction` then the AMALTHEA constraint will be translated into an RCM `Data Reaction`. Figure 11 provides an example of a latency data age constraint. The resulting representation in the RCM model is highlighted in Figure 10, marking 4.

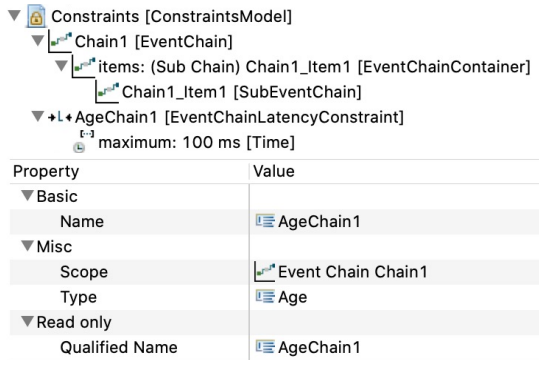

Figure 11: Extract of an AMALTHEA architecture depicting a latency data age constraint.

4.2. Back-annotating timing analysis results

The above relationships allow for the translation of an AMALTHEA into an RCM architecture. Response-time analysis and end-to-end data-propagation delay analysis can be executed on the generated RCM architecture. Once the timing analyses are run, their results should be annotated back to the starting AMALTHEA model. To perform such back-annotation we propose to use the AMALTHEA `MeasurementModel`. We use the `EventChainMeasurement` element and the `TaskMeasurement` element for storing the results of the timing analyses for data chains and tasks, respectively. To this end, we propose to create three `custom properties` named `Worst-Case Reaction Time`, `Worst-Case Response Time and Creator`. The first two properties identify the type of the recorded results while the last property its origin. It should be noted that these results are statically determined meaning that they give us values for the upper bounds and not for the distribution of the measured constraints. Besides, since the results are specific for a model, changes in the AMALTHEA model should lead to an invalidation of the measurement model. An example is shown in Figure 12 where the results of the timing analysis for an event chain `EC_Sequence_FL` of type reaction constraint is highlighted.

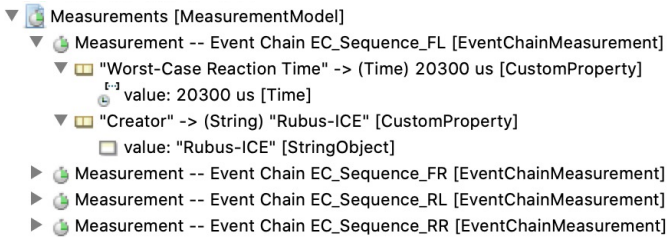

Figure 12: Extract of an AMALTHEA architecture depicting the back annotation of latency constraints.

5. Automating the translation

This section provides details on the implementation of the proposed automation mechanism. First, we provide a high-level description of the mechanism and after we discuss each phase of the automation mechanism in greater detail.

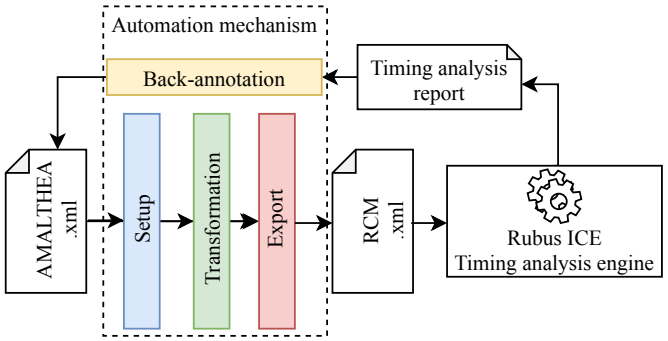

Figure 13: Overview of the automation mechanism and its phases.

5.1. Overview of the Transformation Process

Figure 13 shows the proposed automation mechanism and its composing phases, which are: *setup*, *transformation*, *export* and *back-annotation*. The setup, transformation and export phases represent the heart of the automation mechanism and provide for the automatic translation of an AMALTHEA model into an RCM one, which can then be used, amongst other things, as input for high precision timing analysis. The timing analysis is entrusted to the Rubus ICE development environment [43], which collects its results in a report that can be used to back-annotate the original AMALTHEA model. APP4MC is the Eclipse-based platform built around the AMALTHEA language [11]. Among other utilities, APP4MC provides means for manipulating AMALTHEA models using Java. The proposed automation mechanism uses such APP4MC utilities, therefore, is based on Java, too.

5.1.1. Setup

The setup phase is responsible for preparing the necessary infrastructure for the model transformation phase. This includes parsing the arguments of the transformation program, loading the respective AMALTHEA model from the XML file and creating the folder structure for the RCM model (if it does not yet exist). The arguments of the program define:

- The path to the AMALTHEA model (or to the timing analysis report).
- The destination folder of the RCM model.
- A flag that indicates the direction (i.e. transformation or back-annotation).

This is done using the `AMALTHEALoader` class provided by APP4MC. The class returns an `AMALTHEA` object that represents the loaded AMALTHEA model, which can be used to operate on the model.

5.1.2. Transformation

The transformation phase is the core of the automation mechanism. Hereafter, we outline the steps of the program and provide some details using pseudo-code. The transformation phase operates on the `AMALTHEA` object that was loaded from the provided AMALTHEA model file in the setup phase. The first step of the transformation is the extraction of the frequency domain defined in the AMALTHEA model. This is required as the execution time of the runnables are specified in ticks whereas the execution time of RCM SWCs is specified in time units. Algorithm 1 presents the pseudo-code of the part of the transformation responsible for translating the AMALTHEA hardware model. The transformation traverses all `HwStructure` elements of the `HwModel` (line 1). If a `HwStructure` element is of type `MICROCONTROLLER`, the element is transformed to an RCM `Node` element and added to the RCM model (line 2-4). In turn, all `HwModule` elements of the `HwStructure` are traversed and if they are of type `ProcessingUnit` they are transformed to an RCM `Core` element (line 5-7). The RCM `Core` is then added to the previously created `Node` element (line 8).

Algorithm 1: Hardware Model Transformation

```
1  for ∀ HwStructure ∈ amalthea.HwModel do
2      if HwStructure.type == MICROCONTROLLER then
3          rcmNode = toRcmNode(HwStructure)
4          rcm.add(rcmNode)
5          for ∀ HwModule ∈ HwStructure do
6              if HwModule is ProcessingUnit then
7                  rcmCore = toRcmCore(HwModule)
8                  rcmNode.add(rcmCore)
9              end
10         end
11     end
12 end
```

Once the hardware model is successfully transformed, the software model is transformed. Algorithm 2 shows the pseudocode for this part of the transformation. As only the parts of the software that are mapped to the platform need to be transformed, the `MappingModel` of AMALTHEA is utilised. The algorithm traverses all `TaskAllocation` elements of the `MappingModel` (line 1). For each `TaskAllocation` element, an empty list of RCM `SoftwareCircuit` elements is created. For each of the `RunnableCall` elements that are defined in the AMALTHEA task's `ActivityGraph` one RCM `Software Circuit` is created and added to the temporary list (line 3-6). The created `Software Circuit` has trigger input and output ports, but also data ports for each of the communication labels that are accessed by the original AMALTHEA `runnable`. If the `LabelAccess` element in the AMALTHEA model is defined as `read`, a data input port is created. If the label access is defined as `write` the data port is an output. Once all required `Software Circuits` are created, a trigger source is created for the RCM model that is equivalent to the one of the AMALTHEA task. This `Clock` element is then connected to the trigger port of the first `Software Circuit` in the temporary list (line 9-10). For all other `Software Circuit` pairs in the list the trigger output port is connected to the trigger input port of the respective next `Software Circuit` in the list (line 12). Finally, a `TrigTerminator` element is added to the trigger output port of the last `SoftwareCircuit` in the list. This effectively creates a trigger chain of all `Software Circuit` elements in the list. Afterwards, the `Software Circuit` elements in the temporary list are added to the RCM model (line 15). After this step, the main parts of the AMALTHEA application are represented in the RCM model. However, it remains to connect the data ports of the RCM model to realise the communication between `Software Circuit` element. Data input and output ports are connected if they are a result of access to the same `Label` in the AMALTHEA model. Generally, only one writer exists, but multiple readers are possible.

Algorithm 2: Task Transformation

```
1  for ∀ TaskAllocation ∈ amalthea.MappingModel do
2      tmpSwcList = ∅
3      for ∀ ActivityGraphItem ∈ TaskAllocation.Task do
4          if ActivityGraphItem is RunnableCall then
5              rcmSWC = getRcmSwc()
6              tmpSwcList.add(rcmSWC)
7          end
8      end
9      rcmClock = toRcmTrigClockTT(TaskAllocation.Task)
10     tmpSwcList.first = conTrigIn(rcmClock)
11     for SWC pair ∈ tmpSwcList do
12         linkTrigger()
13     end
14     tmpSwcList.last = conTrigOut(rcmTrigTerminator)
15     rcm.addAll(tmpSwcList)
16 end
```

As the last step, the timing constraints that are specified in the AMALTHEA model need to be represented in the RCM model as well. This information can be obtained from the `ConstraintsModel` of AMALTHEA. While AMALTHEA specified the complete sequence of events that are part of the `EventChain`, it is only necessary to get its beginning and end. At these points RCM requires `AgeDataStart` and `AgeDataEnd` or `ReactionDataStart` and `ReactionDataEnd` elements for data age constraints or reaction constraints respectively. To connect these to the RCM `Software Circuit`, a dedicated data port is added to which the timing constraint is connected.

5.1.3. Export

The export phase is responsible for writing the RCM model to its XML output format that can then be loaded by the Rubus ICE timing analysis engine. Rubus ICE allows for different alternatives to store the RCM model. For example, SWCs can be defined in one file and then referenced from the main file. The same can be done with partition or node elements. In a manual workflow this might be desired as it would save some development effort

and it provides for a more structured representation of the model. In this case, the RCM model is not intended to be modified manually as modifications should be performed on the AMALTHEA model instead. Therefore, the automation mechanism generates a single file that represents the complete RCM model.

5.1.4. Back-annotation

Rubus ICE allows for different alternative file formats for storing the analysis report (TXT file, HTML file and XML file). The back-annotation phase is responsible for parsing the analysis report file and extracting the timing properties and calculated values. The calculated values can then be added to the original AMALTHEA model using the measurement data model as described in Section 4.2. Once all the back-annotations are performed, the model can be written back to the XML file using the provided `AMALTHEAWriter` class.

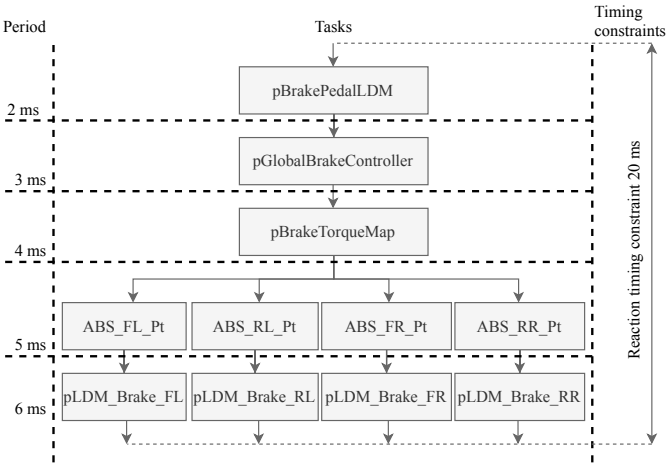

Figure 14: Block diagram of the BBW application. Different tasks and their communication pattern are shown. Task period and reaction time constraints are annotated.

6. Applying the automation mechanism on an industrial use case

In this section, we describe the application of the automation mechanism on the BBW use case. First, we show how the proposed mechanism can automatically translate the BBW AMALTHEA architecture into an RCM one, where response-time analysis and end-to-end data-propagation delay analysis can be run. After, we use the proposed mechanism for back-annotating the timing analysis results to the starting AMALTHEA model. A BBW system replaces the old-fashioned mechanical linkages with a stand-alone braking system, which allows controlling the brakes through electronic means. There are different implementations of the BBW system. In this work, we use the implementation used by an international Swedish automotive OEM. Figure 14 provides for a graphical representation of the BBW use case, which is comprised of 11 periodic tasks, activated at 5 different periods. These tasks ensure that the BBW activates the brakes each time that the brake pedal is pressed. Besides, it ensures that the actuation happens within a specified time interval. To this end, four data chains subject to reaction timing constrains are specified on the architecture. The data chains share the first 3 tasks (`pBrakePedalLDM`, `pGlobalBrakeController` and `pBrakeTorqueMap`) and then continue individually on each wheel (`ABS_FL_Pt` and `pLDM_Brake_FL` for the front left wheel, `ABS_RL_Pt` and `pLDM_Brake_RL` for the rear-left wheel, `ABS_FR_Pt` and `pLDM_Brake_FR` for the front right wheel, and `ABS_RR_Pt` and `pLDM_Brake_RR` for the rear-right wheel). Figure 15 shows the AMALTHEA architecture of the BBW system realised in the Eclipse APP4MC platform, which is the AMLTHEA accompanying development environment.

It is important to note that the automation mechanism uses the XML file rather than the three-based graphical representation of it. Within APP4MC, each model is stored as an XML file so no further processing is needed. In this work, for the sake of clarity and verbosity, we omit the XML representation of the BBW use case and only provide its graphical representation as in Figure 15. Besides, in Figure 15 some model elements have not been expanded and some element names have been cropped out. The interested reader can find the complete AMALTHEA project for the BBW system in [44]. The AMALTHEA software model for the BBW system consists of 11 runnables mapping to the 11 tasks described above. Besides, it contains 10 labels realising the communication between the software elements. The AMALTHEA hardware model for the BBW system contains an `HwStructure` describing the `NXP_MPC5744P` processor [41], which is a single-core processor tailored for automotive applications. The frequency of the processor is specified through the `FrequencyDomain` element and is set to 3.0E8 Hz. The AMALTHEA operating system model and stimuli model for the BBW system specify the scheduler for the available core and the periodic activation of the eleven task elements, respectively. The AMALTHEA event data model of the BBW defines events that are connected to software elements and can be used to specify timing constraints. The AMALTHEA constraints model includes several pieces of information: `ProcessRequirement` elements specify the response-time of tasks, while the `EventChainLatencyConstraint` elements describe the reaction constraints described above. Eventually, the mapping model specifies the allocation of the 11 tasks to the task scheduler and the allocation of the task scheduler to the available core.

As described in the previous section, the proposed automation mechanism is realised as a Java program. This means that it can be run in any instance of the Eclipse platform as the program takes care of importing the relevant packages and solving the dependencies with the APP4MC platform. To execute the mechanism, we have provided the path to the AMALTHEA BBW model as input together

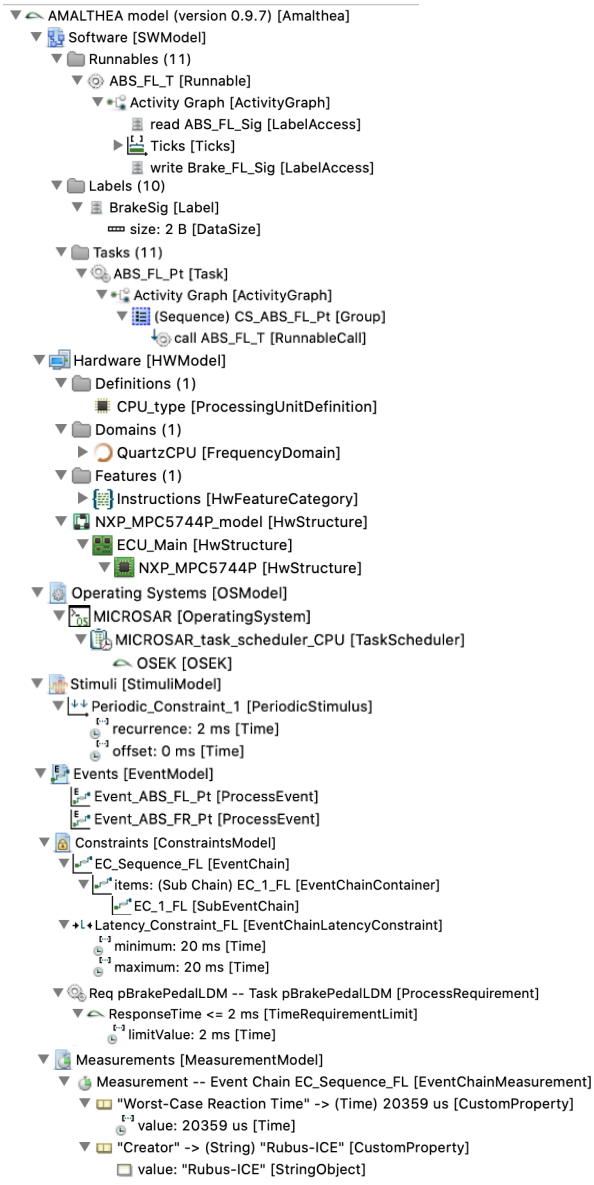

Figure 15: Excerpt of the AMALTHEA architecture of the BBW system realised within the Eclipse APP4MC platform.

with the direction of the mechanism and the path to the folder for storing the generated RCM file. The application of the proposed automation mechanism on the BBW model of Figure 15 produces an XML file describing the corresponding RCM architecture. As for the AMALTHEA architecture, we omit the XML file and show its graphical representation in Figure 16[3]. It should be noted that despite Rubus ICE provides for a graphical editor, all the models are stored as XML files. Besides, all the Rubus ICE plugins, as the timing analysis plugins, work directly on the XML files. The automation mechanism has generated the RCM file in 5.71 ms. We have measured the run-time for the setup, transformation and export phases, which are: 3.20 ms, 0.86 ms and 1.65 ms, respectively. The measured run-time refers to a system containing an Intel Core i9 8-Core processor at 2.3 GHz and 32 GB RAM. It can be noted that the setup phase has required the most time while the actual transformation phase the least time (later in Section 7.2 the scalability of the proposed mechanism is investigated using a large scale industrial application). Figure 16 shows the SWCs generated from the tasks and respective runnables along with their data and control flows. To identify the origin of a SWC, its name is a result of the AMALTHEA task name and the runnable name. In Figure 16 shortened names have been presented to increase readability. Each `SWC` is equipped with data ports (grey circles in Figure 16) and control ports (white triangles in Figure 16) translated from `EventChain` and `Activity Graph` elements. Periodic activation of tasks is translated using clock elements. Execution times, worst-case execution times and other timing properties are specified as properties of the SWCs. The AMALTHEA reaction constraints are translated as `DataReaction` elements represented as grey boxes in Figure 16. To evaluate whether the reaction constraints of 20 ms are met, we run delay analyses on the automatically generated RCM architecture [21]. The analyses are run in Rubus ICE, which displays the analysis results in so-called analysis reports. Figure 17 depicts an excerpt of the analysis report, while the complete report can be found at [44]. The report specifies the name of the constraint on the left-most column, the specified delay in the column in the middle and the calculated delay on the right-most column. In our case, the specified delays of 20 ms are not met as the calculated delays are of 20.359 ms, 20.267 ms, 20.177 ms and 20.037 ms, respectively. To back-annotate the analysis results, we have run the automation mechanism a second time with the path to the timing analysis report and the opposite direction as arguments. The calculated delays together with the tasks maximum response times are annotated back to the original AMALTHEA model. The proposed mechanism enriches the starting XML file with the tags representing the measurement model. The measured run-time for the back-annotation is 1.90 ms. Figure 18 shows the graphical representation of the updated AMALTHEA model containing the analysis results for the BBW system.

[3]The interested reader can find the complete RCM model for the BBW system at [44]

7. Evaluation and discussion

In this section, we describe and discuss the activities we have performed for validating the correctness, scalability and performance of the proposed automation mechanism. Eventually, we discuss the industrial relevance of this work.

7.1. Correctness

In Section 4, we have discussed how the wealth of elements of AMALTHEA could make the existence of other relationships between AMALTHEA and RCM elements possible. We hypothesise that the existence of alternative

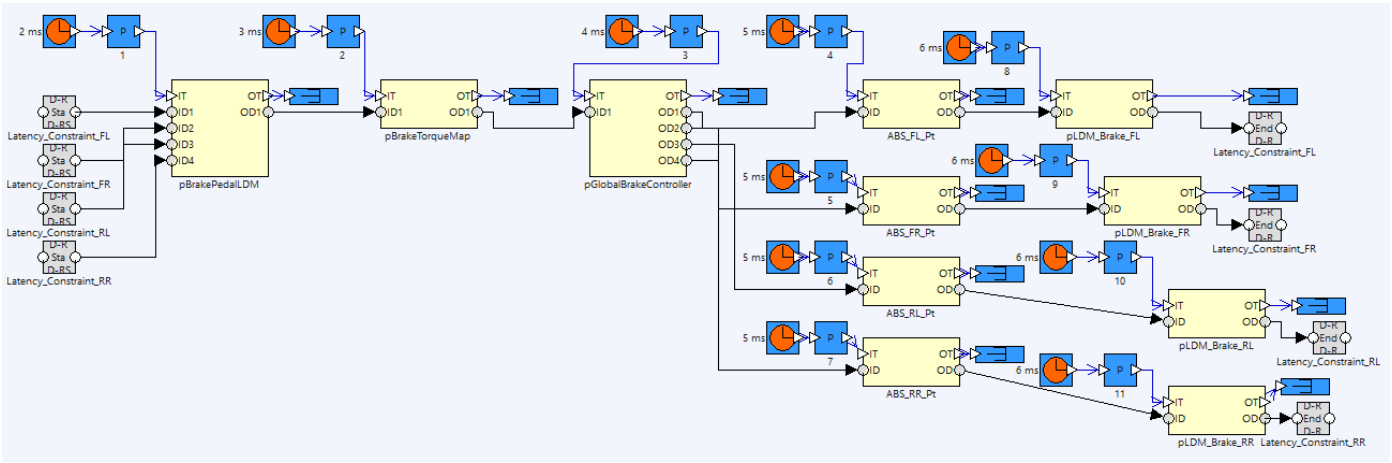

Figure 16: Excerpt of the RCM architecture of the BBW system realised within Rubus ICE.

Constraint	Maximum delay	Calculated delay
Latency_Constraint_FL	20 ms	20359 us
Latency_Constraint_FR	20 ms	20267 us
Latency_Constraint_RL	20 ms	20177 us
Latency_Constraint_RR	20 ms	20087 us

Figure 17: Excerpt of the timing analysis report.

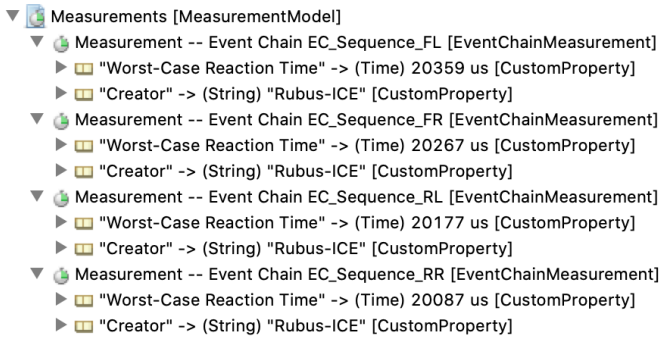

Figure 18: Excerpt of AMALTHEA architecture showing the back-propagation of the analysis results.

relationships does not affect the correctness of the identified ones. To prove the correctness of the mapping scheme we have created and used for the automation mechanism, we have compared the automatically generated RCM model for the BBW system with an RCM model created manually by a software engineer. The engineer has more than 10 years of experience with RCM and automotive software development. Besides, he has been involved in several national and European research projects. We have provided the engineer with the starting AMALTHEA model for the BBW system and the link to the AMALTHEA official documentation. All the artefacts and documentation were provided via email and we have remained available for any clarification.

Figure 19 shows the RCM architecture for the BBW system as developed by the software engineer. It might be noted that this architecture is identical to the one automatically generated and reported in Figure 16 except for two differences regarding the data ports of the SWCs. The first difference is that in our architecture the automation mechanism has created one data port for each latency constraint, while the software engineer has created one single data port for all the constraints. In the case of the BBW system, such a difference does not have any impact as there is no data communicated among tasks (SWCs) since the first task (SWC) reads the break pedal signal. In the general case where data might be passed among tasks (SWCs), there should be separate ports. The second difference is that the engineer used a special data port called `named data port` for improving the graphical representation of the model. The named data port element called `data` links the data output port `DO` of the `pBlobalBrakeController` SWC to all the DI ports of the `ABS_RR_Pt`, `ABS_RL_Pt`, `ABS_FR_Pt` and `ABS_FL_Pt` SWCs without changing the semantic of the data port. Automation mechanisms might be notoriously less effective in capturing specific syntactic and semantic aspects, which can be easily handled by engineers. However, in this case these aspects were only impacting the graphical display of the model. The comparison has shown that the mapping scheme leveraged by the automation mechanism translates AMALTHEA architectures in RCM ones in a similar way to a skilled software engineer with several year of experience with these technologies would do. Hence, our hypothesis seems to be correct: other mapping relationships among AMALTHEA and RCM languages do not affect the correctness of the proposed mapping scheme. This conclusion is strengthened by the results of the execution of the proposed automation mechanism on the EMS and FBEM use cases described in the following section.

7.2. Scalability and performance using Case Studies

By running the automation mechanism on the BBW use case, we have shown that the mechanism applies to

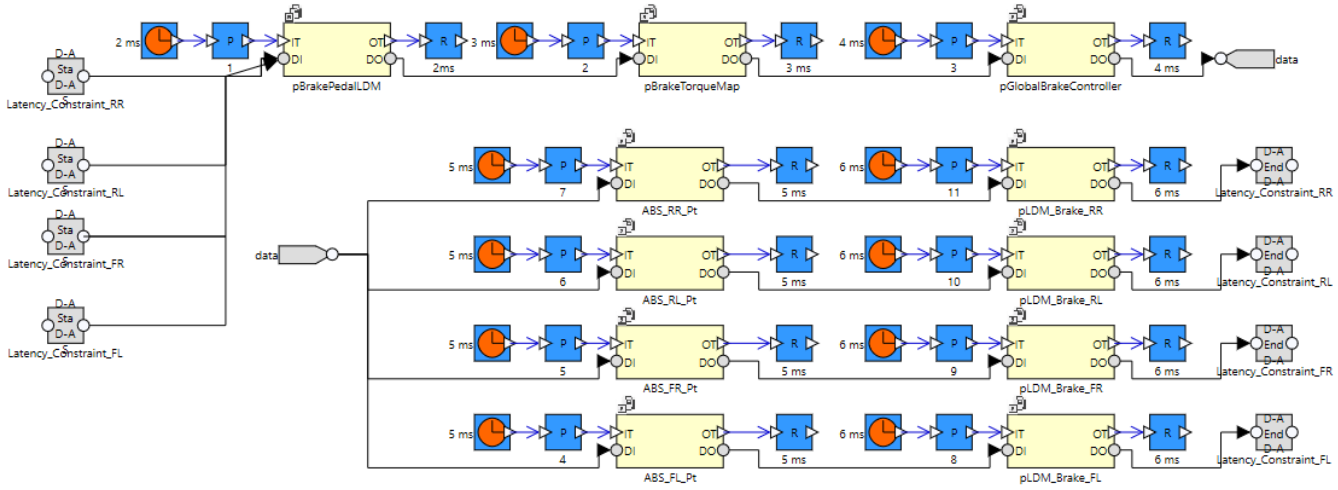

Figure 19: Excerpt of the RCM architecture of the BBW system created manually from a software engineer.

real-world automotive systems and that its performance is acceptable for its intended use. However, the BBW system is a rather concise system, which composes of few dozen of model elements. To demonstrate that the proposed mechanism does not only apply on bigger systems, but that its performance remains acceptable, we apply the automation mechanism on two additional use cases notably the FBEM and the EMS.

7.2.1. Description of the FBEM use case

The FBEM use case was proposed by Robert Bosch GmbH within the 2017 WATERS Industrial Challenge [27]. The challenge describes an engine management system consisting of 1194 runnables with periodic activation and 46 runnables with sporadic activation. Communication among runnables is entrusted to 10000 label elements. The runnables are mapped to 21 tasks and interrupt service routines, that are executed by the operating system. The hardware platform consists of a multi-core processor with four cores that execute at 200 MHz, each with a local scratchpad program and data memory. Global DRAM is used for communication. Latency and other timing elements, such as data propagation constraints on chains of runnables, are expressed as well. Both the hardware and software models are provided in the Amalthea file format. The interested reader can download the full model at the link provided in the Note 2. We have not modified the model except for some minor fixes as follows:

- Added the unit "Hz" to the GenericPLL Frequency Domain element used to denote the CPU frequency.
- Removed the feature accessElement from the model as it is no longer supported after the migration to Amalthea 0.9.7.

7.2.2. Description of the EMS use case

The EMS use case is described by Frey in [26] and is part of the examples provided with AMALTHEA [45]. It executes 43 runnables using 3 different tasks. Communication between the different runnables is realised by 71 communication labels. The hardware platform consist of a single core processor that is clocked with 200 MHz. The interested reader can download the full model at the link provided in the Note 2. We have not modified the model except for some minor fixes as follows:

- Created a mapping model that allocates the three tasks to the operating system, as well as the operating system to the processor.
- Increased the clock frequency to 600 MHz as the initial DemoCar model has a Utilization > 1 which makes it unschedulable using the provided worst-case execution time values.

7.2.3. Performance and Scalability Evaluation

To evaluate the performance and scalability of the transformation process the same measurements are performed as for the BBW use case in Sec. 6. Measurement results are reported for the three transformation phases (1) Setup Phase, (2) Transformation Phase, (3) Export Phase, as well as for the total time taken. 1000 runs of the automation mechanism are recorded. Table 2 presents the collected results. It is noteworthy to point out the difference between the minimum and maximum times measured for the setup phase is rather large for both case studies. This is due to the initial latency of the AMALTHEA model loader that occurs only for the first measurement. Compared to the BBW use case, for the large scale case FBEM case study, the transformation process now requires the most time of all phases, but despite the significant size of the application model the required time is still in the millisecond range. Hence, we conclude that the proposed automation mechanism applies to large scale industrial sized applications with more than 1000 runnables and 10000 communication labels, too, with a maximum total time of 2628 ms. We did

	EMS				FBEM			
	Min	**Max**	**Avrg.**	**SD**	**Min**	**Max**	**Avrg.**	**SD**
Setup Phase	0.74 ms	736 ms	2.27 ms	23 ms	98 ms	1024 ms	113 ms	31 ms
Transformation Phase	0.13 ms	22 ms	0.31 ms	1 ms	530 ms	1561 ms	657 ms	70 ms
Export Phase	0.84 ms	45 ms	1.43 ms	2 ms	173 ms	616 ms	208 ms	31 ms
Total Time	1.78 ms	804 ms	4.02 ms	23 ms	833 ms	2628 ms	979 ms	96 ms

Table 2: Duration of the proposed transformation of 1000 executions with the EMS and the FBEM use cases.

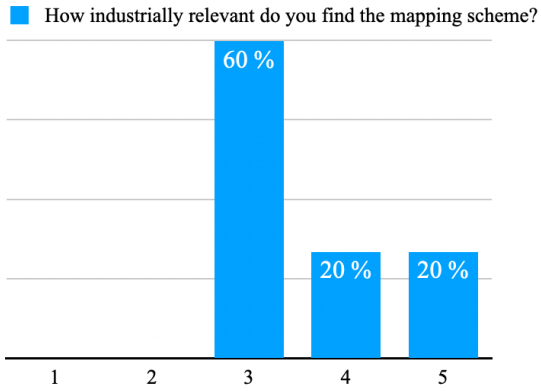

Figure 20: Perceived industrial relevance of this research.

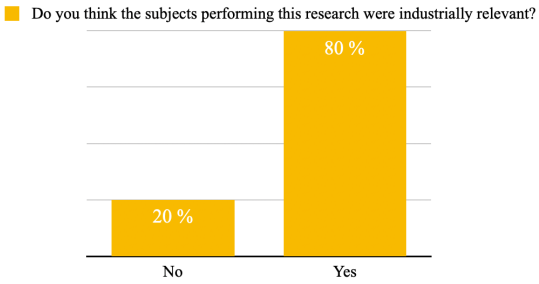

Figure 21: Perceived industrial relevance of the subjects performing this research.

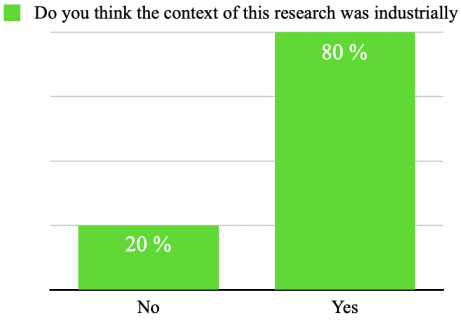

Figure 22: Perceived industrial relevance of the context of this research this research.

not measure the performance of the proposed mechanism when back-annotating the results. The performance of the back-propagation mainly depends on the number of timing constraints specified. In our experience, we have noticed that almost all the models do not contain more than few constraints even for industrial applications. Hence, scalability and performance analysis would not make sense.

7.3. Industrial relevance

To assess the industrial relevance of this work, we have used an online survey. We have asked 10 experts to answer a set of 10 questions drawing on the model for evaluating research rigour and relevance presented by Ivarsson et al. [39]. All the respondents possess a PhD degree and have prior and established experience in the field of automotive software engineering. The interested reader can access the complete list of questions at [44]. The pool of experts was composed of the following profiles:

- 6 researchers in the fields of cyber-physical systems, software engineering, software architecture, model-driven engineering and real-time embedded systems.
- 4 practitioners working for Swedish and German automotive OEM, tier-1 and tier-2 companies.

The respondents found the proposed research to be industrially relevant (Figure 20), with 40% of the experts that found it relevant or extremely relevant (Figure 20). One expert has remarked that "*In my opinion, such mappings have high-industrial relevance, especially in supporting interoperability among various industrial languages and tools*". According to Ivarsson et al., the subject of a research refers to the profiles that performed the research and their suitability to industrial settings [39]. Figure 21 shows that 80% of the experts have found the subjects to be industrially relevant. The context of a research refers to the environment in where the research was carried out, while the research method refers to the protocol used for planning and executing the research [39]. Similar to the subject, 80% of the experts have found the context and the research method of this work to be industrially relevant (Figure 22 and Figure 23). The scale of a research refers to the significance of the use case used for its validation and its relevance for industry. The opinion of the experts on this question was split and only 50% of the experts has found the size of the BBW use case to be industrially relevant. Half of the experts has found the size of the BBW use case not particularly representative of industrial settings. To address this concern, we have complemented the BBW use case with the FBEM use case. We show the application of the automation on the FBEM in Section 5.

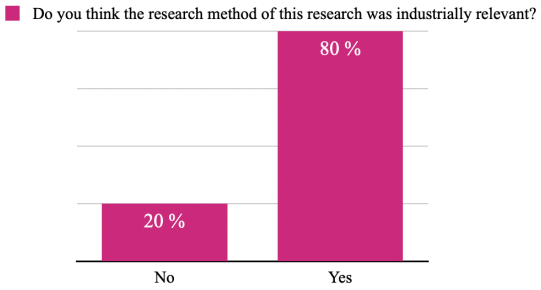

Figure 23: Perceived industrial relevance of the research method of this research.

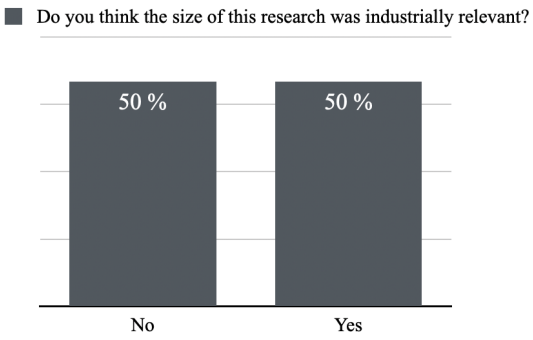

Figure 24: Expert survey results.

8. Related Work

To the best of our knowledge, this research represents the first documented effort in providing an automation mechanism for translating an AMALTHEA architecture into an RCM compliant one. However, mapping schemes between different ALs and their automation by model transformations are not new research lines and there is a vast body of literature focusing on these two areas.

One of the first work focusing on the interplay of Model-Driven Engineering (MDE) techniques and ALs is the study by Lago et al., which propose MDE as a technological solution for building next-generation ALs [46]. In their study, the authors claim that MDE might bring several benefits to ALs including support for defining ALs, managing language and tool extensibility, automatically generating different kinds of artefacts spanning the development life cycle. One example of such an interplay is reported in the work by Malavolta et al. [47], which uses model transformations for providing ALs and tools interoperability. This is achieved using an automated framework called DUALLY. The authors apply the DUALLY approach on an industrial system for transforming a UML architecture into a Darwin/FSP one.

In the automotive domain, there are several works investigating the integration between automotive-specific ALs such as EAST-ADL [48], AUTOSAR [49], RCM [48], etc. EAST-ADL in an automotive-specific AL, which through the years became a *de facto* standard. In their work, Cuenot et al. present a mapping scheme between EAST-ADL and AUTOSAR [50] . The proposed mapping scheme focuses on the relation between the structural elements and events of these two ALs, only. The authors present a validation of their mapping scheme using an industrial use case. Similar to the research by Cuenot et al., the work by Qureshi et al. also proposes a mapping scheme between EAST-ADL and AUTOSAR [51]. However, this work focuses on relating the behavioural elements of the two languages. Other researches focusing on integrating EAST-ADL and other ALs are the work from Enoiu et al. [52] and Kang et al. [53]. In the first work, the authors present a way of integrating architectural models and verification techniques focusing on the integration of EAST-ADL models and timed automata models. In the second work, Kang et al. extend EAST-ADL with energy constraints and integrates the extension with formal analysis techniques based on ERT constraints. To facilitate the integration, the authors propose a mapping scheme, which is validated and demonstrated on an automotive case study.

Other examples of works focusing on the integration of automotive ALs are the researches by Giese et al. [54] and Bucaioni et al. [55]. The former uses model transformation for integrating SYSML and AUTOSAR. In particular, the authors use graph grammars for defining an automation mechanism to tackle consistency problems typical of development scenarios using different ALs. The automation mechanism is exemplified by an experiment done within an industrial project. The research from Bucaioni et al. first provides a metamodel definition for RCM. Later, they provide a model transformation integrating RCM and AUTOSAR for demonstrating how a proper metamodel definition can enable a full-fledged model-driven development.

Sometimes, relations among ALs are used to classify and compare the languages rather than integrating them. This is the case of the works by Sailer et al. [56] and Medvidovic et al. [57]. The former presents a practical comparison of ALs focusing on the design and development of automotive systems on multi-core platforms. The selected languages are AMALTHEA, AUTOSAR and ASAM MDX. The comparison is carried out on a subset of common elements. The latter, first propose a classification framework. Then it uses the proposed framework for classifying and comparing several existing ALs. In the process, the authors identify key properties of ALs, areas where existing ALs provide extensive support and areas where ALs lack of support.

9. Conclusion and Future Work

In this work, we have reported on our experience in providing an automation mechanism for the integration of two automotive architectural languages being AMALTHEA and the Rubus Component Model.

The proposed mechanism allows for the automatic translation of an AMALTHEA architecture into a Rubus Component Model one where high-precision timing analysis can be run. Besides, it allows for the back annotation of the analysis results into the starting AMALTHEA architecture. We have evaluated the applicability of the proposed

mechanism using industrial automotive use cases including the brake-by-wire system and a large scale engine management system consisting of more than 1000 runnables and 10000 communication labels. To evaluate the scalability of the approach we compare the runtime of the translation for both models, demonstrating its practicability in an industrial work-flow. We have discussed the correctness of the proposed mechanism by comparing the automatically generated AMALTHEA and RCM architectures to AMALTHEA and RCM architectures crafted manually as the result of a traditional process. Eventually, we have asked automotive experts to rate the industrial relevance of this research using online surveys. The respondents have found the automation mechanism interesting and industrially relevant.

While working on this research, we have realised that AMALTHEA has an extensive set of data models and elements. This wealth of elements makes the existence of alternative relationships between elements of AMALTHEA and the Rubus Component Model possible. However, we envision that not all the relationships will be meaningful for all the stakeholders involved in the development process. Hence, possible lines for future work encompass investigating relations between AMALTHEA data models and different roles on the development process and extending the automation mechanism with further mapping rules. Another possible extension to the automation mechanism involves the addition of optimisations such as the use of `named data ports`. Another line of future work encompasses the integration of the timing analyses with the automation mechanism so as to reach a full-fledged tool-chain integration. In this context, advanced model transformation techniques can be used for automatically preserving the consistency among the artefacts [58]. Eventually, another interesting future direction is to evaluate the relations between AMALTHEA and RCM and further languages belonging to other (optional) views of architecture frameworks as the so-called deployment viewpoint [59].

Acknowledgment

The work in this paper is supported by the Swedish Knowledge Foundation (KKS), through the project A-CPS, and by the Swedish Governmental Agency for Innovation Systems (VINNOVA), through the project PANORAMA. The authors would like to thank the industrial partners, especially Robert Bosch, Arcticus Systems and Volvo, for their valuable inputs.

References

[1] R. N. Charette, This car runs on code, IEEE Spectrum 46 (3) ((2009)) 3.

[2] S. Behere, M. Törngren, A functional reference architecture for autonomous driving, Information and Software Technology 73 (2016) 136–150.

[3] S. Saidi, S. Steinhorst, A. Hamann, D. Ziegenbein, M. Wolf, Future automotive systems design: research challenges and opportunities: special session, in: Proceedings of the IEEE International Conference on Hardware/Software Codesign and System Synthesis, 2018, p. 2.

[4] A. Finkelstein, J. Kramer, B. Nuseibeh, L. Finkelstein, M. Goedicke, Viewpoints: A framework for integrating multiple perspectives in system development, International Journal of Software Engineering and Knowledge Engineering 2 (01) (1992) 31–57.

[5] R. Hilliard, Iso/iec/ieee 42010.

[6] D. Garlan, R. T. Monroe, D. Wile, Acme: Architectural description of component-based systems, Foundations of component-based systems 68 (2000) 47–68.

[7] I. Malavolta, P. Lago, H. Muccini, P. Pelliccione, A. Tang, What industry needs from architectural languages: A survey, IEEE Transactions on Software Engineering 39 (6) (2013) 869–891.

[8] M. Broy, M. Gleirscher, S. Merenda, D. Wild, P. Kluge, W. Krenzer, Toward a holistic and standardized automotive architecture description, Computer 42 (12) (2009) 98–101.

[9] A. Bucaioni, P. Pelliccione, Technical architectures for automotive systems, in: 2020 IEEE International Conference on Software Architecture (ICSA), IEEE, 2020, pp. 46–57.

[10] Y. Dajsuren, M. van den Brand, A. Serebrenik, R. Huisman, Automotive adls: a study on enforcing consistency through multiple architectural levels, in: Proceedings of the 8th international ACM SIGSOFT conference on Quality of Software Architectures, 2012, pp. 71–80.

[11] Introduction to APP4MC, https://www.eclipse.org/app4mc/help/app4mc-1.0.0/index.html#section1.

[12] K. Hänninen, J. Mäki-Turja, M. Sjödin, M. Lindberg, J. Lundbäck, K.-L. Lundbäck, The Rubus Component Model for Resource Constrained Real-Time Systems, in: 3rd IEEE International Symposium on Industrial Embedded Systems, (2011).

[13] ITEA 3 AMALTHEA project, https://itea3.org/project/amalthea.html.

[14] ITEA 3 AMALTHEA4public project, https://itea3.org/project/amalthea4public.html.

[15] PANORAMA - Boosting Design Efficiency for Heterogeneous Systems, https://itea3.org/project/panorama.html.

[16] MultEx - Software development using multiple execution models, https://www.es.mdh.se/projects/61-MultEx.

[17] H. A. Hansson, H. W. Lawson, M. Strömberg, S. Larsson, Basement: A distributed real-time architecture for vehicle applications, in: The Engineering of Complex Real-Time Computer Control Systems, 1997, pp. 223–244.

[18] A. Bucaioni, A-cps: Automation in high-performance cyber physical systems development., in: STAF (Co-Located Events), 2019, pp. 15–20.

[19] A. Bucaioni, Boosting the development of high-performance automotive systems, in: Junior Researcher Community Event at Software Technologies: Applications and Foundations 2019 STAF-JRC19, 28 Aug 2019, Eindhoven, Greece, 2019.

[20] L. Sha, T. Abdelzaher, K.-E. Årzén, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, A. K. Mok, Real time scheduling theory: A historical perspective, Real-time systems 28 (2-3) (2004) 101–155.

[21] S. Mubeen, J. Mäki-Turja, M. Sjödin, Support for end-to-end response-time and delay analysis in the industrial tool suite: Implementation issues, experiences and a case study, Computer Science and Information Systems 10 (1) (2013) 453–482.

[22] ISO 26262-1:2011: Road Vehicles in Functional Safety, http://www.iso.org/.

[23] A. Bucaioni, M. Becker, J. Lundbäck, H. Mackamul, From amalthea to rcm and back: a practical architectural mapping scheme, in: 2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 2020, pp. 537–544. doi:10.1109/SEAA51224.2020.00089.

[24] S. Sendall, W. Kozaczynski, Model transformation: The heart and soul of model-driven software development, Software, IEEE 20 (5) (2003) 42–45.

[25] T. Gorschek, P. Garre, S. Larsson, C. Wohlin, A model for technology transfer in practice, IEEE software 23 (6) (2006) 88–95.
[26] P. Frey, A timing model for real-time control-systems and its application on simulation and monitoring of autosar systems, Ph.D. thesis, Universität Ulm (2011).
[27] A. Hamann, D. Dasari, S. Kramer, M. Pressler, F. Wurst, D. Ziegenbein, Waters industrial challenge 2017 (2017 [Online]).
[28] P. Hank, S. Müller, O. Vermesan, J. Van Den Keybus, Automotive ethernet: in-vehicle networking and smart mobility, in: 2013 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013, pp. 1735–1739.
[29] S. Mubeen, J. Mäki-Turja, M. Sjödin, Communications-Oriented Development of Component- Based Vehicular Distributed Real-Time Embedded Systems 60 (2) ((2014)) 207–220.
[30] S. Mubeen, J. Mäki-Turja, M. Sjödin, Integrating mixed transmission and practical limitations with the worst-case response-time analysis for controller area network, Journal of Systems and Software 99 (2015) 66 – 84.
[31] S. Mubeen, H. W. Lawson, J. Lundbäck, M. Gälnander, K.-L. Lundbäck, Provisioning of predictable embedded software in the vehicle industry: The rubus approach, in: 2017 IEEE/ACM 4th International Workshop on Software Engineering Research and Industrial Practice (SER IP), (2017), pp. 3–9.
[32] S. Mubeen, M. Ashjaei, M. Sjödin, Holistic modeling of time sensitive networking in component-based vehicular embedded systems, in: 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 2019.
[33] Timing Augmented Description Language (TADL2) syntax, semantics, metamodel Ver. 2, Deliverable 11, Aug. 2012.
[34] F. N., R. K., N. J., J. 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).
[35] M. S., N. T., S. M., L. J., L. K., Supporting timing analysis of vehicular embedded systems through the refinement of timing constraints, Software & Systems Modeling (2017) 1–31.
[36] J. Mäki-Turja, M. Sjödin, Tighter response-times for tasks with offsets, in: Real-time and Embedded Computing Systems and Applications Conference (RTCSA), Springer-Verlag, 2004.
URL http://www.es.mdh.se/publications/622-
[37] S. Mubeen, J. Mäki-Turja, M. Sjödin, Implementation of holistic response-time analysis in rubus-ice: Preliminary findings, issues and experiences, in: The 32nd IEEE Real-Time Systems Symposium (RTSS), WIP Session, 2011, pp. 1–4.
URL http://www.es.mdh.se/publications/2209-
[38] S. Mubeen, J. Mäki-Turja, M. Sjödin, Support for end-to-end response-time and delay analysis in the industrial tool suite: Issues, experiences and a case study, in: Computer Science and Information Systems, vol. 10, no. 1, pp 453-482, January 2013.
[39] M. Ivarsson, T. Gorschek, A method for evaluating rigor and industrial relevance of technology evaluations, Empirical Software Engineering 16 (3) (2011) 365–395.
[40] P. Runeson, M. Höst, Guidelines for conducting and reporting case study research in software engineering, Empirical software engineering 14 (2) (2009) 131.
[41] NXP, MPC5744P Data Sheet 32-bit MCU suitable for ISO26262 ASILD chassis and safety applications, rev. 6.1 (2017).
URL https://www.nxp.com/docs/en/data-sheet/MPC5744P.pdf
[42] M. Becker, D. Dasari, S. Mubeen, M. Behnam, T. Nolte, End-to-end timing analysis of cause-effect chains in automotive embedded systems, Journal of Systems Architecture 80 (2017) 104 – 113.
[43] Rubus-ICE: Integrated component Development Environment, http://www.arcticus-systems.com (2017).
[44] A. Bucaioni, M. Becker, Enabling Automated Integration of Architectural Languages:an Experience Report from the Automotive Domain (Sep. 2021). doi:10.5281/zenodo.5529656.
URL https://doi.org/10.5281/zenodo.5529656
[45] P. Dziurzanski, A. K. Singh, L. S. Indrusiak, B. Saballus, Benchmarking, system design and case-studies for multi-core based embedded automotive systems, in: Proceedings of the 2nd International Workshop on Dynamic Resource Allocation and Management in Embedded, High Performance and Cloud Computing (DREAMCloud), 2016.
[46] P. Lago, I. Malavolta, H. Muccini, P. Pelliccione, A. Tang, The road ahead for architectural languages, IEEE Software 32 (1) (2014) 98–105.
[47] I. Malavolta, H. Muccini, P. Pelliccione, D. Tamburri, Providing architectural languages and tools interoperability through model transformation technologies, IEEE Transactions on Software Engineering 36 (1) (2009) 119–140.
[48] East-adl domain model specification, (2010), http://www.atesst.org/home/liblocal/docs/ATESST2_D4.1.1_EAST-ADL2-Specification_2010-06-02.pdf.
[49] Autosar techincal overview, (2016), http://autosar.org.
[50] P. Cuenot, P. Frey, R. Johansson, H. Lönn, M.-O. Reiser, D. Servat, R. T. Koligari, D. Chen, Developing automotive products using the east-adl2, an autosar compliant architecture description language, in: Embedded Real-Time Software Conference, Citeseer, 2008.
[51] T. N. Qureshi, D. Chen, H. Lönn, M. Törngren, From east-adl to autosar software architecture: a mapping scheme, in: European Conference on Software Architecture, Springer, 2011, pp. 328–335.
[52] E. P. Enoiu, R. Marinescu, C. Seceleanu, P. Pettersson, Vital: A verification tool for east-adl models using uppaal port, in: IEEE 17th International Conference on Engineering of Complex Computer Systems, 2012, pp. 328–337.
[53] E.-Y. Kang, P.-Y. Schobbens, Extending east-adl towards formal modeling and analysis of energy-aware real-time systems, in: 2013 10th IEEE International Conference on Control and Automation (ICCA), 2013, pp. 1890–1895.
[54] H. Giese, S. Hildebrandt, S. Neumann, Towards integrating sysml and autosar modeling via bidirectional model synchronization., in: MBEES, 2009, pp. 155–164.
[55] A. Bucaioni, A. Cicchetti, F. Ciccozzi, S. Mubeen, M. Sjödin, A Metamodel for the Rubus Component Model: Extensions for Timing and Model Transformation from EAST-ADL., Journal of IEEE Access 5 (1).
[56] A. Sailer, S. Schmidhuber, M. Hempe, M. Deubzer, J. Mottok, Distributed multi-core development in the automotive domain-a practical comparison of asam mdx vs. autosar vs. amalthea, in: ARCS 2016; 29th International Conference on Architecture of Computing Systems, VDE, 2016, pp. 1–8.
[57] N. Medvidovic, R. N. Taylor, A classification and comparison framework for software architecture description languages, IEEE Transactions on software engineering 26 (1) (2000) 70–93.
[58] R. Eramo, A. Bucaioni, Understanding bidirectional transformations with tggs and jtl, Electronic Communications of the EASST 57.
[59] P. Pelliccione, E. Knauss, R. Heldal, S. M. Ågren, P. Mallozzi, A. Alminger, D. Borgentun, Automotive architecture framework: The experience of volvo cars, Journal of systems architecture 77 (2017) 83–100.