Provisioning of Predictable Embedded Software in the Vehicle Industry: The Rubus Approach

Saad Mubeen*, Harold "Bud" Lawson†, John Lundbäck‡, Mattias Gålnander‡, Kurt-Lennart Lundbäck‡
* Mälardalen University, Västerås, Sweden
‡ Arcticus Systems AB, Järfälla, Sweden
† Lawson Konsult AB, Lidingö, Sweden
*saad.mubeen@mdh.se; †bud@lawson.se
‡{john.lundback, mattias.galnander, kurt.lundback}@arcticus-systems.com

***Abstract*—Providing computer-based services for vehicular systems has evolved to the point where majority of functions are realised by software. However, the need to provide safety in critical functions such as braking and engine control requires an approach that can guarantee reliable operation of the functions. At the same time, there are a variety of vehicle functions that are less critical. The main challenge for the vehicle manufacturers is to provide both types of functions in an economic and reliable manner. To meet this challenge, this paper considers the Rubus tool chain for model- and component-based development of vehicle software and a well-proven (in the industrial use for over twenty years) and certified (according to ISO 26262) real-time operating system for its execution. The paper provides an overview of the Rubus approach and driving concepts as well as the research results that are used in providing its tool chain. Moreover, the paper presents a success story of a unique academic-industrial collaboration in the vehicle domain that has resulted in sustained development of the tool chain. The collaborators form a clear value chain from academia, through tool developer, to the end users of the technology. The paper also highlights the perspectives of the collaborators and discusses the challenges faced, experiences gained and lessons learned from several technology transfer projects.**

I. Introduction

A large share of innovation and customer value in modern vehicles comes from advanced computer-controlled functionality. With the increasing volume of such functionality, the vehicle software has tremendously increased in size and complexity. For example, the amount of software in a car has increased from 100 lines of code in late 1970s to more than 100 million lines of code in a span of approximately three decades [1], [2]. The distributed nature of vehicle software over tens of nodes or Electronic Control Units (ECUs) further adds to its complexity. Moreover, the safety-critical nature of several vehicle functions puts real-time requirements on them. The developers of such functions are required to ensure that the functions are predictable, i.e., they behave in a timely manner when executed. The software complexity can be managed by using the tools that employ the principles of component-based software engineering (CBSE) [3] and model-driven engineering (MDD) [4]. The predictability of the functions can be verified by the tools that implement real-time schedulability analysis [5], [6], [7]. Such analysis can validate the timing requirements, without performing exhaustive testing, before the functions are deployed to the target platforms.

This paper discusses a case of an industrial tool chain, namely Rubus [8], that has proven to be very successful in applying the principles of CBSE, MDD and real-time scheduling theory in practice for over two decades. The Rubus tool chain supports model- and component-based development of vehicle software and its predictable execution on a real-time operating system (RTOS) that is certified in ISO 26262 safety standard according to ASIL D [9]. The paper describes the experiences of various collaborators in the academic-industrial collaboration shown in Fig. 1. The collaboration has been ongoing for 20 years. It has resulted in the development of the Rubus tool chain and its evolution based on the state-of-the-art research, industrial needs and feedback from the end users. One unique characteristic of the selected collaboration is that it offers a clear value chain from academia (mainly Mälardalen University); through tool developer (Arcticus Systems[1]); and finally, to the end users of the technology such as Volvo Construction Equipment (VCE)[2] and BAE Systems Hägglunds[3]. The paper also describes some important lessons learned during several technology transfer projects.

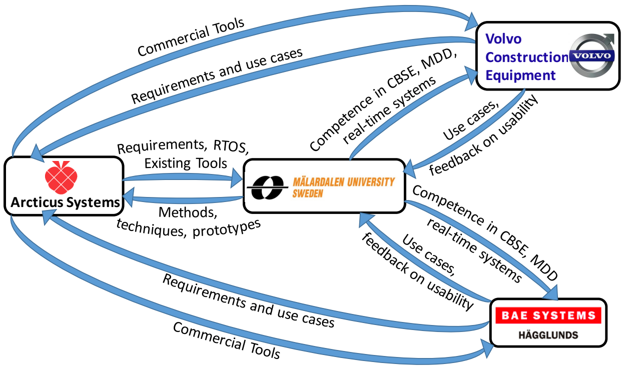
Volvo Construction Equipment
Arcticus Systems
MÄLARDALEN UNIVERSITY SWEDEN
BAE SYSTEMS HÄGGLUNDS
Commercial Tools
Requirements and use cases
Competence in CBSE, MDD, real-time systems
Use cases, feedback on usability
Requirements, RTOS, Existing Tools
Methods, techniques, prototypes
Competence in CBSE, MDD real-time systems
Use cases, feedback on usability
Requirements and use cases
Commercial Tools

Fig. 1. Example of flows of information within the collaboration.

II. Rubus: History, Evolution & Driving Concepts

A. The Rubus Concept

Arcticus Systems introduced the Rubus Kernel for industrial use in 1996. It was selected for the implementation of a Limited Slip Coupling Device for four-wheel drive vehicles by Haldex and later by Volkswagen. Since then Rubus has evolved into a significant product that is utilised by several international companies including VCE, BAE Systems Hägglunds, Elektroengine, BorgWarner, Hoerbiger and Knorr-Bremse. The Rubus concept is based upon the principles of CBSE and MDD. It is centered around the Rubus Component Model (RCM) [10] and its tool suite, namely Rubus-ICE (Integrated Component model development Environment) that consists of modelling tools, code generators, run-time infrastructure, analysis and simulation tools as shown in Fig. 3. An application

[1] http://www.arcticus-systems.com.
[2] https://www.volvoce.com.
[3] http://www.baesystems.com.

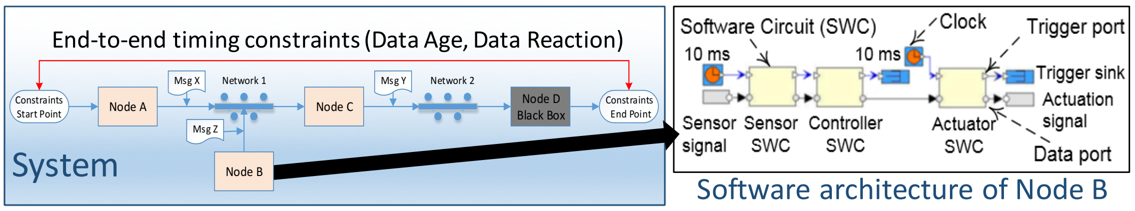

Fig. 2. System-level modelling and specification of timing constraints in a distributed embedded system using Rubus-ICE.

developed using Rubus-ICE can be executed on a variety of platforms, that is, various hardware platforms and RTOSs.

The main goal of Rubus is to be aggressively resource efficient and to provide means for developing predictable, timing analysable and synthesisable control functions in resource-constrained embedded systems. The highest-level hierarchical element in RCM is called the system which contains the models of nodes (ECUs) and networks. The lowest-level hierarchical element in RCM is called the Software Circuit (SWC). It encapsulates basic functions and has the run-to-completion semantics. The name "software circuit" is derived from the hardware analogy where component data and control flow behave as a chain of circuits. This promotes the analysis and verification of system timing constraints and resource utilisation that is accomplished by separating data and control flows in a network of SWCs. Various elements in RCM are depicted in the example software architecture shown in Fig. 2.

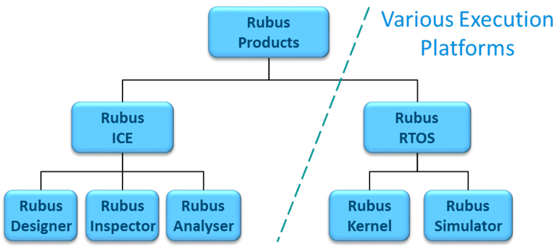

Fig. 3. Various tools in the Rubus tool chain.

B. Model Driven Development using Rubus

MDD has had an increasingly important role in designing and implementing software in vehicular embedded systems. Due to the complexity of these systems, the development must rely more and more upon automation and the interoperability amongst models such as Simulink. Various features provided by the Rubus tool chain are portrayed in Fig. 4. The three models provide various viewpoints reflecting all of the necessary information concerning the development, analysis, synthesis and execution of vehicular embedded systems.

1) Rubus Component Model - Viewpoint of the Development Team: The Rubus Designer tool is used to interactively describe the application that is developed using RCM. The developer designs the system in a platform-independent manner that focuses on the application. Timing and resource constraints are expressed in the model. The structure of the application is developed by means of SWCs. The structural model of a distributed embedded system with multiple networks and software architecture of a node in RCM are shown in Fig. 2.

2) Rubus Analysis Model - Viewpoint of the Analysis Framework: The resulting RCM design is formal, which lends itself to static analysis that is mapped to the actual run-time platform. In this viewpoint, the Rubus Analyser analyses the type checking, execution order and real-time requirements. This analysis helps in reducing late, costly and time-consuming testing efforts. Furthermore, mathematical models and supporting tools provide formal evidence of fulfilling the requirements [6]. The Rubus Analyser supports pre-runtime timing analysis of the system at various levels. For example, a single node is analysed by calculating the tasks' response times and comparing them with corresponding deadlines. The Rubus Analyser implements Response Time Analysis (RTA) of tasks with offsets [11], RTA of Controller Area Network (CAN) and its higher-level protocols [12], analysis of multiple networks, end-to-end timing analysis [6], [13] for calculating the age and reaction delays (corresponding constraints are specified in Fig. 2), as well as timing analysis of black box nodes (see Fig. 2) whose software architectures are not available [14].

The Rubus Inspector supports platform-independent formal and semi-formal model-in-the-loop testing environment. It supports unit testing as well as sub-system testing (node and network) and system testing (distributed system with multiple nodes and networks). Test inputs can be generated from LabView/Simulink and Matlab environments. Another tool, namely the Rubus Simulator provides for testing and verifying the composite of the Rubus Kernel and application software.

3) Rubus Run-time Model - Viewpoint of the Run-time Platform: The code for the actual run-time platform is synthesised from the software architecture. This automated synthesis prevents error-prone and costly integration errors. The Rubus Kernel or any other RTOS can be used as the runtime platform.

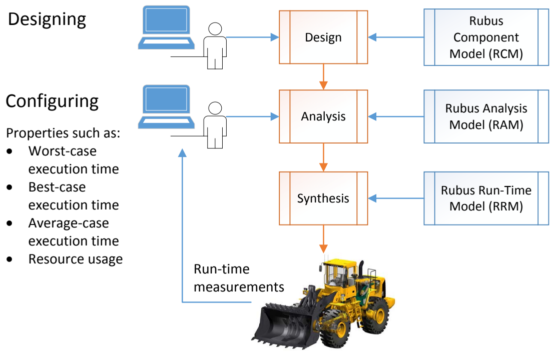
Designing
Configuring
Properties such as:
Worst-case execution time
Best-case execution time
Average-case execution time
Resource usage
Design
Analysis
Synthesis
Rubus Component Model (RCM)
Rubus Analysis Model (RAM)
Rubus Run-Time Model (RRM)
Run-time measurements

Fig. 4. Rubus Conceptual Models.

C. Certified Real-time Operating System (RTOS)

. The Rubus RTOS provides support for RCM in achieving an optimised real-time software system. The RTOS has been utilised in a wide variety of real-time applications such as wheel loaders, articulated haulers, excavators and several four-wheel drive vehicles. The main features of the RTOS include the support for the execution and communication among time-, event- and interrupt-triggered threads, static allocation of resources, scalability and portability. The combination of dynamic and static scheduling supported by the Rubus RTOS enables the design of optimised real-time software systems. The RTOS has been ported to various targets and development environments including Freescale MPC-processors, Texas DSP, Infineon xc167-processors and various compiler environments, e.g., Green Hills, WindRiver, Tasking, Microsoft VS and GCC.

III. Related Work

This paper targets the vehicle industrial domain where the main focus is on EAST-ADL [15] and EAST-ADL-like models for functional modelling and on AUTOSAR [16], RCM and other models for the implementation of software architecture.

A. EAST-ADL and AUTOSAR

EAST-ADL is an architecture description language in the automotive domain. EAST-ADL is inspired by the SysML language [17] that is used for the systems engineering [18]. EAST-ADL is mapped to several automotive standards including ISO26262 for functional safety and AUTOSAR for the implementation of software architecture. It defines a top-down development methodology that advocates the separation of concerns principle by defining various abstraction levels for the development of vehicle software. Fig. 5 shows the abstraction levels along with various methodologies, models, languages and tools that are used at each level. The highest abstraction level, called the vehicle level, informally captures the features and requirements on the end-to-end functionality of the vehicle in a solution-independent way. The analysis level formally captures the requirements in an allocation-independent way. This level supports a high-level analysis for functional verification. The artifacts at the design level are developed independent of implementation details. These artifacts include middleware abstraction, hardware architecture and software functions to hardware allocation. The implementation level provides an implementation of the system functionality in terms of software components and their interconnections.

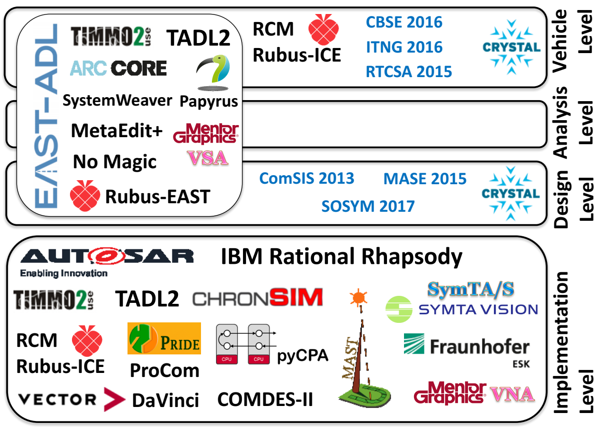

Fig. 5. Models and tools that are used at the abstraction levels of EAST-ADL.

AUTOSAR provides a standardised software architecture for automotive embedded software at the implementation level shown in Fig. 5. It describes the software development at a higher abstraction compared to RCM. Unlike RCM, it does not separate control and data flows among components within a node. AUTOSAR does not differentiate between the modelling of intra- and inter-node communication which is unlike RCM. The timing model in AUTOSAR has been introduced fairly recently compared to that of RCM. We refer the reader to [19] for details about the timing models. There are some similarities between AUTOSAR and RCM, e.g., the sender-receiver communication in AUTOSAR resembles the pipe-and-filter communication in RCM. AUTOSAR is more focused on the functional and structural abstractions, hiding the implementation details about execution and communication.

B. TIMMO, TIMMO2USE, MARTE, TADL and TADL2

TIMMO [20] is an initiative to provide AUTOSAR with a timing model. It is based on a methodology and the TADL language that can express timing requirements on the software architecture. TADL is inspired by the UML profile MARTE [21]. The TIMMO methodology uses EAST-ADL for structural modelling and AUTOSAR for the implementation of the software architecture. Both TIMMO and EAST-ADL focus on the top three levels in Fig. 5. TADL is redefined and released in the TADL2 specification of the TIMMO2USE project [20]. Most of these initiatives lack the support for expressing the low-level details at the higher levels such as linking information in distributed component chains. These details are necessary to extract the timing model from the software architecture to perform the end-to-end timing analysis.

C. Other Related Models, Approaches and Tools

There are several other related component models and modelling approaches such as COMDES-II [22], ProCom [23] and TECS [24]. The timing analysis supported by ProCom is not performed with such a high precision as it is done in Rubus-ICE [25]. To the best of our knowledge, none of these models support the extraction of end-to-end timing models at the higher abstraction levels. This is because these models are developed to model the software architecture only at the implementation level. These models rely on EAST-ADL at the higher abstraction levels. The end-to-end timing models cannot be completely extracted at the higher abstraction levels of EAST-ADL mainly for two reasons: (1) EAST-ADL does not differentiate between the control and data flows, (2) EAST-ADL cannot express the low-level details at the higher levels such as linking information in distributed chains [19]. There are middleware approaches that support the modelling of the software, e.g., RT CORBA, minimum CORBA and CORBA lightweight services [26]. The downside of these approaches is that their run-time frameworks are heavyweight which are not suitable for resource-constrained embedded systems. On the other hand, RCM has a small run-time footprint.

DaVinci[4] is a tool for AUTOSAR-based development. It does not support the extraction of timing models at the higher abstraction levels. The Palladio tool[5] allows for modelling of the software architecture and its analysis based on several quality attributes including response times. However, this tool does not support the end-to-end timing analysis [13], [6]. There are several other tools that support modelling of the systems using the methodology shown in Fig. 5, e.g., Papyrus, Mentor Graphics VSA, Rubus-EAST, EATOP, MetaEdit+, Enterprise Architect, No Magic and System Weaver [27]. These tools are only usable at the first three levels in Fig. 5. None of these tools support the extraction of end-to-end timing models at the higher levels. After implementing the recent research results identified by ComSIS 2013 [6], MASE 2015 [28], RTSCA 2015 [14], CBSE 2016 [29], ITNG 2016 [30] and SOSYM 2017 [31] in Fig. 5, Rubus-ICE supports the development of vehicular embedded systems at all the abstraction levels.

[4] http://vector.com/vi_davinci_developer_en.html.
[5] http://www.palladio-simulator.com/tools/quality_dimensions.

IV. Various Perspectives in the Collaboration

This section discusses the perspectives of various stakeholders in the academic-industrial collaboration shown in Fig. 1. The stakeholders include academia, tool provider and two end users of the tool chain. The end users have contributed to the evolution of the tool chain by not only supplying industrial needs, requirements and use cases but also providing the efficacy of the new methods, techniques and tool extensions.

A. Perspective of the Tools Provider

There is a large number of tool providers in the vehicle domain. In this paper we select the case of the Rubus tool chain for several reasons. The software developed using Rubus has a small run-time foot print (timing and memory overheads) as compared to several other component-based development technologies including AUTOSAR. While most of the tool chains and models need complementary tools from other vendors to support the software development at all abstraction levels of EAST-ADL, the Rubus tool chain is usable at all the abstraction levels. Arcticus and their customers are relatively open in discussing the theories, methods and details about their tools, use cases and experiences with the research community, which is unlikely in the case of many companies. Another reason is the successful usage of Rubus in the vehicle industry for over 20 years. Other reasons include the support for formal and early reasoning about the system functional and extra-functional properties, automatic code synthesis and traceability. According to the CEO of Arcticus Systems, “A tool provider in the vehicle domain should strive to be agile and fast in identifying the industrial needs and challenges and providing solutions based on the state-of-the-art research in close collaboration with academia.”

B. Academic Perspective

RCM was initially developed from the project BASEMENT and its conceptual component model [32]. Since then RCM and its tool chain have evolved through several technology transfer projects. Vice versa, experience and feedback from the industrial use of Rubus have inspired many research projects over the years. A graphical view of different component models that have been developed in academia and by Arcticus Systems in collaboration with various industrial partners is shown in Fig. 6. The number of participating researchers (PhD students and senior researchers) in the projects shown in Fig. 6 ranges from 3 to 25. As a result of this collaboration, the state of the art has been extended in the area of model-based software development of embedded real-time systems by developing new theories, techniques, methods and models as well as by performing several industrial case studies. The scientific results produced within this collaboration have been published in over 100 peer-reviewed research publications[6,7]. According to the research leader at Mälardalen University, “To remain academically excellent and scientifically relevant, we find the long-term cooperation within the collaboration highly invigorating. Over the years the collaboration have had a profound impact on our whole research environment.”

C. End-user Perspective: Volvo Construction Equipment

VCE is the oldest and one of the world-leading manufacturers of heavy construction equipment vehicles, e.g., wheel loaders, articulated haulers and excavators. The company has been successfully applying model-based software development, using the Rubus tool chain, to provide computer-controlled functionality in the vehicles since 1997. According to VCE, Rubus has proven to work efficiently from small applications with only a few objects up to very large applications with thousands of objects. Some of their experiences are as follows.

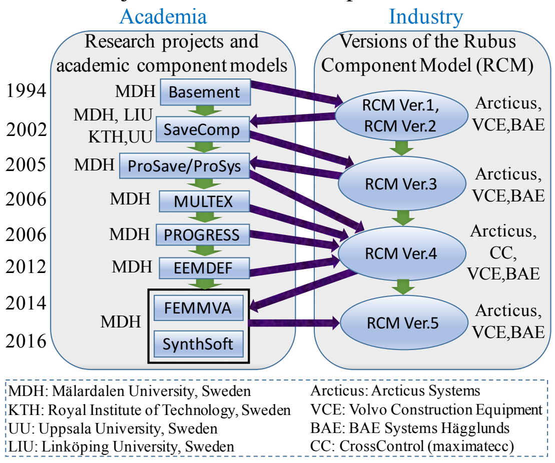
Academia
Industry
Research projects and academic component models
Versions of the Rubus Component Model (RCM)
1994
2002
2005
2006
2006
2012
2014
2016
MDH
MDH, LIU
KTH,UU
MDH
MDH
MDH
MDH
MDH
Basement
SaveComp
ProSave/ProSys
MULTEX
PROGRESS
EEMDEF
FEMMVA
SynthSoft
RCM Ver.1, RCM Ver.2
RCM Ver.3
RCM Ver.4
RCM Ver.5
Arcticus, VCE,BAE
Arcticus, VCE,BAE
Arcticus, CC, VCE,BAE
Arcticus, VCE,BAE
MDH: Mälardalen University, Sweden
KTH: Royal Institute of Technology, Sweden
UU: Uppsala University, Sweden
LIU: Linköping University, Sweden
Arcticus: Arcticus Systems
VCE: Volvo Construction Equipment
BAE: BAE Systems Hägglunds
CC: CrossControl (maximatecc)

Fig. 6. Contributions to RCM and Rubus-ICE from academia and industry.

Development and testing times: Small development time is of essence for VCE. The SWC concept in Rubus offers advantages such as breaking down of an application into smaller components that can be distributed to a large number of developers. This enables development of vehicle functionality in parallel, hence shortening the development time. The support for model-in-the-loop testing and creation of unit tests for each component has resulted in reduction of the amount of bugs that end up during the integration. As a result, time to test has been significantly shortened.

Support for interoperability: Interoperability of information and models during the development of vehicle software is very important for VCE. The Rubus tool chain supports this by allowing its integration with other tools such as Simulink. Using Rubus, there are various means of either using the application programming interfaces to read/write data or directly accessing the model information using scripts like Python.

Information management: Rubus tool chain uses the XML exchange format, which enables it to use existing change management systems such as SVN, ClearCase and GIT. Merging is also facilitated since the information is human readable but still possible for a machine to process. Hence, creating model information from other tools becomes relatively easier.

Pre-runtime timing analysis: A lot of functionality in VCE’s vehicles have real-time requirements. VCE is required to verify such requirements before the functionality is deployed to the vehicles. The pre-runtime end-to-end timing analysis in Rubus-ICE [6] has proven to yield good results with respect to early verification, feasibility validation and timing checks.

Certification: VCE needs to deliver safety certified vehicle functionality. Since the Rubus RTOS is certified according to ISO-26262:2011 ASIL-D, it serves the purpose for VCE.

According to the project manager of platforms development at VCE, ”This collaboration has helped us in improving our strategies for the development of predictable software in our vehicles. In our view, the most attractive features of the tool chain include user-friendly tools, simplicity to understand and use by the engineers, light run-time footprint and automation.”

[6] http://www.es.mdh.se/publications?filter=true&author=63&ppp=all.
[7] http://www.arcticus-systems.com/research/publications/.

D. End-user Perspective: BAE Systems Hägglunds

BAE is a renowned global manufacturer of advanced defense, combat and security systems. The main reasons for selecting the Rubus tool chain for the software development in their products are as follows.

Requirement for the time- and event-triggered execution: BAE requires time-triggered architectural approach to execute the software threads that compose majority of the repetitive functions in their products. The support for event-triggered execution of threads is needed to service various internal and external events in the system. Both the time- and event-triggered modelling and execution of software components are fully supported by RCM, Rubus-ICE and Rubus RTOS.

Requirement to support background threads and services: BAE requires the support to implement background threads that are appropriate for services of diverse character such as diagnostics and as well as vital functions for the implementation of communication protocols. This requirement is fully supported by the Rubus tool chain.

Requirement for pre-runtime timing analysis: BAE requires pre-runtime verification of the predictability of both time- and event- triggered functions. BAE performs pre-runtime verification of time-triggered SWCs by using the correct-by-construction offline schedule generated by the Rubus scheduler. In the case of event-triggered SWCs, response-time analysis and end-to-end timing analysis in Rubus-ICE provides pre-runtime verification of the timing behaviour of the SWCs.

Requirement for coupling the SWCs with Simulink: The behaviour of control functions are developed using Simulink. Hence, it is required to couple each SWC in the software architecture with Simulink. The coupling is achieved at a suitable level via a modified code generator. The generated code becomes an entry point for the SWC, which is then allocated to a time-triggered chain with an appropriate periodicity.

According to the senior software architect of land vehicles at BAE Systems, “This long-term collaboration has been very beneficial for us in getting the support of a useful tool chain. With the time-triggered execution in Rubus, majority of our functions are fully supported due to their repetitive nature. The communication and diagnostics in our systems are well supported by utilising the event-triggered execution in Rubus. Interoperability between Rubus and Simulink eases our effort to develop the control logic.”

V. Ongoing Collaboration: the PreView Project

Contemporary functions and advanced features in modern vehicles require new levels of computational power, true parallelism and more complex coordination among subsystems. Majority of ECUs that are used in the vehicle industry include single-core processors which are unable to fulfil these requirements. ECUs with multi-core processors offer an efficient support for running such computation-intensive vehicle functions by executing various activities in parallel [33]. As compared to single-core ECUs, the multi-core ECUs are more prone to unpredictable behaviour due to sharing of caches and memory banks among the cores. This problem is expected to increase in the future with the introduction of more advanced architectures with cluster sets of cores having more advanced memory interconnects, e.g., many-core platforms. A seamless tool chain for model- and component-based development of vehicle software and its predictable execution on multi-core ECUs is still missing in the vehicle industry. Providing such a tool chain is the main focus of an ongoing project PreView [34] within the collaboration. Various activities in this project are depicted in Fig. 7. Other ongoing projects[8] within the collaboration that are in line with this goal include EMC[2] [35] and DPAC [36]. The collaboration is also developing a generic approach to model transformations in order to seamlessly exchange information with other tool chains in the vehicular domain.

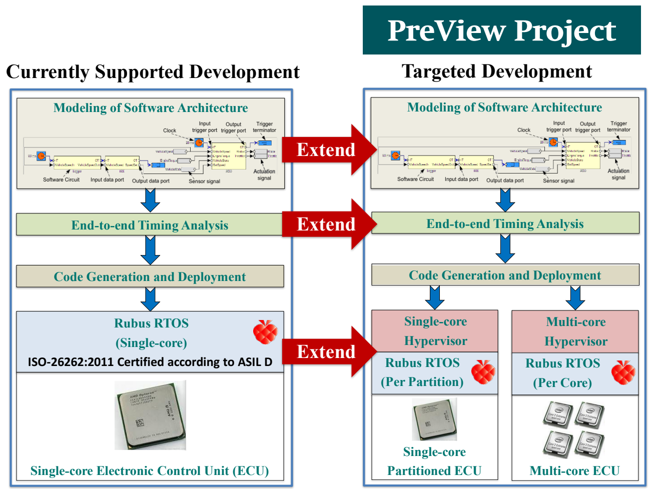

Fig. 7. Ongoing research activities within the collaboration.

VI. Experiences and Lessons Learned

This section discusses important experiences and lessons learned from the technology transfer projects shown in Fig. 6.

A. Simplicity, Learning Time and Ease of Use

Simplicity, short learning time and ease of use are very important aspects of a tool chain that are associated to its industrial adoption. No doubt, the tool chain can implement complex methods, analysis techniques and optimisations; however, such complexities should be abstracted from the end user. It can be challenging to hide the complexities in many cases. In this regard, an effort should be made to minimise the overhead that is related to the learning time of engineers when the tool chain is extended/upgraded. We faced this challenge in the EEMDEF project (see Fig. 6) when modelling and timing analysis support in the Rubus tool chain was extended from single-node to distributed embedded systems. In order to address this challenge the implementer (a researcher who developed the techniques and implemented them in the stand-alone tools) and the integrator (an engineer with limited experience of integrating complex modelling and real-time analysis techniques but fully aware of overall objective) had to not only implement, integrate and test the techniques but also deliver tutorials, workshops, usage methodology and simple use cases to the end users. The later activities contributed towards minimising the learning efforts by the end users of the extended tool chain. We expect to face the same challenge in an ongoing project PreView [34] (see Fig. 6) that aims to extend the model- and component-based software development and end-to-end timing analysis support in the Rubus tool chain from single-core to multi-core platforms.

[8] http://www.arcticus-systems.com/research/.

B. Implementation, Integration and Testing Efforts

In the vehicle industry, a two-step approach is used to transfer research results to an existing tool chain that supports the plugin mechanism for the integration of new tools. In the first step, the research results which have already been formally proven for correctness by the research community are implemented as a stand-alone tool. A typical challenge in this step is to show that the implementation is correct. This challenge can be addressed by implementing the sanity checking and error handling routines as part of the implementation. In addition, a significant number of test cases are created and then verified to show the correctness of the implementation. Model checkers like UPPAAL [37] can be helpful in this step.

In the second step, the stand-alone tool is plugged in with the tool chain. The functionality of the plugin is verified again because new bugs might have been introduced by the glue code. Moreover, the input and output interfaces of the plugin can be different from those in the first step. For the verification during this step, many small and large applications with varying architectures are created, modelled and analysed by the extended tool chain. In addition, the implementer has to verify these applications by hand or by means of semi-automatic tools that are specifically developed for this purpose. The correctness of the extended tool chain is verified by comparing the results provided by the tool chain and the results that are calculated manually or semi-automatically. In crux, the second step requires continuous consultation and communication between the plugin integrator and the implementer of the research results. The second step, in particular, integration testing is a non-trivial and time-consuming activity. In our experience from several technology transfer projects, the second step requires 4-5 times more time and effort than the first step. It is important to carefully consider this aspect in the planning for any technology transfer project.

C. Feedback from the End Users

The end users of the tool chain have been asked to provide their feedback after each technology transfer project shown in Fig. 6. One question that is common in the feedback received from several projects is concerned with how can the tool chain assist engineers in converting an infeasible system into a feasible one. Assume that the end-to-end timing analysis is performed on the software architecture of a system. The analysis results show that some of the timing requirements are violated. How can the tool chain guide the engineer to make better design decisions or modify timing properties on the software architecture such that the timing requirements are met? It is not trivial for a tool chain to provide this feedback because there can be many reasons behind the system being unschedulable. Good news is that there are several heuristic solutions that can be implemented in a tool chain to provide such feedback. One possible solution is achieved by converting *data chains* in the software architecture to *trigger chains* and/or *mixed chains* [6]. Each software component in a data chain is activated independently. Whereas in a trigger chain, the first component is activated independently while the rest of the components are activated by their predecessors. A mixed chain is a combination of a data and a trigger chain. Another solution is to implement a trace analyser in the tool chain. This analyser records the execution of the system and then presents a graphical comparison of the trace with the calculated end-to-end delays. Such a comparison can help the engineer to acquire better understanding of the system's schedulability. Another solution is to automatically apply job-level dependencies in component chains to reduce the end-to-end delays [38].

D. Legacy Support

An extended tool chain should not only allow the development of new systems but also support extension of legacy (previously developed) systems. The tool chain should also support modelling and end-to-end timing analysis of the systems that incorporate legacy ECUs, third-party propriety ECUs and COTS components. It becomes challenging to model and perform the end-to-end timing analysis of these systems because the software architectures of these ECUs are hidden or not available. For example, the ECUs in the steer-by-wire system are provided by one tier-1 supplier. Whereas, another tier-1 supplier extends the steer-by-wire system by adding the collision-avoidance functionality. In order to deal with this challenge, the black-box modelling and end-to-end timing analysis technique [29], [14] can be implemented in a tool chain. These techniques also allow model refinements early during the development of vehicle software by supporting early end-to-end timing analysis of the software architecture.

E. Academic Assumptions Vs Industrial Limitations

Often, there exists a gap between the assumptions on which research results are based and the practical limitations that are found in the industrial settings. When the research results are implemented in an industrial tool chain, this gap should be minimised so that the assumptions correctly correspond to the industrial limitations. For instance, if timing analysis is implemented in a tool chain then the gap can be minimised by considering offsets in the timing analysis [39] instead of using pessimistic assumptions such as "all events in the system happen at once" [5]. Similarly, the assumptions in the timing analysis should correctly match with practical limitations in the interface controllers, types and size of message queues, and transmission patterns supported by the higher-level protocols/commercial extensions of in-vehicle networks. This lesson was learned during the EEMDEF project (see Fig. 6). Consequently, the existing analysis for CAN was extended [12]. If the assumptions do not correctly match with the limitations, the tool chain can provide underestimated analysis results, which could result in the failure of the vehicular system.

F. Computation Time Complexity Vs Accuracy of the Tool

The analysis or optimisation tool that takes shorter time to provide somewhat overestimated but safe results has a better chance of industrial adoption as compared to the tool that takes much longer time to provide exact results. For instance, the real-time analyses are often iterative [6]. The tools implementing the analyses may take significantly large time while analysing a complex system. Provisioning of timing analysis tools that provide tight results in short times (in the order of few minutes or less) was one of the main requirements in several technology transfer projects shown in Fig. 6.

G. Interoperability

Providing a seamless tool chain that consists of various tools for the development of vehicle software is challenging. This is because of the mismatch that exists between structural and semantic assumptions in modelling languages that are used in different phases. This mismatch causes large problems when design artefacts are transformed among the corresponding tools. Productivity in the industry is affected by ad hoc, non-trivial, manual and tedious translations due to incompatibility among the tools and their file formats. In this context, there is a strong need to investigate two questions. First, how to effectively and efficiently work with the tools at various development phases. Second, how to support automated or semi-automated translations among the tools at various development phases while preserving their semantics. According to the industrial members within the collaboration, "Automated interoperability to allow information exchange among software development tools and models for cyber-physical systems is going to be the next big thing in the vehicle industry."

VII. Conclusion

This paper has discussed a success story of the collaboration between academia and the vehicle industry. The collaboration has resulted in sustained development and evolution of the Rubus component model and tool chain by leveraging the state-of-the-art research results and the industrial needs. The Rubus tool chain supports model- and component-based development of vehicle software and a well-proven and safety certified real-time operating system for its execution. The collaborators form a clear value chain, state-of-the-art techniques are developed by academia, the tool developers implement these techniques in the tool chain, and finally, the end users employ the tool chain for reliable and cost-effective development of vehicle software. The industrial impact of the collaboration can be seen from the successful use of the tool chain in the vehicle industry for over two decades. Whereas, its scientific impact is evident from the fact that there are over 100 peer-reviewed scientific publications that directly/indirectly include the concepts, techniques, methods and models behind the tool chain as well as the related prototypes and case studies. The paper has also discussed the perspectives of various stakeholders in the collaboration and provided experiences and lessons learned during several technology transfer projects in the vehicle domain. We believe, the experiences and lessons learned in this paper can provide useful guidance to researchers as well as to the tool developers, integrators, application developers and practitioners during any technology transfer project in the vehicle domain. Based on the industrial impact of the techniques developed within this collaboration, we believe that the tools implementing these techniques can prove helpful for the vehicle manufacturers to decrease the costs for the software development, configuration and testing.

Acknowledgement

The work in this paper is supported by the Swedish Knowledge Foundation within the project PreView. We thank our all industrial partners, especially Arcticus Systems, Volvo Construction Equipment and BAE Systems Hägglunds, Sweden.

References

[1] M. Broy, I. Kruger, A. Pretschner, and C. Salzmann, "Engineering automotive software," *Proceedings of the IEEE*, vol. 95, no. 2, pp. 356 –373, Feb. 2007.

[2] T. Scott, CTO of Information Systems and Services Division, General Motors Coorporation, "Keynote Talk," in *CeBIT America Conference*, May 2004.

[3] I. Crnkovic and M. Larsson, *Building Reliable Component-Based Software Systems*. Norwood, MA, USA: Artech House, Inc., 2002.

[4] D. C. Schmidt, "Guest editor's introduction: Model-driven engineering," *Computer*, vol. 39, no. 2, pp. 25–31, Feb. 2006.

[5] N. Audsley, A. Burns, R. Davis, K. Tindell, and A. Wellings, "Fixed priority pre-emptive scheduling: an historic perspective," *Real-Time Systems*, vol. 8, no. 2/3, pp. 173–198, 1995.

[6] S. Mubeen, J. Mäki-Turja, and M. Sjödin, "Support for end-to-end response-time and delay analysis in the industrial tool suite: Issues, experiences and a case study," *Computer Science and Information Systems*, vol. 10, no. 1, 2013.

[7] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, "System level performance analysis - the symta/s approach," *Computers and Digital Techniques*, vol. 152, no. 2, pp. 148–166, Mar. 2005.

[8] "Rubus models, methods and tools," http://www.arcticus-systems.com.

[9] ISO 26262-1:2011: Road vehicles in Functional safety. http://www.iso.org/.

[10] K. Hänninen et.al., "The Rubus Component Model for Resource Constrained Real-Time Systems," in *3rd IEEE International Symposium on Industrial Embedded Systems*, Jun. 2008.

[11] J. Mäki-Turja and M. Nolin, "Tighter response-times for tasks with offsets," in *Real-time and Embedded Computing Systems and Applications Conference (RTCSA)*, Aug. 2004.

[12] S. Mubeen, J. Mäki-Turja, and M. Sjödin, "Integrating Mixed Transmission and Practical Limitations with the Worst-Case Response-Time Analysis for Controller Area Network," *Journal of Systems and Software*, vol. 99, pp. 66 – 84, 2015.

[13] N. Feiertag, K. Richter, J. Nordlander, and J. Jonsson, "A Compositional Framework for End-to-End Path Delay Calculation of Automotive Systems under Different Path Semantics," in *Workshop on Compositional Theory and Technology for Real-Time Embedded Systems*, Dec. 2008.

[14] S. Mubeen, M. Sjödin, T. Nolte, J. Lundbäck, M. Gålnander, and K.-L. Lundbäck, "End-to-end Timing Analysis of Black-box Models in Legacy Vehicular Distributed Embedded Systems," in *21st International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, Aug. 2015.

[15] "EAST-ADL Domain Model Specification, V2.1.12,," http://www.east-adl.-info/Specification/V2.1.12/EAST-ADL-Specification_V2.1.12.pdf.

[16] "AUTOSAR Techincal Overview, Release 4.1, Rev. 2, Ver. 1.1.0., The AUTOSAR Consortium, Oct., 2013," http://autosar.org.

[17] Systems Modeling Language, Ver. 1.4., Sep. 2015, http://www.omgsysml.org,.

[18] D. Chen, L. Feng, T. Qureshi, H. Lnn, and F. Hagl, "An architectural approach to the analysis, verification and validation of software intensive embedded systems," *Computing*, vol. 95, no. 8, pp. 649–688, 2013.

[19] S. Mubeen, J. Mäki-Turja, and M. Sjödin, "Communications-Oriented Development of Component- Based Vehicular Distributed Real-Time Embedded Systems," *Journal of Systems Architecture*, vol. 60, no. 2, pp. 207–220, 2014.

[20] TIMMO2USE project. https://itea3.org/project/timmo-2-use.html, acc. May, 2016.

[21] "The UML Profile for MARTE: Modeling and Analysis of Real-Time and Embedded Systems," Jan. 2010. [Online]. Available: http://www.omgmarte.org/

[22] X. Ke, K. Sierszecki, and C. Angelov, "COMDES-II: A Component-Based Framework for Generative Development of Distributed Real-Time Control Systems," in *IEEE RTCA*, Aug. 2007.

[23] S. Sentilles, A. Vulgarakis, T. Bures, J. Carlson, and I. Crnkovic, "A Component Model for Control-Intensive Distributed Embedded Systems," in *the 11th International Symposium on Component Based Software*, 2008, pp. 310–317.

[24] A. Ohno, T. Azumi, and N. Nishio, "TECS components providing functionalities of OSEK specification for ITRON OS," *Journal of Information Processing*, vol. 22, no. 4, pp. 584–594, 2014.

[25] S. Mubeen, T. Nolte, M. Sjödin, J. Lundbäck, and K.-L. Lundbäck, "Supporting Timing Analysis of Vehicular Embedded Systems through the Refinement of Timing Constraints," *Accepted for publication in the Journal of Software and Systems Modeling, DOI: 10.1007/s10270-017-0579-8*, 2017.

[26] "Catalog of Specialized CORBA Specifications. OMG Group." [Online]. Available: http://www.omg.org/technology/documents/

[27] EAST-ADL Tooling, http://www.east-adl.info/Tooling.html, accessed Jan. 2017.

[28] A. Bucaioni, A. Cicchetti, F. Ciccozzi, R. Eramo, S. Mubeen, and M. Sjödin, "Anticipating implementation-level timing analysis for driving design-level decisions in east-adl," in *International Workshop on Modelling in Automotive Software Engineering*, Sep. 2015.

[29] S. Mubeen, T. Nolte, M. Sjödin, J. Lundbäck, M. Gålnander, and K.-L. Lundbäck, "Modeling of Legacy Distributed Embedded Systems at Vehicle Abstraction Level," in *19th International Symposium on Component Based Software Engineering*, Apr. 2016.

[30] S. Mubeen, T. Nolte, J. Lundbäck, M. Gålnander, and K.-L. Lundbäck, "Refining Timing Requirements in Extended Models of Legacy Vehicular Embedded Systems Using Early End-to-end Timing Analysis," in *13th International Conference on Information Technology: New Generations (ITNG)*, Apr. 2016.

[31] S. Mubeen, T. Nolte, M. Sjödin, J. Lundbäck, and K.-L. Lundbäck, "Supporting timing analysis of vehicular embedded systems through the refinement of timing constraints," *Software & Systems Modeling*, pp. 1–31, 2017. [Online]. Available: http://dx.doi.org/10.1007/s10270-017-0579-8

[32] H. Hansson, H. Lawson, O. Bridal, C. Eriksson, S. Larsson, H. Lon, and M. Stromberg, "Basement: an architecture and methodology for distributed automotive real-time systems," *IEEE Transactions on Computers*, vol. 46, no. 9, pp. 1016–1027, Sep. 1997.

[33] Roland Berger, Consolidation in Vehicle Electronic Architectures, In Think: Aact, Jul., 2015. Available at: https://www.rolandberger.com/en/Publications/pub_-consolidation_in_vehicle_electronic_architectures.html, accessed October, 2016.

[34] PreView: Developing Predictable Vehicle Software on Multi-core, http://www.es.mdh.se/projects/442-PreView, acces. Oct., 2016.

[35] EMC^2 project. http://www.artemis-emc2.eu/, accessed Jan., 2017.

[36] DPAC - Dependable Platforms for Autonomous systems and Control, http://www.es.mdh.se/projects/414-DPAC, acces. Oct., 2016.

[37] K. G. Larsen, P. Pettersson, and W. Yi, "Uppaal in a nutshell," *In Springer International Journal of Software Tools for Technology Transfer,*, vol. 1, no. 1+2, pp. 134–152, May 1997.

[38] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte, "Synthesizing job-level dependencies for automotive multi-rate effect chains," in *RTCSA*, 2016.

[39] J. Mäki-Turja and M. Nolin, "Efficient implementation of tight response-times for tasks with offsets," *Real-Time Syst.*, vol. 40, no. 1, pp. 77–116, 2008.