Virtual Node - To Achieve Temporal Isolation and Predictable Integration of Real-Time Components

Rafia Inam, *Student Member, IEEE*, Jukka Mäki-Turja, Jan Carlson, and Mikael Sjödin

***Abstract*—We present an approach of two-level deployment process for component models used in distributed real-time embedded systems to achieve predictable integration of real-time components. Our main emphasis is on the new concept of virtual node with the use of a hierarchical scheduling technique. Virtual nodes are used as means to achieve predictable integration of software components with real-time requirements. The hierarchical scheduling framework is used to achieve temporal isolation between components (or sets of components). Our approach permits detailed analysis, e.g., with respect to timing, of virtual nodes and this analysis is also reusable with the reuse of virtual nodes. Hence virtual node preserves real-time properties across reuse and integration in different contexts.**

***Index Terms*—Hierarchical scheduling, real-time systems, reusability, component-based software-engineering.**

I. Introduction

COMPONENT integration can be explained as the mechanical task of wiring components together [1]. Since it is rare that two components perfectly match, component integration requires more than just matching the needs and services of one component with the needs and services of others. In real-time embedded systems the components and components integration must satisfy both (1) functional correctness and (2) extra-functional correctness, such as satisfying timing properties.

Temporal behavior of the real-time components poses more difficulties in their integration. When multiple components are deployed on the same hardware node, the timing behavior of each of the components is typically altered in unpredictable ways. This means that a component that is found correct during unit testing may fail, due to a change in temporal behavior, when integrated in a system. Even if a new component is still operating correctly in the system, the integration could cause a previously integrated (and correctly operating) component to fail. Similarly, the temporal behavior of a component is altered if the component is reused in a new system. Since also this alteration is unpredictable, a previously correct component may fail when reused.

Some of these problems can be solved using scheduling analysis [2][3], however these techniques only allow very simple models; typically simple timing attributes such as period and deadline are used. Components often exhibit a too complex behavior to be amenable for scheduling analysis. And, even if a suitable analysis technique should exist, such analysis requires knowledge of the temporal behavior of all components in the system. Thus, a component cannot be deemed correct without knowing which components it is integrated with. As a result, the reusability of a component is restricted since it is very difficult to know beforehand if the component will pass a schedulability test in a new system.

For large-scale real-time embedded systems, methodologies and techniques are required to provide temporal isolation so that the run-time timing properties could be guaranteed. Further the real-time properties of the components should be maintained for their reuse in large-scale industrial embedded systems.

Contributions:

The main contributions of this paper are as follows:

- We propose the concept of a *Virtual Node (VN)*, which is an execution-platform concept that preserves temporal properties of the software executed in the virtual node [4, 5]. The virtual node is intended for coarse-grained components for single node deployment and with potential internal multitasking.
- We propose to *integrate hierarchical scheduling framework (HSF) [6] within the components (virtual nodes)* to realize our ideas of providing temporal properties of real-time components, their predictable integrations and reusability.
- We describe how *the virtual node can be applied in the run-time infrastructure* in three different component technologies: ProCom [7, 8], AUTOSAR [9], and AADL [10].

Paper Outline: Section II describes the component technologies we study in this paper. In section III we describe the virtual node execution-mechanism and the

This work was supported in part by the Swedish Foundation for Strategic Research (SSF), via the research program PROGRESS.

R. Inam, J. Mäki-Turja, J. Carlson and M. Sjödin are with the Mälardalen Real-Time Research Centre, SE-721 23 Västerås, Sweden. (e-mail: rafia.inam@mdh.se; jukka.maki-turja@mdh.se; jan.carlson@mdh.se; mikael.sjodin@mdh.se).

hierarchical scheduling framework used by the virtual node. We explain the usage of virtual node in the above mentioned three component models in section IV, in section V we conclude the paper and present the future work to be done.

II. Component Technologies

Component-Based Software Engineering (CBSE) and Model-Based Engineering (MBE) are two emerging approaches to develop embedded control systems like software used in trains, airplanes, cars, industrial robots, etc. In this section we briefly outline the component technologies we will target in our work. We discuss three representatives of technologies that use component-based software engineering (AUTOSAR), model-based engineering (AADL) and a combination of CBSE and MBE (ProCom).

We present related work from the perspective of deployment of the components on physical platform and the generation of final executables of the system in the above mentioned technologies.

A. ProCom

ProCom component model combines both CBSE and MBE techniques for the development of the system parts, hence also exploits the advantages of both. It takes advantages of encapsulation, reusability, and reduced testing from CBSE. From MBE it makes use of automated code generation and performing analysis at an earlier stage of development. In addition ProCom achieves additional benefits of combining both approaches (like flexible reuse, support for mixed maturity, reuse and efficiency tradeoff) [4].

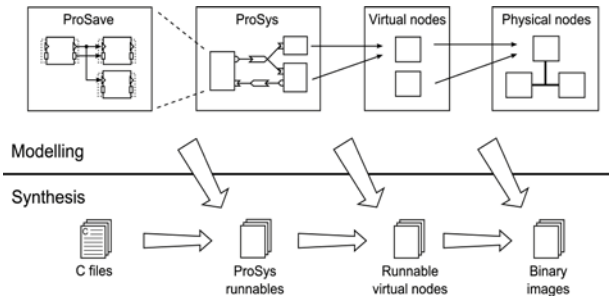

Fig. 1. The ProCom component model: Overview of the modeling- and runnable realms.

The ProCom component model can be described in two distinct realms: modeling and runnable realms as shown in Figure 1. In the modeling realm the models are made using CBSE and MBE, while in the runnable realm the synthesis of runnable entities is done from the model entities. Both realms are explained as follows:

The Modeling Realm: Modeling in ProCom is done by four discrete but related formalisms as shown in Figure 1. The first two formalisms relate to the system functionality modeling while the later two represent the deployment modeling of the system.

Functionality of the system is modeled by the ProSave and ProSys components at different levels of granularity. The basic functionality (data and control) of a simple component is captured in ProSave component level, which is passive in nature. At the second formalism level many ProSave components are mapped to make a complete subsystem called ProSys that is active in nature. Both ProSys and ProSave allow composite components. For details on ProSave and ProSys, including the motivation for separating the two, see [7], [8].

The deployment modeling is used to capture the deployment related design decisions and then mapping the system to run on the physical platform. Many ProSys components can be mapped together on a virtual node (many-to-one mapping) together with a resource budget (i.e. CPU usage and memory requirements) required by those components.

After that many virtual nodes could be mapped on a physical node i.e. an ECU: an electronic control unit. The relationship is again many-to-one. This part represents all the physical nodes, their intercommunication through the network and the type of the network etc. Figure 2 represents how four virtual nodes (VN1, VN2, VN3, and VN4) are allocated to the three physical nodes (Node1, Node2, and Node3). Details about the deployment modeling are provided in [4].

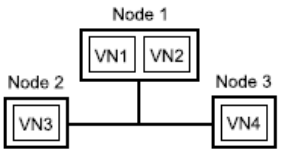

Fig. 2. Allocation of the virtual nodes to the physical nodes.

The Runnable Realm: is the synthesis of the runnables/executables from the ProCom model entities. The primitive ProSave components are represented as simple C language source code in runnable form. From this C code the ProSys runnables are generated which contains the collection of operating system tasks. Virtual node runnables will implement the local scheduler and will contain the server task. Hence virtual node runnable actually encapsulates the set of tasks, resource allocations, and a real-time scheduler within a server in a two-level hierarchical scheduling framework. Final binary image will be generated by connecting different virtual nodes together with a global scheduler and using some middleware to provide intra-communications among the virtual node executables. As this work is going on, some of the details about the runnable realm are given in [5].

Deployment—Two-steps Process: Rather than deploying a whole system in one big step, the deployment of the ProCom components on the physical platform is done in the following two steps:

1) First the ProSys subsystems are deployed on an intermediate node called *Virtual Node*. The allocation of ProSys subsystems to the virtual nodes is many-to-one relationship. The additional information that is added at this step is the *resource budgets*.

2) The virtual nodes are then deployed on the physical nodes. The relationship is again many-to-one means more than one virtual node can be deployed to one physical node.

This two-steps deployment process allows not only the detailed analysis in isolation from the other components to be deployed on the same physical node, but once checked for correctness, it also preserves its temporal properties for further reuse of this virtual node as an independent component. Section III describes this further.

B. AUTOSAR

AUTomotive Open System ARchitecture (AUTOSAR) [9] is an open standard for automotive electronics architecture. It is developed by a number of automotive manufacturers and suppliers to deal with the increasing complexity and to fulfill a number of future vehicle requirements (such as safety and availability, driver assistance, software updates, environment, and infotainment). The key features of AUTOSAR are modularity, configurability, standardized interfaces and a runtime environment. It provides standardized modular software infrastructure and basic software for embedded automotive systems. A layered-software platform has been developed to achieve modularity, scalability, transferability, and reusability of components.

AUTOSAR methodology is a standardized technique that describes all the major steps in a complete development cycle of a system. It encloses all steps from the system level configurations till the generation of ECU executable binaries.

Functional software is developed using component-based approach. A component is developed over many layers of AUTOSAR, including: Application layer, Runtime Environment (RTE), Basic software and ECU hardware as shown in Figure 3. Some important layers are:

• Application layer resides at the top of RTE. At this layer, an application consists of one or many AUTOSAR software components and sensor/actuator components.

• RTE connects AUTOSAR components. It is responsible for configurations and communication among components. It enables both communication between components on the same ECU and also communication between components on different ECUs. Hence it makes the components completely independent from the underlying hardware. Components communicate with each other using ports (e.g., PPort, RPort) and port interfaces (e.g., client-server, sender-receiver).

• Basic software (BSW) provides services to Input/ Output (I/O), communication, memory, and system. It has access to hardware (e.g., sensors, actuators), Internal/External memory, microcontroller onboard peripheral devices and communications. BSW consists of Internal drivers (e.g., EEPROM, CAN, etc.), external drivers (e.g., external EEPROM, etc.), Interfaces that offer generic API for upper layers, handlers, and managers. BSW uses complex drivers to handle timing and functional requirements of complex sensors and actuators.

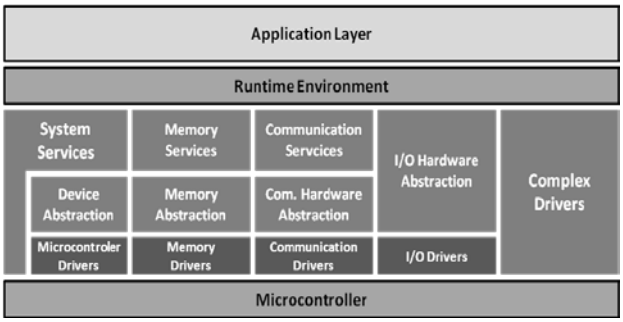

Fig. 3. AUTOSAR layered architecture [9].

• Microcontroller Abstraction layer resides at the bottom just above the underlying ECUs. It separates the above layers from the hardware and provides standardized interfaces for communication of upper layers to the ECU.

Software component (SW-C) at ECU level contains at least one or several runnable entities (or simply runnables). A runnable is small fragment of sequential code within a component. Runnable entities are grouped into operating system tasks executed on ECUs. Runnables grouped onto one task may belong to different software components. Operating system controls and schedules these tasks. These OS tasks can be of one of the categories, basic tasks (Category1 without WaitEvent) or extended tasks (Category2 with WaitEvent). All runnables are activated by RTEEvents.

Deployment: Deployment in AUTOSAR begins when RTE generator maps all runnables to the OS tasks and build inter-ECU and intra-ECU communications among them. This mapping is dependent on different extra-functional properties and behaviors of the runnables e.g., runnable with Category1 will be mapped differently from the runnable with Category2. Three different rules for mapping are given in the AUTOSAR RTE specifications [11]. After mapping, RTE generator configures each ECU. In the last, the OS tasks bodies are constructed by RTE generator.

The main disadvantage of AUTOSAR is that it lacks clear and well-defined timing properties that further affect the execution semantics too. A tool suite supporting the complete AUTOSAR methodology is still missing.

C. AADL

Architecture Analysis and Design Language (AADL) was developed as a SAE Standard AS-5506 [10] in 2004 to design and analyze software and hardware architectures of distributed real-time embedded systems. It supports MBE and has both textual and graphical representations. It also supports syntax and semantics analyses of the language. Modeling of software and hardware parts is supported by software components (e.g., process, data, thread, thread group, subprogram), and execution platform components (e.g., processor, memory, bus, device) respectively. It also allows hybrid components (e.g., system) [12]. Properties and new functional aspects can be attached to the elements (e.g., components, connections) using the properties defined in the SAE standard, and communication among components is performed using component interfaces i.e., ports. Ocarina [12] is a tool suite by Telecom Paris that facilitates the design of AADL models and their mapping on a hardware platform, assessment of these models (e.g., syntactic/semantic analysis, schedulability analysis performed by Ocarina and Cheddar [13]), and then automatic code generation from these models and their deployment.

Deployment: Automatic code generation is done using the Ocarina compiler [12] that comprises of two traditional parts: the frontend and the backend.

1) The frontend is responsible for lexical checking, syntactic analysis, semantic analysis and instantiation. It generates the lexems, then generates the abstract syntax tree and add semantics to it, and at the last step produces the instance tree. It also scrutinizes all syntactic, semantic and instance errors and warnings.

2) The backend part is responsible for code generation in three steps; first the expansion of instance tree, second the conversion of this instance tree into a syntactic tree of the target language (Ada or C) and the last step is the code generation that generates the code in C or Ada language.

Ocarina supports code generation in Ada and C languages using a middleware API called PolyORB (PolyORB for Ada while PolyORB-HI for C). This middleware provides execution services and wraps the POSIX API, hence it is POSIX compliant. Runnable entities are presented by processes. A process contains many tasks and it is a selfcontained runnable entity that executes on a hardware platform without any programmatic dependencies. The final executable binaries are generated by compiling the Ocarina automatic generated code (in C or Ada) together with the user written application code (in C or Ada) and the AADL runtime (e.g. PolyORB, PolyORB-HI).

POK is another type of runnable entity for AADL is implemented by Julien [14]. This technique is an extension of the first one implemented by Ocarina. It employs a hierarchical scheduling concept in a partition. A partition is a combination of several processes and a scheduler called Virtual Processor. Each partition is isolated in terms of space and time. Each process again encloses several tasks and a local scheduler. A local scheduler schedules all the tasks of a particular process. Virtual Processor is then responsible for scheduling all the processes in a particular partition.

III. VIRTUAL NODE

The concept of virtual node is used to achieve not only temporal isolation and predictable temporal properties of real-time components but also to get better reusability of components with real-time properties. Further it reduces the efforts related to testing, validation and certification. This concept is based on two-level deployment process. It means that the whole system is generated in two steps rather than a big synthesis step. At the first level of deployment, functionality (in form of design-time components) is deployed to virtual nodes, and virtual nodes are assigned execution resources. In this way behavior is encapsulated with respect to timing and resource usage and VN becomes a reusable component in addition to the design-time components. In the second level of deployment, these virtual nodes are deployed on a physical platform together with a global scheduler [5].

A virtual node includes the executable representation of the components (e.g. a set of tasks), a resource allocation, and a real-time scheduler to be executed within a server in the hierarchical scheduling framework. Hierarchical scheduling is described as follow:

A. Hierarchical Scheduling Framework (HSF)

A two-level Hierarchical Scheduling Framework (HSF) [6] is used to provide the temporal isolation among the virtual nodes. In hierarchical scheduling, the CPU is partitioned into a set of servers, each server can use a different scheduling policy, and are in turn scheduled by a global (system-level) scheduler. Hence a two-level HSF can be viewed as a tree with one parent node (global scheduler) and many leaf nodes (local schedulers) as illustrated in Figure 4.

The leaf nodes contain its own internal set of tasks that are scheduled by a local (subsystem-level) scheduler. The parent node is a global scheduler that schedules local schedulers. Using an appropriate HSF, subsystems can be developed and analyzed in isolation from each other. As each subsystem has its own local scheduler, after satisfying the temporal constraints, the temporal

properties are saved within each subsystem. Later, a global scheduler is used to combine all the subsystems together without violating the temporal constraints that are already analyzed and stored in them. Accordingly we can say that the HSF provides partitioning of the CPU between different servers. Thus, server-functionality can be isolated from each other for, e.g., fault containment, compositional verification, validation and certification, and unit testing.

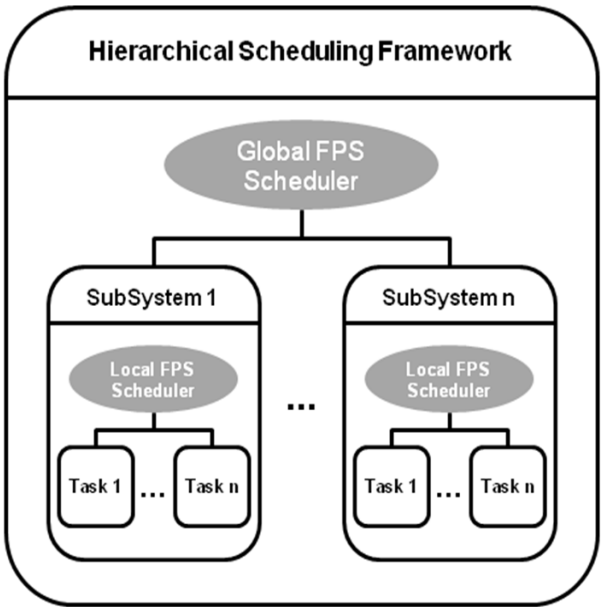

Fig. 4. Two-level hierarchical scheduling framework.

Using HSF a subsystem (virtual node in our case) can be developed and analyzed in isolation, with its own local scheduler at first step of deployment and its temporal properties are preserved. Then at the second step of deployment an arbitrary global scheduler is used for the integration of multiple subsystems (virtual nodes) without violating the temporal properties of the individual subsystems analyzed in isolation. A brief overview of our hierarchical scheduling framework implementation is given here.

HSF implementation in FreeRTOS: The two-level hierarchical scheduling implementation is done independently from components [17][18]. We have chosen FreeRTOS [16], a portable open source real-time scheduler for the implementation. Its main properties like open source, small and scalable, support for 23 different hardware architectures, and ease to extend and maintain makes it a perfect choice to be used within the PROGRESS project [7][8]. The motivations for choosing FreeRTOS and the details about its real-time kernel are provided in [17][18].

We have implemented time-triggered periodic tasks within the FreeRTOS operating system to support hard real-time components. The HSF implementation supports two kinds of servers, idling periodic and deferrable servers. The implementation uses fixed priority preemptive scheduling (FPPS) for both global and local-level scheduling. FPPS is flexible and simple to implement, plus is the de-facto industrial standard for task scheduling and FreeRTOS native scheduling policy. The resource sharing policy of FreeRTOS to access local shared resources has been improved, and the support for inter-subsystem resource sharing to access global shared resources has been implemented in the HSF implementation. This entails: support for Stack Resource Policy (SRP) [19] for local resource sharing to avoid problems like priority inversions and deadlocks, and Hierarchical Stack Resource Policy (HSRP) [20] for global resource sharing with three different methods to handle overrun [21] to handle the budget expiration within the critical section. These three types of overrun mechanisms are overrun without payback (BO), with payback (PO), and enhanced overrun (EO). Implementation of BO is very simple, the server simply executes and overruns its budget, and no further action is required. For PO and EO we need to measure the overrun amount of time to pay back at the server's next activation. We have also provided legacy support for existing systems or components to be executed within our HSF implementation as a subsystem.

We have performed a detailed experimental evaluation [17] [18] on the implementation to test its temporal behavior and performance measures on an AVR-based 32-bit EVK1100 board [22]. The AVR32UC3A0512 micro-controller runs at the frequency of 12MHz and its tick interrupt handler at 1ms. We have tested the implementation for the correct behavior of idling and deferrable servers and of overrun mechanisms by plotting the traces of the execution of the system. We have also evaluated the system behavior during the overload situation and tested the temporal isolation among servers. We showed that when one server is overloaded and its tasks miss deadlines, it does not affect the behavior of other servers in the system, even if the priority of the overloaded server is highest; hence proves the temporal isolation and fault containment behavior of HSF.

IV. Applying Virtual Node Concept to ProCom, AUTOSAR, and AADL

In the component models we are currently studying the virtual node concept to be applied in the following way:

A. ProCom

In ProCom the Virtual Node is an integrated model concept. That means that the virtual nodes exist both on the modeling level and as executable entities as shown in Figure 1. The system is generated using two-level deployment process rather than a big synthesis step. A set

of ProSys subsystems are mapped to one virtual node which can then be integration-tested and validated for the correct temporal behavior. This virtual node preserves its temporal properties and hence becomes a reusable entity that is ready to deploy in numerous systems and stored for future reuse.

At the modeling level, each virtual node contains a set of integrated ProSys components plus the resources (CPU budget, memory) required for these ProSys components. At the executable level, virtual node contains the set of executable tasks, resources required to run those tasks and a real-time local scheduler to schedule these tasks. The local scheduler runs within a global scheduler in a HSF.

The final executables that can be downloaded and executed on the physical node consists of a set of virtual nodes and simple real-time scheduler linked together. The scheduler is the top level scheduler in the hierarchical scheduling framework, and is responsible for dispatching the servers of each virtual node according to their bandwidth reservation. As the real-time properties of the virtual node are preserved within the local scheduler, therefore when integrated with other virtual nodes on a physical node, the real-time properties of the whole integrated system will be guaranteed.

B. AUTOSAR

For AUTOSAR, we propose to map a number runnables to a virtual node. Thus, an AUTOSAR component can be deployed to a set of virtual nodes; the natural choice would be to use one virtual node per physical node that the component will be distributed over. Using this approach the component can be developed and its timing behavior tested without accounting for interference from other AUTOSAR components deployed at the same physical nodes.

However, since the AUTOSAR component-model and methodology does not recognize the virtual node as an entity of its own, reuse in different organizations or different software architectures may be difficult. However, the virtual node still provides strong encapsulation of the runnables and thus makes the functionality robust against future changes in both the runnables and in other components running in other virtual nodes.

C. AADL

For AADL, we propose to map the generated code from AADL models along with user written code to the virtual node. Hence instead of synthesizing whole system in a single big-bang step, the synthesis will be performed in smaller steps. The synthesis will be done at the two levels:

1) First the individual runnables will be created in isolation and timing analysis will be performed on them.

2) Then some middleware (e.g., PolyORB, PolyORB-HI) can be used for their intra-communications and to generate a whole system.

Currently a similar concept of two level code generation has been used for ARINC653 systems [15] using AADL. It is supported by the tool POK [14] that uses Ocarina tool for AADL models development and Cheddar tool for scheduling analysis. POK supports partitioning of the CPU and hierarchical scheduling for the underlying ARINC653 systems by using virtual processor. This approach is not generic in embedded real-time systems since ARINC653 is an avionics standard, therefore, the use of virtual processor is restricted to the avionics only.

V. CONCLUSIONS AND FUTURE WORK

We have described our technique of two-level deployment process to allow predictable integration of software components with temporal requirements. The technique is based on the concept of virtual nodes which use hierarchical scheduling to achieve temporal isolation and predictable execution of components allocated to the virtual nodes. The virtual node will become a real-time executable reusable entity. We have described how this technique can be used for three different component models: ProCom, AUTOSAR and AADL.

Future work is to do the code synthesis for generating and configuring virtual nodes from ProSys subsystems in ProCom component model. It includes the integration of our HSF implementation within the virtual node. Once these implementation efforts are complete will have all the links in a complete development chain for model driven engineering of component based system in the ProCom component technology:

- Using the ProCom Integrated Development Environment (PRIDE) components can be developed, assembled and deployed to virtual nodes.
- Using scheduling analysis of hierarchically scheduled systems [21] we can determine schedulability of both individual virtual nodes and the final composition of multiple virtual nodes on a single physical node.
- And, with our implemented code synthesis and runtime platform we can generate and execute the components and their applications in a predictable way.

The next step will then be to validate the generality of the virtual-node concept by applying it to AUTOSAR and AADL technologies.

REFERENCES

[1] I. Crnkovic and M. Larsson. “Building reliable component-based software systems”. *Artech House, Inc*., 2002.

[2] L. Sha, T. Abdelzaher, K.-E. Årzén, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A. K. Mok, “Real time scheduling theory: A historical perspective,” *Real-Time Systems*, vol. 28, no. 2/3, pp. 101–155, 2004.

[3] J. Stankovic, M. Spuri, M. D. Natale, and G. Buttazzo, "Implications of classical scheduling results for real-time systems," *IEEE Computer*, pp. 16–25, June 1995.

[4] J. Carlson, J. Feljan, J. Mäkki-Turja, and M. Sjödin, "Deployment modelling and synthesis in a component model for distributed embedded systems", In *36th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, September 2010.

[5] R. Inam, J. Carlson, J. Mäkki-Turja, and M. Sjödin, "Using temporal isolation to achieve predictable integration of real-time components", *Proceedings of Wip Euromicro Conference on Real-Time Systems (ECRTS10)*, July 2010.

[6] Z. Deng and J. W.-S. Liu, "Scheduling real-time applications in an open environment," *in Proceedings of IEEE Real-Time Systems Symposium*, December 1997.

[7] T. Bureš, J. Carlson, I. Crnkovic, S. Sentilles, and A. Vulgarakis, "ProCom – the Progress component model reference manual, version 1.0," *Mälardalen University, Technical Report MDH-MRTC-230/2008-1-SE*, June 2008.

[8] T. Bureš, J. Carlson, S. Sentilles, and A. Vulgarakis, "A component model family for vehicular embedded systems," *in The 3rd International Conference on Software Engineering Advances*. IEEE, October 2008.

[9] "Autosar project-page," www.autosar.org.

[10] SAE International, "AADL specification," http://www.sae.org/technical/standards/AS5506/1.

[11] AUTOSAR Partnership, "Specification of RTE V2.0.1 R3.0 Rev 0001 ," 2008, http://www.autosar.org/.

[12] J. Hugues, B. Zalila, L. Pautet, and F. Kordon, "From the prototype to the final embedded system using the ocarina AADL tool suite," *ACM Trans. Embedded Computer Systems*, vol. 7, no. 4, pp. 1–25, 2008.

[13] F. Singhoff, J. Legrand, L. Nana, and L. Marc, "Cheddar: a flexible real time scheduling framework," *Ada Lett.*, vol. XXIV, no. 4, pp. 1–8, 2004.

[14] J. Delange, L. Pautet, A. Plantec, M. Kerboeuf, F. Singhoff, and F. Kordon, "Validate, simulate, and implement arinc653 systems using the aadl," *Ada Lett.*, vol. 29, no. 3, pp. 31–44, 2009.

[15] Airlines Electronic Engineering, "Avionics Application Software Standard Interface". *TR, Aeronautical Radio, INC*, 1997.

[16] "FreeRTOS web-site," http://www.freertos.org/.

[17] Rafia Inam, Jukka Mäki-Turja, Mikael Sjödin, S. M. H. Ashjaei, Sara Afshar. "Support for hierarchical scheduling in FreeRTOS". *In Proc. of the IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2011)*, September 2011.

[18] Rafia Inam, Jukka Mäki-Turja, Mikael Sjödin, Moris Behnam. "Hard real-time support for hierarchical scheduling in FreeRTOS". *In Proc. of the 7th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT 11)*, July 2011.

[19] T. Baker. "Stack-based scheduling of real-time processes". *Journal of Real-Time Systems*, 3(1):67–99, 1991.

[20] R. I. Davis and A. Burns. "Resource sharing in hierarchical fixed priority pre-emptive systems". *In IEEE Real-Time Systems Symposium* (RTSS'06).

[21] M. Behnam, T. Nolte, M. Sjödin, and I. Shin, "Overrun methods and resource holding times for hierarchical scheduling of semi-independent real-time systems," *IEEE Transactions on Industrial Informatics*, vol. 6, no. 1, Feb 2010.

[22] "ATMEL EVK1100 product page," http://www.atmel.com/dyn/Products/.

Rafia Inam (S'09) is a Ph.D. student at Mälardalen Real-Time Research Centre, Mälardalen University, Västerås, Sweden. Her research interests include timing behavior and reusability of realtime embedded systems, and realtime hierarchical scheduling.

She received her M.Sc. degrees in computer science from the Quaid-i-Azam University, Islamabad, Pakistan, and her M.S. degree in networks and distributed Systems from Chalmers University of Technology (CTH), Göteborg, Sweden, in 1997, and 2010, respectively.

Rafia is awarded scholarship by IEEE Industrial Electronic Society for the paper "Support for Hierarchical Scheduling in FreeRTOS" in proceedings of the 16th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA' 11), IEEE Industrial Electronics Society, Toulouse, France, September, 2011.

Jukka Mäki-Turja is a software-research specialist at Arcticus Systems and senior lecturer and researcher at Mälardalen Real-Time Research Centre, Mälardalen University, Västerås, Sweden. His research interest lies in design and analysis of predictable real-time systems. Jukka received his PhD in computer science from Mälardalen University in 2005 with response time analysis for tasks with offsets as focus and has since pursued the research in predictable component-based software design for embedded systems.

Jan Carlson is a senior lecturer at the School of Innovation, Design and Engineering, Mälardalen University, Sweden. He received his M.Sc. degree in Computer Science from Linköping University in 2000, and his doctoral degree from Mälardalen University in 2007.

His research interests include component models for embedded systems, event pattern detection, formal methods and logic programming.

Mikael Sjödin is a professor of real-time systems at Mälardalen Real-Time Research Centre, Mälardalen University, Västerås, Sweden. Mikael is focusing his research on new methods to construct software for embedded control systems in the vehicular and telecom industry. The current research goal is to find methods that will make software development cheaper, faster and yield software with higher quality.

Concurrently, Mikael is also been pursuing research in analysis of real-time systems, where the goal is to find theoretical models for real-time systems that will allow their timing behavior and memory consumption to be calculated.

Mikael received his PhD in computer systems 2000 from Uppsala University (Sweden). Since then he has been working in both academia and in industry with embedded systems, real-time systems, and embedded communications. Previous affiliations include Newline Information, Melody Interactive Solutions and CC Systems. In 2006 he joined the MRTC faculty as a full professor with specialty in real-time systems and vehicular software-systems.