Using Temporal Isolation to Achieve Predictable Integration of Real-Time Components

Rafia Inam, Jukka Mäki-Turja, Jan Carlsson, Mikael Sjödin
Mälardalen Real-Time Research Centre
Västerås, Sweden
Email: rafia.inam@mdh.se

***Abstract*—We present the concept of a virtual node as means to achieve predictable integration of software components with real-time requirements. The virtual node is based on the technology using the hierarchical scheduling framework to achieve temporal isolation between components (or sets of components). The temporal isolation means that all components deployed to a virtual node will retain their temporal properties when integrated with other components on a physical node. We present how the concept of virtual nodes is applicable to three different component technologies: ProCom, Autosar and AADL.**

***Keywords*-real-time systems; component integration**

I. Introduction

Integration of software components with real-time requirements is a tricky business. 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.

Using temporal models of component behavior and requirements some of these problems may be mitigated by using scheduling analysis [1], [2]. However, contemporary technique only allow very simple models; typically simple timing attributes such as *period* and *deadline* are used. Thus, often components 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 and the reuse 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.

To remedy this situation we propose the concept of a *virtual node*, which is an execution-platform concept that preserves temporal properties of the software executed in the virtual node [3]. The virtual node is intended for coarse-grained components for single node deployment and with potential internal multitasking. In this paper we describe the virtual node concept and how it can be applied in the run-time infrastructure in three different component technologies: ProCom [4], Autosar [5], and AADL [6].

Paper Outline: Section II describes the component technologies we study in this paper. In section III we describe the virtual node execution-mechanism, and in section IV we conclude the paper with a description of ongoing work.

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 technologies that use CBSE (e.g., AUTOSAR) or MBE (e.g., AADL) or both CBSE and MBE (e.g., ProCom). We present these technologies from the perspective of deployment of the components on a physical platform and the generation of final executables of the system.

A. ProCom

ProCom model uses both CBSE and MBE for the development of its components hence it not only exploits the benefits of both approaches (i.e. encapsulation and reusability of CBSE, and early analysis and automated code generation of MBE) but also achieves additional benefits of combining both approaches (like flexible reuse, support for mixed maturity, reuse and efficiency tradeoff) [3].

Modeling in ProCom is supported by four distinct but related formalisms as shown in Figure 1. ProSave and ProSys are used to model the functional architecture of the system under construction, addressing the different concerns that exist on different levels of granularity in distributed embedded systems. In short, ProSys models a system as a collection of active, concurrent subsystems communicating via asynchronous message passing, and ProSave addresses

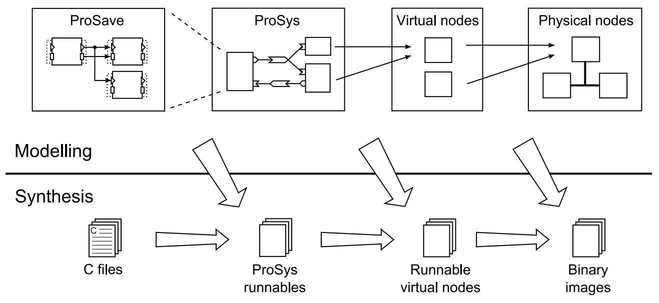

Figure 1. An overview of the deployment modelling formalisms and synthesis artefacts.

the detailed structure of an individual subsystem, by specifying how data and control are transferred between passive components. Both ProSys and ProSave allow composite components, i.e., components that are internally realized by a collection of interconnected subcomponents. For details on ProSave and ProSys, including the motivation for separating the two, see [4], [7].

Deployment is performed in two steps, introducing an intermediate level where ProSys subsystems are allocated to *virtual nodes* that, in turn, are allocated to physical nodes. This approach allows more detailed analysis to be performed without full knowledge of other parts that will share the same physical node in the final system. A realisation based on hierarchical scheduling and resource budgets ensures that a virtual node can be analysed independently from the rest of the system, also with respect to timing. Section III describes this further.

B. Autosar

AUTomotive Open System ARchitecture (AUTOSAR) [5] is an open standard for automotive electronics architecture whose latest version was released in 2010. 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[1] 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. 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.
- 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 [8].

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 [8]. After mapping, RTE generator configures each ECU. In the last, the OS tasks bodies are constructed by RTE generator. The main disadvantage

[1]An ECU, Electronic Control Unit, is a node in an automotive network.

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 SEA Standard AS-5506 [6] 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) [9]. Properties and new functional aspects can be attached to the elements (e.g., components, connections) using the properties defined in the SEA standard, and communication among components is performed using component interfaces i.e., ports. Ocarina [10] 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 [11]), and then automatic code generation from these models and their deployment. Automatic code generation is done using the Ocarina compiler [9] that comprises of two traditional parts: the frontend(lexical, syntactic, and sematic analyses and instantiation) and the backend (expansion and conversion of instance tree, and code generation in C or Ada).

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 self-contained 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).

III. Virtual Nodes

The concept of a virtual node is based on a two-level hierarchical scheduling framework [12]. A HSF is introduced to support CPU time sharing among servers under different scheduling disciplines. In HSF, a system $\mathcal{S}$ consists of one or more servers $S_s \in \mathcal{S}$. The HSF can be generally viewed as a two-level tree, where each leaf-node represents a server with its own local scheduler for scheduling internal tasks, and CPU time is allocated from the parent node, as illustrated in

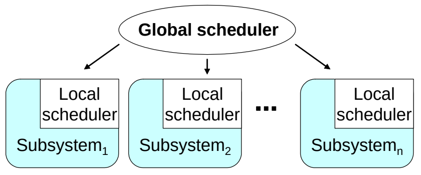

Figure 2. Two-level hierarchical scheduling framework.

Figure 2. 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.

A virtual node represents the functionality of component (or a set of integrated components) combined with allocated execution resources[2].

A virtual node includes the executable representation of the components (i.e. a set of tasks), a resource allocation, and a real-time scheduler to be executed within a server in the HSF. The server will execute with a guaranteed temporal behavior, using its allocated CPU bandwidth, regardless of any other execution on the physical node. Thus, once a server has been configured for the virtual node, its real-time properties will be preserved when the virtual node is integrated with other virtual nodes on a physical node, or when a virtual node is reused in another context. We do not prescribe what type of scheduler a virtual node should use; it can be any type of scheduler that provides real-time guarantees.

To get the final compiled binary that can be downloaded and executed on the target, a set of virtual nodes and simple real-time scheduler is 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.

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

ProCom

In ProCom the Virtual Node is an integrated model concept. That means that the virtual nodes exist both on the modelling level and as executable entities. A set of ProSys subsystems are mapped to one virtual node which can then be integration-tested and validated for correct temporal behaviour.

This virtual node then becomes a reusable entity that is ready to deploy in numerous systems and stored for future reuse.

Autosar

For Autosar, we propose to map a number runnables to a virtual node. An Autosar component can be deployed to a

[2]Currently, we focus mainly on the resources CPU-bandwidth and memory. However, in future work other resources, such as energy, could be added.

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 behaviour 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.

AADL

We propose to map the generated code from AADL models along with user written code to the virtual node. Hence instead of synthesizing the whole system in a single big step, the synthesis will be performed in smaller steps. It will be done at two levels. First the individual runnables will be created in isolation and timing analysis will be performed on them. Then some middleware (e.g., PolyORB, PolyORB-HI) could 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 [13] using AADL, supported by the tool suite POK [14] that uses Ocarina for AADL models and Cheddar for scheduling analysis. POK supports partitioning and HSF 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.

IV. Conclusions and Ongoing Work

We have described our technique 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 predictable execution of components allocated to the virtual nodes. We have described how this technique can be used for three different component models: ProCom, Autosar and AADL.To get minimal changes and better utilization of the system, it will be good to match scheduling techniques to the underlying system (e.g. using fixed-priority scheduling for AUTOSAR, TDMA for AADL). Ongoing work is to implement the hierarchical scheduling framework in FreeRTOS [15] for our target platform EVK1100 (an AVR32-based board) [16]. Also, code synthesis for generating and configuring virtual nodes from ProSys subsystems is ongoing. Once these implementation efforts are complete, we 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 [17] 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 be to validate the generality of the virtual-node concept by applying it to Autosar and AADL.

References

[1] 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.

[2] 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.

[3] J. Carlsson, J. Feljan, and M. Sjödin, "Deployment Modelling and Synthesis in a Component Model for Distributed Embedded Systems," To appear in SEAA, September 2010.

[4] T. Bureš, J. Carlson, I. Crnković, 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.

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

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

[7] 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.

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

[9] G. Lasnier, B. Zalila, L. Pautet, and J. Hugues, *OCARINA : An Environment for AADL Models Analysis and Automatic Code Generation for High Integrity Applications*. Springer Berlin Heidelberg, 2009, iSBN 978-3-642-01923-4.

[10] 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. Embed. Comput. Syst.*, vol. 7, no. 4, pp. 1–25, 2008.

[11] 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.

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

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

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

[16] "ATMEL EVK1100 product page," http://www.atmel.com/dyn/Products/tools_card.asp?tool_id=4114.

[17] 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, February 2010.