Implementing and Evaluating Communication-Strategies in the ProCom Component Technology

Rafia Inam, Mikael Sjödin
Mälardalen Real-Time Research Centre,
Västerås, Sweden
Email: {rafia.inam, mikael.sjodin}@mdh.se

***Abstract*—This paper presents two strategies to support communication between real-time executable Runnable Virtual Node (RVN) components in the ProCom component technology. We describe the currently implemented server-based communication strategy which uses a dedicated server for communication. We compare the server-based technique with a direct (RVN-to-RVN) communication strategy. The paper also describes how these strategies could be evaluated for real-time performance, and the real-time analysis technologies needed to perform such an evaluation.**

***Index Terms*—real-time software components; hierarchical design**

I. Introduction

ProCom is a component technology for development of hard real-time embedded control-systems [1]. We have previously presented the ProCom-concept of a *runnable virtual node* that allows a two-step deployment process [2]. The first-step is an initial virtual deployment to a virtual node and in the last-phase step virtual nodes are deployed on physical nodes. In this paper we focus on the communication strategies among virtual nodes. We explain how ProCom realizes the communication today, and outline how an alternative strategy for communication could be designed. We also propose an evaluation metric to allow comparison of the real-time performance of two different communication strategies.

In ProCom the realization of a piece of functionality can follow a flow through many software components. Data may originate at one component (e.g. a sensor) and passes through various other computational components, before terminating at the final component (e.g. an actuator). Hence, the data follows a chain of components $(C1, C2, \ldots, Cn)$, each potentially having its own periodicity and timing properties. For an embedded system with real-time constraints, the end-to-end timing behavior is not only dependent on the timing properties of its constituent components but also on the message-chains among those components. ProCom provides a hierarchy of component types, where the top-level component-type is called a *virtual node*. The virtual node (typically) contains a set of other ProCom components and can be synthesized to become an independent executing unit called a *runnable virtual node (RVN)* [3].

In ProCom today, communication *inside* an RVN is implemented with shared buffers and semaphore-locks. In this paper we focus on the communication *between* RVNs, which is implemented using a *communication server*. The communication server provides temporal isolation between RVNs and buffers all communication, making the execution of each RVN very predictable. However, the buffering incurs delays which could adversely affect the fulfillment of real-time requirements. Hence, an interesting approach would be to use a more direct communication strategy, where RVNs communicate with each other directly without an intermediate node. In this paper we describe how a comparison study of the server-based and direct approaches could be performed.

The main contributions of this paper are:

- We present the current server-based implementation for predictable communication between RVNs in ProCom.
- We present an alternative communication strategy using a direct communication method.
- We propose how these two strategies could be evaluated and compared for their real-time performance.

Outline: In Section II, we describe the RVN, how it embeds hierarchical scheduling using the two-level deployment process, and how it is used within the ProCom technology. Section III presents the inter-RVN communication strategies and their performance evaluation criteria are provided in Section IV. Finally, Section V concludes the article and describes future work.

II. The Runnable Virtual Node (RVN)

A runnable virtual node is an execution platform concept that preserves functional as well as temporal properties of the software executed within it [3]. The idea is to encapsulate the real-time properties into model-driven reusable component-based systems to achieve predictable integrations and reusability of those components along with maintainability, testability, and extendibility.

A. An RVN-server

An RVN is implemented as a server within a two-level *Hierarchical Scheduling Framework (HFS)* (an RVN-server), and includes a set of tasks, a resource allocation ($\langle budgetQ, periodP \rangle$ of server), and a real-time scheduler as shown in Figure 1. The scheduler is local-level, and schedules the task set according to allocated resources using a scheduling policy (currently fixed-priority preemptive scheduling FPPS). The final executables that can be downloaded and executed on the physical node consists of a set of RVNs and a top-level real-time scheduler linked together. The top-level scheduler

in the HSF is responsible for dispatching the RVN-servers according to their bandwidth reservations. Thus, once a server has been configured for the RVN, its non-functional (timing) properties are preserved along with its functional properties when the RVN is integrated with other RVNs on a physical node, or when it is reused in another context [3]. Further it reduces the efforts related to testing, and validation.

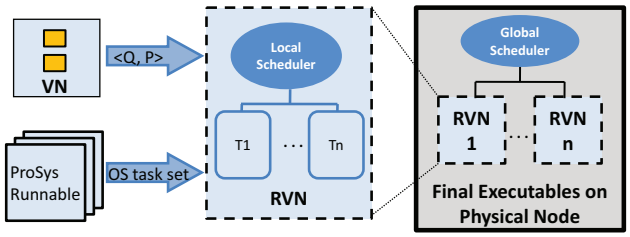

Fig. 1. An RVN encapsulates and preserves functional and non-functional (timing) properties.

In our implementation, the RVN is executed by a server in the HSF running on-top of FreeRTOS [4]. Using HSF, the functionality of different servers can be isolated from each other for, e.g., fault containment, compositional verification, validation and certification, and unit testing [5]. The official release of FreeRTOS only supports a single level fixed-priority scheduling. We have, however, previously presented an implementation of two-level HSF for FreeRTOS [6] with associated primitives for hard real-time sharing of resources both within and between servers [7]. The HSF implementation supports two kinds of servers, idling periodic [8] and deferrable servers [9]. The implementation uses FPPS for both global and local-level scheduling. For local resource-sharing (within a server) the Stack Resource Policy (SRP) [10] is used, and for global resource-sharing (between servers) the Hierarchical Stack Resource Policy (HSRP) [11] is implemented. The HSF supports CPU resource reservations by associating a tuple $\langle Q, P \rangle$ to each server, where P is the server period and the server budget Q ($0 < Q \leq P$) is the allocated portion of CPU resources every P. Given Q, P, and information on resource holding times, the schedulability of a server and/or a whole system can be calculated with the methods presented in [7].

B. The RVN Concept in The ProCom Component Model

The ProCom component technology targets control-intensive embedded systems like software used in trains, airplanes, cars, industrial robots, etc. The ProCom component model [12] is specifically developed to address the reuse of design artefacts (e.g., extra-functional properties, analysis results, and behavioral models) as well as predictable integration and reuse of the executable components [3]. The PROGRESS Integrated Development Environment (PRIDE) tool [13] supports modeling and automatic-synthesis of components at different levels [1].

The RVN concept uses a two-level deployment process rather than a single big deployment: i.e. deploying functional entities to RVNs in *a first-step* (during which, e.g., the timing properties of RVNs are validated), and then, deploying RVNs to the physical nodes in *a second-step* (integrating RVNs along with their preserved timing properties) [3]. The two-level process gives development benefits with respect to composability, system integration, testing, validation, certification, and reuse.

The RVN is an integrated concept in ProCom. From the modeling perspective, a virtual node (VN) is equivalent to a set of ProSys components with an added resource reservation. In the executable form, the RVN is constructed by mapping the set of tasks (synthesized from ProSys-runnable components) to a server and assigning scheduling parameters (assignment of task-priorities) during the first-step of deployment. Internal validity of the timing-constraints of the RVN can then be assessed using, e.g., simulation, testing or a *local scheduling-analysis* provided in [7]. In this manner, after configuration the RVN-server preserves its timing properties within it. It also adds an implementation of message channels used to send messages among virtual nodes. The final binaries are generated for a hardware node during the second-step of deployment by connecting different RVNs together with a global scheduler, assigning server-priorities, and using a *middleware API* for *inter-RVN communications*.

III. The Inter-RVN Communication

The RVN provides main benefits of predictable integration and increased reuse of executable real-time components. It leads up to the necessity to make the communication among RVNs not only fast and predictable but the communication should also support the reuse of the executable components. To achieve these benefits, the inter-component or inter-RVN communication is implemented independently from the underlying platform as a *middleware API* by moving the information about system and communication outside the component code. Later the middleware interface functions are integrated into the layered ProCom model.

A. Middleware API

The inter-RVN communication is a combination of data and trigger ports and is based on messages. The middleware API is implemented a-synchronously via message passing and the cyclic shared buffers, where channels are used to distribute the messages to other RVNs using a defined set of connections. The communication is independent of the underlying operating system (HSF implementation in our case). It includes the support for transparent communication within RVNs mapped on the same hardware node, called *local-RVN communication*, and among RVNs mapped on different hardware nodes through a communication media or channel (e.g. CAN bus), called *distributed-RVN communication*. Both types of communication can be synthesized before generating the final binaries for the target platform.

The middleware API is well-integrated with the layered deployment process of the ProCom model. The main communication code (including data structures) is initialized and two periodic tasks *sender* and *message-port updater* are created at every physical node during the first-step of deployment. These

tasks are responsible to send and receive messages among RVNs respectively.

The inter-RVN communication could be realized in two different ways: either integrating the middleware API directly into the communicating RVNs, called *Direct Communication*, or using a server to embed the middleware tasks in it, called a *Server-based Communication*. Both strategies are explained here.

B. Server-based Communication Strategy

Since RVNs are implemented as servers within a two-level HSF, it makes sense to embed the middleware API within a server. A *Communication (also called a system) server* along with its timing properties is automatically generated for inter-RVN communications (if needed), at the second-step of deployment. Both communication tasks, sender and message-port updater, are automatically assigned to the server as shown in Figure 2. Additionally there can be a hardware-driver task in it if needed. The communication server has the highest priority among all the servers in the system, with a very small budget and its only functionality is to copy the messages from the sender port of one component to the receiver port of another component, by executing the middleware API tasks within it.

This method provides benefits of (1) increased reuse of RVN by keeping the communication separated from the RVN code and (2) predictability by executing the communication API in a server within the HSF implementation. However, it has some overhead of server execution. Moreover, it also increases (3) maintainability and flexibility to change the communication code without affecting the timing properties of RVNS.

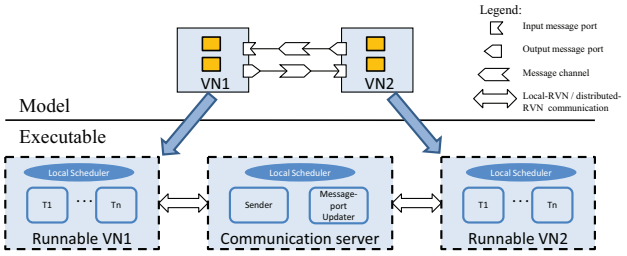

Fig. 2. Server-based inter-RVN communication at modeling and executable levels.

The final executables are generated by resolving the local-RVN communication by mapping it to middleware API, and synthesizing the distributed-RVN communication among hardware nodes (if needed). All synthesis is done by generating C-code, so the final step is compiling the generated code, and linking all code with the operating system and middleware binaries.

The local-RVN communication is provided by the middleware where the output and input message ports write and read the data respectively. One-step shared cyclic buffers that can be accessed by multiple tasks, are added to these ports for reliable message delivery and efficient memory usage. The only additional requirement is a communication channel to be generated for the distributed-RVN communication. (At this stage the PRIDE tool provides the distributed-RVN communication in the form of a TCP/IP connection over Ethernet. A real-time distributed-RVN communication is not automatically generated by the tool and has to be provided manually).

Since inter-RVN communication is implemented as a server within a two-level HSF, simple semaphores cannot be used to protect shared buffers. Thus some synchronization protocols for hierarchical schedulers are required to access the buffers in a safe manner. SRP and HSRP protocols are implemented in the HSF implementation [7]. The shared buffers among the tasks of same RVN and among tasks of different RVNs are arbitrated using SRP and HSRP respectively. HSF leverages the communication among components with the advantages of short and predictable global blocking, and predictable and well-defined communication.

C. Direct Communication Strategy

Inter-RVN communication can also be done directly among RVNs (i.e. RVN-to-RVN) without using the communication server. Shared message queues which are accessed via SRP and HSRP APIs [7] can be used for this purpose. The RVN can encapsulate the middleware API within it to send and receive the data and/or messages (see Figure 3) at the first-step of deployment. It requires a separate configuration of RVN for each communication. The final binaries are generated from RVNs along with the code for the communication mechanism used (local- or distributed-RVN) at the final-step of deployment.

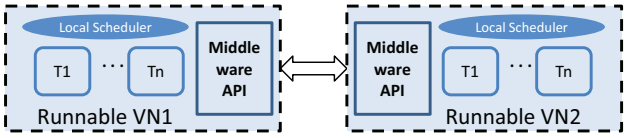

Fig. 3. Direct inter-RVN communication at executable level.

The direct communication is fast as compared to the server-based communication. However, it reduces the reusability of executable RVNs since RVNs need to be configured for each system separately, depending on that system's communication requirements. Further, any change in the middleware communication code will not only require a code-change in all RVNs but will also affect the timing properties of all RVNs involved in that communication.

IV. Performance Evaluation

We plan to evaluate and compare the costs of the server-based strategy with that of the direct communication strategy. To perform the evaluation, we plan to implement a simulator, generating random subsystems and tasks in subsystems, both types of communication, and measuring the costs of server-based and direct communication methods among RVNs. For evaluation, the end-to-end latencies of inter-RVN communication should be measured for both strategies and then compared.

A. Evaluation Criteria

We propose the use of a metric *Worst Case Reaction Time (WCRT)* as the total time taken by a message-chain from message-originator until the message reaches its message-terminator component. For hard real-time systems, of course, the worst case is a more interesting metric than any statistical metric. Also, metrics like worst-case response-time for tasks do not help us to evaluate end-to-end delays.

B. Evaluating the Server-based Strategy

When using the communication server, there is no direct control-flow between components. Instead, data flows through buffers inside the communication server.

The originator, the communication server, and the terminator are running asynchronous, each with its own period. This makes the traditional scheduling analysis technique very difficult to use to obtain the WCRT. Instead, data-path analysis, like the technique proposed by Feiertag *et al.* [14], could be used. A prerequisite to use their technique is that the worst-case response-times of each task in the RVNs are known. The response-times can be calculated using the analysis techniques proposed in [7].

C. Evaluating the Direct Communication Strategy

When using direct communication, control-flow is direct between the RVNs and we could use existing scheduling-analysis techniques to calculate WCRTs. However, different analysis techniques would likely give different results – and there are no analysis techniques that provide a perfect fit for our strategy, rather a complicated task-model.

Real-time calculus could be used to summarize delays from the different components [15]. This could take into account the potential difference in periodicity between an RVN and the components executing inside the RVN. However, it would be difficult to model the explicit control-flow between RVNs in real-time calculus. To improve the precision it would be desirable to use more exact analysis techniques based on the *holistic scheduling analysis* by Tindell [16] with explicit modeling of precedence constraints [17]. These techniques cannot be used right out of the box though, instead they would have to be incorporated into the existing scheduling analysis applicable to the RVNs [7].

V. Conclusions and Future Work

We have presented two different strategies to support communication among real-time executable components RVNs, and plan to evaluate and compare them. We have implemented the server-based technique as a means to encapsulate the middleware API to provide a predictable communication mechanism to preserve the RVN's timing properties at run-time and for better RVN reuse.

For future work, we plan to compare the cost of the server-based method with the cost of the direct communication method (among components/subsystems). We plan to present the simulated results of our evaluation for both methods and their comparison. The direct communication can be evaluated by incorporating the scheduling analysis of [16], [17] into the analysis techniques provided in [7], while the server-based method will be evaluated using the techniques provided in [14].

At the moment the message passing mechanism is implemented using periodic tasks for receiving and sending messages. This may cause delays in the system and inefficient use due to unnecessary idle time. It would be interesting to see how much performance can be gained by making the message passing mechanism event-triggered. For example, we expect a better response time, less context switching.

References

[1] Etienne Borde and Jan Carlson. Towards verified synthesis of procom, a component model for real-time embedded systems. In *14th International ACM SIGSOFT Symposium on Component Based Software Engineering (CBSE)*. ACM, June 2011.

[2] R. Inam, J. Mäki-Turja, J. Carlson, and M. Sjödin. Virtual Node – To Achieve Temporal Isolation and Predictable Integration of Real-Time Components. *International Journal on Computing (JoC)*, 1(4), 2012.

[3] Rafia Inam. *Towards a Predictable Component-Based Run-Time System*. Number 145. Licentiate thesis, January 2012.

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

[5] Z. Deng and J.W.-S. Liu. Scheduling real-time applications in an open environment. In *IEEE Real-Time Systems Symposium(RTSS'97)*, pages 308–319, 1997.

[6] Rafia Inam, Jukka Mäki-Turja, Mikael Sjödin, S. M. H. Ashjaei, and Sara Afshar. Support for Hierarchical Scheduling in FreeRTOS. In *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA' 11)*.

[7] Rafia Inam, Jukka Mäki-Turja, Mikael Sjödin, and Moris Behnam. Hard Real-time Support for Hierarchical Scheduling in FreeRTOS. In *7th Annual Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT' 11)*, pages 51–60, Porto, Portugal, 2011.

[8] L. Sha, J.P. Lehoczky, and R. Rajkumar. Solutions for some Practical problems in Prioritised Preemptive Scheduling. In *Proc. IEEE Real-Time Systems Symposium (RTSS)*, pages 181–191, 1986.

[9] J.K. Strosnider, J.P. Lehoczky, and L. Sha. The deferrable server algorithm for Enhanced Aperiodic Responsiveness in Hard Real-time Environments. *IEEE Transactions on Computers*, 44(1), 1995.

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

[11] R. I. Davis and A. Burns. Resource sharing in hierarchical fixed priority pre-emptive systems. In *IEEE Real-Time Systems Symposium (RTSS'06)*, pages 389–398, 2006.

[12] Séverine Sentilles, Aneta Vulgarakis, Tomáš Bureš, Jan Carlson, and Ivica Crnković. A Component Model for Control-Intensive Distributed Embedded Systems. In *11th International Symposium on Component Based Software Engineering*, pages 310–317, October 2008.

[13] PRIDE Team. PRIDE: the PROGRESS Integrated Development Environment, 2010. "http://www.idt.mdh.se/pride/?id=documentation".

[14] Nico Feiertag, Kai Richter, Johan Nordlander, and Jan 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 (CRTS'08)*, November 2008.

[15] Lothar Thiele, Samarjit Chakraborty, and Martin Naedele. Real-time calculus for scheduling hard real-time systems. In *IEEE International Symposium on Circuits and Systems*, pages 101–104 vol. 4, Geneva, May 2000.

[16] K. Tindell and J. Clark. Holistic Schedulability Analysis For Distributed Hard Real-Time Systems. Technical Report YCS197, Real-Time Systems Research Group, Department of Computer Science, University of York, November 1994. URL ftp://ftp.cs.york.ac.uk/pub/realtime/papers/-YCS197.ps.Z.

[17] J.C. Palencia Gutierrez and M. Gonzalez Harbour. Exploiting Precedence Relations in the Schedulability Analysis of Distributed Real-Time Systems. In *Proc. 20^{th} IEEE Real-Time Systems Symposium (RTSS)*, December 1999.