Performance Preservation using Servers for Predictable Execution and Integration

Rafia Inam
Mälardalen Real-Time Research Centre
Mälardalen University, Västerås, Sweden
Email: rafia.inam@mdh.se

***Abstract*—In real-time embedded systems the components and components integration must satisfy both functional correctness and extra-functional correctness, such as satisfying timing properties. Deploying multiple real-time components on a physical node poses timing problems in components's integration. These timing problems during integration further effect predictability and reusability of real-time components.**

We propose a novel concept of *runnable virtual node (RVN)* whose interaction with the environment is bounded both by a functional and a temporal interface, and the validity of its internal temporal behaviour is preserved when integrated with other components or when reused in a new environment. Our realization of RVN exploits the latest techniques for hierarchical scheduling framework to achieve temporal isolation, and the principles from component-based software-engineering to achieve functional isolation. Proof-of-concept case studies executed on an AVR based 32- bit micro-controller demonstrates the preserving of real-time properties within components for predictable integration and reusability in a new environment without altering its temporal behaviour in both hierarchical scheduling and RVN contexts.

We also take a step ahead towards expanding the performance preserving servers' concept for multicore platform on which the scheduling of real-time tasks is inherently unpredictable due to the contention for shared physical memory and caches. It results in proposing and implementation of a novel type of server, called *Multi-Resource Server (MRS)* which controls the access to both CPU and memory bandwidth resources such that the execution of real-time tasks become predictable. The MRS provides temporal isolation both between tasks running on the same core, as well as, between tasks running on different cores. Further, we provide the schedulability analysis for MRS to provide predictable performance when composing multiple components on a shared multi-core platform.

***Index Terms*—Component integration, predictable execution, server-based scheduling, memory throttling, CPU resource, memory resource, memory bandwidth.**

I. Introduction

In embedded real-time systems, a continuous increasing trend in size and complexity of embedded software has observed during the last decades. To battle this trend, modern software-development technologies are being adopted by the real-time industry. The reuse of legacy code is an answer to main challenges of development cost, time to market, and to the increasing complexity of these applications. The trend of software use and reuse is observed in embedded software for automotive, consumer electronics and avionics applications areas. Moreover, many industrial systems are developed in an evolutionary fashion, reusing components from previous versions or from related products [1]. For example, modern cars contain nearly 100 million lines of code running on around 70 to 100 embedded processors [2]. The new Boeing 787 "Dreamliner" is a recent example with a significant proportion of reused modules from another Boeing airplane [3]. It means that components are reused and re-integrated in new environments. For systems with real-time requirements, this integration and reuse in a new environment poses new challenges.

This work is supervised by Prof. Mikael Sjödin and Dr. Moris Behnam.

A. Research Problem and Challenges

Composition/integration of real-time applications (also referred to as components in [4]) can be explained as the mechanical task of wiring components together [1]. For real-time embedded systems the components and components integration must satisfy (1) functional correctness and (2) extra-functional i.e. temporal correctness such as satisfying timing properties. Temporal behavior of 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. Further the reuse of a component is restricted because it is very difficult to know beforehand if the component will pass a schedulability test in a new system. For real-time embedded systems, methodologies and techniques are required to provide temporal isolation so that the timing properties could be guaranteed.

One promising technique for integrating complex real-time components on a single processor to overcome these deficiencies is a Hierarchical Scheduling Framework (HSF) [5], [4] in which the components are executed as servers with the specified budgets and periods. It supports CPU-time sharing among components or applications, hence isolating components' functionality from each other for, e.g., temporal fault containment, compositional verification, and unit testing. It

also supplies an efficient mechanism (i) to provide predictable integration of components by rendering temporal partitioning among components, (ii) to support independent development and analysis of real-time components, and (iii) to provide the analysis of integrated components at the system level [4]. HSF has been proposed to develop complex real-time systems by enabling temporal isolation and predictable integration of software-functions [6].

However, integrating HSF within a component technology for embedded real-time systems raises challenges of preserving the timing properties within components to apply these properties during components' integration and deployment on unicore hardware platform, and to guarantee the temporal properties of real-time components for their predictable integration and reusability. Another challenge during component integration is to provide communication among various components of a target software system. This communication should also be predictable in case of real-time components and do not affect the schedulability of tasks.

Using multicore platforms for real-time applications presents more challenges due to shared resources like shared last-level caches and memory-bus. One such challenge is to achieve and maintain predictable execution of concurrent tasks that compete for both CPU- and memory-bandwidth resources and share caches. On unicore platforms, the server-based scheduling approach successfully bounds the interference between the applications running [7], [8], [4]. However, this approach is limited in provisioning of the CPU-resource only and it does not take care about the activities that are located on different cores and can still interfere with other applications in an unpredictable manner hence imposing negative impact on system performance and real-time guarantees. A main source of such an unpredictable negative impact is the contention for shared physical memory. In commercially existing hardware, there are currently no mechanisms that allow a core to protect itself from negative impact if another core starts stealing its memory bandwidth or starts polluting the shared cache. For performance-critical real-time systems, overcoming this problem is paramount. Thus the memory-bandwidth should also be considered to guarantee predictable performance of real-time applications that are located in different cores for multicore platforms, especially when migrating or reusing software from a single-core to a multi-core architecture. Hence, there is a need to develop software technologies to track, and eventually police, the consumed memory-bandwidth in order to achieve predictable software execution on multi-core platform.

While the incorporation of memory-bandwidth resource into the server-based scheduling increases the predictable execution of real-time applications on different cores, the shared caches still hinders the predictability and is another challenge that we plan to target. The scheduling alone (i.e. controlling the allocation of resources over time) is not enough to achieve complete timing isolation. The cache pollution can have a tangible effect on timing properties of tasks executing in different servers and there is a strong need to investigate in either implementing some cache-partitioning techniques (like [9]) or bounding caches using some static analysis technique (like [10]).

Further, the schedulability analysis for the newly proposed server-based approach is another challenge that we target.

Please note that in this work we focus on the schedulability of tasks, i.e. meeting their deadlines, as the main timing property to achieve predictability. A component's timing behaviour is *predictable* during its integration and reuse, as long as the schedulability of tasks that have been validated during its development within a component is guaranteed when multiple components are integrated together.

B. Research Goal and Refined Research Challenges

The overall goal of the research is: *to provide methods and tools to achieve the predictable execution and integration of run-time components with real-time properties.*

The research goal can be refined and formulated into the following research challenges:

1) Achieving predictability during real-time components' integration and their reuse.
2) Developing runtime mechanisms to preserve the temporal properties within real-time components and to integrate legacy code in new environment.
3) Integrating HSF technique within the CBSE to improve today's embedded system development by reusing preserved temporal properties.
4) Providing analysis framework for components' integration.

II. Contributions

The aim of the research work is to preserve the timing properties of the real-time components to achieve predictable integrations and reusability of those components. To achieve this, new methods and techniques for embedded real-time systems are introduced where timing properties of the components are preserved, in order to make the integration of real-time components' predictable. Further the real-time properties of the components are maintained for reuse in real-time embedded systems. The main contributions of the thesis are as follows:

A. Implementation of HSF for unicore platform

This contribution addresses challenges 1 and 2. We have provided a two-level hierarchical scheduling support for FreeRTOS operating system with the consideration of minimal modifications in FreeRTOS kernel [11]. FreeRTOS [12] is a portable open source real-time scheduler and is selected for HSF implementation because of its main properties like small and scalable, support for more than twenty different hardware architectures, and easy to extend and maintain.

We extend FreeRTOS scheduler for idling periodic [13] and deferrable [14] servers using fixed-priority preemptive scheduling at both global and local levels. We performed a detailed experimental evaluation on the implementation to test its temporal behavior and overhead measures of the implementation on an AVR-based 32-bit EVK1100 board.

Further, We extend our scheduler to support resource sharing among arbitrary tasks which execute in arbitrary components [15]. We also provide support for legacy server along with wrappers for old operating system APIs to reuse the legacy code without the need of modifying the legacy code. We extended our initial implementation with these properties and evaluated on the same target platform using a legacy FreeRTOS application and a set of synthetic experiments [15].

B. Presentation and realization of RVN concept

To address the challenges of preserving the timing properties within components and to apply these properties during components' integration and reuse (addressing first three challenges), we have proposed and realized the concept of a *runnable virtual node (RVN)* [16], [17], [18]. The RVN is intended for coarse-grained components for single node deployment and with potential internal multitasking. We integrate our HSF implementation within a component technology for embedded real-time systems in order to provide guaranteed temporal properties of real-time components, and predictable integrations and reusability of those components. An RVN represents the functionality of software-component (or a set of integrated components) combined with allocated timing resources and a real-time scheduler to be executed as a server in the HSF. Thus the functional properties (functionality of components) are combined and preserved with their extra-functional properties (timing requirement) in the virtual nodes. In this way it encapsulates the behavior with respect to timing and resource usage and becomes a reusable executable component in addition to the design-time components.

Moreover, we have implemented a server-based communication strategy that supports predictable integration and reuse of the timing properties of RVNs by keeping the communication code in a separate server [19]. This strategy incorporates the maintainability and flexibility to change the communication code without affecting the timing properties of RVNs. We have evaluated the server-based strategy with a more direct communication strategy for efficiency and reusability properties of RVNs. Hence using RVNs and server-based inter-RVN communication, complex real-time systems can be developed as a set of well defined reusable components encapsulating functional and timing properties [20]. The work is based on the ProCom component-technology [21], however, we believe that our concept is applicable also to commercial component technologies like AADL, AUTOSAR [17].

C. Presentation and realization of Multi-Resource Server for multicore platform

In this contribution we address challenges 1 and 2, and target statically partitioned multi-core real-time systems. We present the Multi-Resource Server (MRS) technology that schedules the resources CPU- and memory-bandwidth, and shared cache in order to achieve a predictable execution of embedded real-time systems [22], [23]. A first step in achieving predictable execution is to accurately measure the amount of consumed memory-bandwidth for each application and secondly to incorporate the cache partitioning. Such measurements can be used to track down bottlenecks, provide better partitioning among cores, and ultimately be used to arbitrate and police accesses to the memory bus and shared caches.

MRS enables predictable execution of real-time applications on multi-core platforms through resource reservation approaches in the context of CPU-bandwidth reservation and memory-bandwidth reservation. The MRS provides temporal isolation both between tasks running on the same core, as well as, between tasks running on different cores. The latter could, without MRS, interfere with each other due to contention on a shared memory bus.

We have implemented the MRS as a user-space library for Linux running on multicore COTS hardware [24]. We have incorporated the CPU- and memory-resources and we intend to bound the cache access as well. We demonstrate how the MRS can be used to preserve the functionality of a legacy application when it is executed on a single core while another core executes tasks with adverse memory behavior. We demonstrate for a synthetic task-set how the MRS can be used to isolate tasks from each other to prevent adverse behavior of some tasks to negatively impact other tasks.

D. Proof-of-concept case studies

In this contribution we validate our solutions of 1st, 2nd, and 3rd challenges. Although the synthetic experiments confirm the correctness of the approaches, the example case studies validate that the approaches can be used practically.

Since virtual node is an integrated concept within the ProCom component model [21], we implemented an example case study in the PRIDE tool [25] that supports the development of systems using ProCom components running on the HSF implementation [11]. We have used ProCom components for development of a cruise controller and an adaptive cruise controller for automotive applications. Our motivating case study is simple, but exercises the execution-time properties and evaluates the integration and reusability of the run-time components. The case study demonstrates the temporal-fault containment within an RVN as well as the reuse of RVNs in new environment thereby facilitating predictable integration.

For MRS, we have demonstrated that MRS can be used to "encapsulate" legacy systems and to give them enough resources to fulfill their purpose [24]. In our case study a legacy media-player is integrated with several resource-hungry tasks running at a different core. We show that without MRS the media-player starts to drop frames due to the interference from other tasks; while the use of MRS alleviates this problem.

E. Presenting schedulability analysis for MRS

In this contribution we address challenge 4, and we describe the problem of achieving composability of independently developed real-time subsystems to be executed on a multi-core platform, and we provide a solution to tackle it [26]. First, we evaluate existing work for achieving real-time predictability on multi-cores and illustrate their lack with respect to composability.

Secondly, to analyze that the multi-resource servers provide composable hierarchical scheduling on multi-core platforms, we extend traditional schedulabity analysis for the multi-resource server. We outline a theoretical framework to provide hard real-time guarantees for tasks executing inside a multi-resource server and have presented a local schedulability analysis technique to assess the composability of subsystems containing hard real-time tasks [26]. Using the compositional analysis technique, the system schedulability is checked by composing the subsystems interfaces which abstracts the resource demand of subsystems [4].

III. Conclusions and future work

We have presented and implemented the concept of runable virtual node to bind the functional and temporal properties within the component to achieve predictable integration of real-time components. Our results demonstrate this predictable effect during component's reuse as well. Furthermore, we have presented and implemented the server-based communication strategy to make the communication among RVNs predictable.

For multicore platforms, the scheduling is inherently unpredictable due to the shared resources (in addition to CPU) like last-level shared caches, memory-bus and memory. We have proposed a solution of using Multi-Resource Server that supports both the CPU- and memory bus-portioning among applications executing on concurrent cores. We have implemented the server and our results have shown improvements in isolation. However, our solution does not guarantee complete isolation, as last-level cache and memory is still shared among applications.

In future, we intend to implement some software-based cache-partitioning technique (like [9]) to support complete temporal isolation on multicore platforms as well. Page coloring is a software-based technique that provides both cache-partitioning and memory partitioning by assigning memory pages to the caches. We intend to enhance the isolation of MRS by incorporating this technique. We provided the initial local compositional schedulabitlity analysis for the server in [26]. We also intend to complete the schedulability analysis for MRS by incorporating the cache effects and the bus scheduling into it.

References

[1] I. Crnkovic and M. Larsson, editors. *Building Reliable Component-Based Software Systems*. Artech House publisher, 2002. ISBN 1-58053-327-2.

[2] R. N. Charette. "This car runs on code". *Spectrum*, 46(2), 2009. "http://spectrum.ieee.org/greentech/advanced-cars/this-car-runs-on-code".

[3] Charlotte Adams. Product focus: Cots operating systems: Boarding the boeing 787, 2005. [Online]. Available: http://www.aviationtoday.com/, last checked: 20.03.2013.

[4] I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In *Proc. 24^{th} IEEE Real-Time Systems Symposium (RTSS' 03)*, pages 2–13, December 2003.

[5] Z. Deng and J. W.-S. Liu. Scheduling real-time applications in an open environment. In *Proc. 18^{th} IEEE Real-Time Systems Symposium (RTSS' 97)*, December 1997.

[6] T. Nolte, I. Shin, M. Behnam, and M. Sjödin. A Synchronization Protocol for Temporal Isolation of Software Components in Vehicular Systems. *IEEE Transactions on Industrial Informatics*, 5(4):375–387, November 2009.

[7] J. P. Lehoczky, L. Sha, and J.K. Strosnider. Enhanced aperiodic responsiveness in hard real-time environments. In *Proc. 8^{th} IEEE Real-Time Systems Symposium (RTSS' 87)*, pages 261–270, December 1987.

[8] B. Sprunt, L. Sha, and J. P. Lehoczky. Aperiodic Task Scheduling for Hard Real-Time Systems. *Real-Time Systems*, 1(1):27–60, June 1989.

[9] R. Mancuso, R. Dudko, E. Betti, M. Cesati, M. Caccamo, and R. Pellizzoni. Real-time cache management framework for multi-core architectures. In *Proc. 19^{th} IEEE Real-Time Technology and Applications Symposium (RTAS' 13)*, pages 45–54, 2013.

[10] S. Schliecker and R. Ernst. Real-time Performance Analysis of Multiprocessor Systems with Shared Memory. *ACM Transactions in Embedded Computing Systems*, 10(2):22:1–22:27, January 2011.

[11] R. Inam, J. Mäki-Turja, M. Sjödin, S. M. H. Ashjaei, and S. Afshar. Support for Hierarchical Scheduling in FreeRTOS. In *16th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA' 11)*, France, September 2011.

[12] Richard Barry. *Using the FreeRTOS Real Time Kernel*. Real Time Engineers Ltd., 2010.

[13] L. Sha, J.P. Lehoczky, and R. Rajkumar. Solutions for some Practical problems in Prioritised Preemptive Scheduling. In *Proc. 7^{th} IEEE Real-Time Systems Symposium (RTSS' 86)*, pages 181–191, December 1986.

[14] 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.

[15] R. Inam, J. Mäki-Turja, M. Sjödin, and M. Behnam. Hard Real-time Support for Hierarchical Scheduling in FreeRTOS. In *7th Annual Workshop (OSPERT' 11)*, pages 51–60, Porto, Portugal, July 2011.

[16] Rafia Inam, Jukka Mäki-Turja, Jan Carlson, and Mikael Sjödin. Using temporal isolation to achieve predictable integration of real-time components. In *22nd Euromicro Conference on Real-Time Systems (ECRTS' 10) WiP Session*, pages 17–20, July 2010.

[17] 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), January 2012.

[18] R. Inam, J. Mäki-Turja, M. Sjödin, and J. Kunčar. Real-Time Component Integration using Runnable Virtual Nodes. In *Proc. of the 38th Euromicro Conference on Software Engineering and Advanced Applications (SEAA' 12)*.

[19] R. Inam and M. Sjödin. Implementing and Evaluating Communication-Strategies in the ProCom Component Technology. In *Proc. of the 24th Euromicro Conference on Real-Time Systems (ECRTS' 12), WiP*. ACM SIGBED Review, July 2012.

[20] R. Inam, J. Carlson, M. Sjödin, and J. Kunčar. Predictable integration and reuse of executable real-time components. *Journal of Systems and Software*, 91(0):147 – 162, 2014.

[21] 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.

[22] R. Inam, M. Sjödin, and M. Jägemar. Bandwidth Measurement using Performance Counters for Predictable Multicore Software. In *17th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA' 12), WiP*, pages 1–4, September 2012.

[23] R. Inam, J. Slatman, M. Behnam, M. Sjödin, and T. Nolte. Towards implementing Multi-Resource Server on multi-core Linux platform. In *18th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA' 13), WiP*, pages 1–4, September 2013.

[24] R. Inam, N. Mahmud, M. Behnam, T. Nolte, and M. Sjödin. The Multi-Resource Server for predictable execution on multi-core platforms. In *(RTAS' 14)*, April 2014.

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

[26] M. Behnam, R. Inam, T. Nolte, and M. Sjödin. Multi-core composability in the face of memory bus contention. In *5th International Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS'12)*. ACM, December 2012.