TOWARDS A PREDICTABLE COMPONENT-BASED RUN-TIME SYSTEM

Rafia Inam

2012

Mälardalen University Press Licentiate Theses
No. 145

School of Innovation, Design and Engineering
Abstract

In this thesis we propose a technique to preserve temporal properties of real-time components during their integration and reuse. We propose a new concept of runnable virtual node which is a coarse-grained real-time component that provides functional and temporal isolation with respect to its environment. A virtual node’s interaction with the environment is bounded by both 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.

The first major contribution of this thesis is the implementation of a Hierarchical Scheduling Framework (HSF) on an open source real-time operating system (FreeRTOS) with the emphasis of doing minimal changes to the underlying FreeRTOS kernel and keeping its API intact to support the temporal isolation between a numbers of applications, on a single processor. Temporal isolation between the components during runtime prevents failure propagation between different components.

The second contribution of the thesis is with respect to the integration of components, where we first illustrate how the concept of the runnable virtual node can be integrated in several component technologies and, secondly, we perform a proof-of-concept case study for the ProCom component technology where we demonstrate the runnable virtual node’s real-time properties for temporal isolations and reusability.

We have performed experimental evaluations on EVK1100 A VR based 32-bit micro-controller and have validated the system behaviour during heavy-load and over-load situations by visualizing execution traces in both hierarchical scheduling and virtual node contexts. The results for the case study demonstrate temporal error containment within a runnable virtual node as well as reuse of the node in a new environment without altering its temporal behaviour.
Abstract

In this thesis we propose a technique to preserve temporal properties of real-time components during their integration and reuse. We propose a new concept of runnable virtual node which is a coarse-grained real-time component that provides functional and temporal isolation with respect to its environment. A virtual node’s interaction with the environment is bounded by both 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.

The first major contribution of this thesis is the implementation of a Hierarchical Scheduling Framework (HSF) on an open source real-time operating system (FreeRTOS) with the emphasis of doing minimal changes to the underlying FreeRTOS kernel and keeping its API intact to support the temporal isolation between a numbers of applications, on a single processor. Temporal isolation between the components during runtime prevents failure propagation between different components.

The second contribution of the thesis is with respect to the integration of components, where we first illustrate how the concept of the runnable virtual node can be integrated in several component technologies and, secondly, we perform a proof-of-concept case study for the ProCom component technology where we demonstrate the runnable virtual node’s real-time properties for temporal isolations and reusability.

We have performed experimental evaluations on EVK1100 AVR based 32-bit micro-controller and have validated the system behaviour during heavy-load and over-load situations by visualizing execution traces in both hierarchical scheduling and virtual node contexts. The results for the case study demonstrate temporal error containment within a runnable virtual node as well as reuse of the node in a new environment without altering its temporal behaviour.
To my husband Inam
To my husband Inam
Acknowledgments

First of all, I am grateful to my supervisors Professor Mikael Sjödin and Dr. Jukka Mäki-Turja without whose guidance and assistance this study would not have been successful. I specially thank Prof. Mikael Sjödin for his advices, invaluable inputs, support and encouragement, and always finding time to help me.

Many thanks go to Prof. Philippas Tsigas for informing me about the PhD position and encouraging me to apply at MRTC for a position.

I have attended a number of courses during my studies. I would like to give many thanks to Hans Hansson, Ivica Crnkovic, Mikael Sjödin, Thomas Nolte, Emma Nehrenheim, Daniel Sundmark, and Lena Dafgar for guiding me during my studies.

I want to thank the faculty members; Hans Hansson, Ivica Crnkovic, Paul Pettersson, Damir Isovic, Thomas Nolte, Dag Nyström, Cristina Seceleanu, Jan Carlson, Sasikumar Punnekkat, Björn Lisper, and Andreas Ermedahl for giving me vision to become a better student.

I would also like to thank to the whole administrative staff, in particular Gunnar, Malin, Susanne and Carola for their help in practical issues.

My special thanks also to all graduate friends, especially Sara D., Farhang, Andreas G., Aida, Aneta, Séverine, Svetlana, Ana, Adnan, Andreas H., Moris, Huseyin, Bob (Stefan), Luis (Yue), Hang, Mikael, Nima, Jagadish, Nikola, Federico, Saad, Mehrdad, Juraj, Luka, Leo, Josip, Barbara, Antonio, Abhilash, Lars, Batu, Mobyen, Shahina, Giacomo, Raluca, Eduard, and others for all the fun and memories.

I want to thank Moris, Farhang, Notle, Jan, Jiří, and Daniel Cederman - whom I have enjoyed working with. I supervised the three master students,
Acknowledgments

First of all, I am grateful to my supervisors Professor Mikael Sjödin and Dr. Jukka Mäki-Turja without whose guidance and assistance this study would not have been successful. I specially thank Prof. Mikael Sjödin for his advices, invaluable inputs, support and encouragement, and always finding time to help me.

Many thanks go to Prof. Philippas Tsigas for informing me about the PhD position and encouraging me to apply at MRTC for a position.

I have attended a number of courses during my studies. I would like to give many thanks to Hans Hansson, Ivica Crnkovic, Mikael Sjödin, Thomas Nolte, Emma Nehrenheim, Daniel Sundmark, and Lena Dafgård for guiding me during my studies.

I want to thank the faculty members; Hans Hansson, Ivica Crnkovic, Paul Pettersson, Damir Isovic, Thomas Nolte, Dag Nyström, Cristina Secceleanu, Jan Carlson, Sasikumar Punnekkat, Björn Lisper, and Andreas Ermedahl for giving me vision to become a better student.

I would also like to thank to the whole administrative staff, in particular Gunnar, Malin, Susanne and Carola for their help in practical issues.

My special thanks also to all graduate friends, especially Sara D., Farhang, Andreas G., Aida, Aneta, Séverine, Svetlana, Ana, Adnan, Andreas H., Moris, Hüseyin, Bob (Stefan), Luis (Yue), Hang, Mikael, Nima, Jagadish, Nikola, Federico, Saad, Mehrdad, Juraj, Luka, Leo, Josip, Barbara, Antonio, Abhilash, Lars, Batu, Mobyen, Shahina, Giacomo, Raluca, Eduard, and others for all the fun and memories.

I want to thank Moris, Farhang, Notle, Jan, Jiří, and Daniel Cederman - whom I have enjoyed working with. I supervised the three master students,
Mohammad, Sara A., and Wu. I wish them best of luck.

Finally, I would like to extend my deepest gratitude to my family. Many thanks go to my parents for their support and unconditional love in my life. My deepest gratitude goes to my husband Inam for being always positive and supportive in all these rough and tough days and to my daughters Youmna and Urwa for bringing endless love and happiness to our lives.

This work has been supported by the Swedish Foundation for Strategic Research (SSF), via the research programme PROGRESS.

Rafia Inam
Västerås, January, 2012
List of Publications

Papers Included in the Licentiate Thesis


---

1The included articles have been reformatted to comply with the licentiate layout.
Additional Papers, not Included in the Licentiate Thesis

Conferences and Workshops


Technical reports


## Contents

**I Thesis**  
1 Introduction 3  
  1.1 Research Problem ........................................... 3  
  1.2 Proposal .......................................................... 4  
    1.2.1 Runnable Virtual Node ................................. 4  
  1.3 Contributions .................................................. 5  
  1.4 Background ..................................................... 7  
    1.4.1 ProCom Component Model ............................... 7  
    1.4.2 Hierarchical Scheduling Framework ................... 9  
  1.5 Thesis Overview ............................................... 11  

2 Research Overview 15  
  2.1 Research Goal .................................................. 15  
  2.2 Research Methodology and Research Guiding Questions ...... 15  

3 Runnable Virtual Node 19  
  3.1 Runnable Virtual Node Concept .............................. 19  
  3.2 HSF Implementation in FreeRTOS ............................ 20  
  3.3 Applying Runnable Virtual Node in ProCom ................. 23  
    3.3.1 Modeling Level ............................................. 23  
    3.3.2 Execution Level .......................................... 23  
    3.3.3 The Synthesis of the Final Executables ............... 24  
  3.4 Evaluation Through a Case Study ........................... 24  

4 Conclusions and Future Work 29  
  4.1 Summary .......................................................... 29  
  4.2 Questions Revisited .............................................. 30
# 4.3 Future Research Directions
Bibliography

## II Included Papers

### 5 Paper A: Virtual Node – To Achieve Temporal Isolation and Predictable Integration of Real-Time Components

| Section | Page |
|---------|------|---|
| 5.1 Introduction | 41 |
| 5.1.1 Contributions | 41 |
| 5.2 Component Technologies | 42 |
| 5.2.1 ProCom | 42 |
| 5.2.2 AUTOSAR | 45 |
| 5.2.3 AADL | 47 |
| 5.3 Virtual Node | 48 |
| 5.3.1 Hierarchical Scheduling Framework (HSF) | 49 |
| 5.3.2 HSF Implementation in FreeRTOS | 50 |
| 5.4 Applying Virtual Node Concept to ProCom, AUTOSAR, and AADL | 51 |
| 5.4.1 ProCom | 52 |
| 5.4.2 AUTOSAR | 52 |
| 5.4.3 AADL | 53 |
| 5.5 Conclusions and Future Work | 53 |

Bibliography

### 6 Paper B: Support for Hierarchical Scheduling in FreeRTOS

| Section | Page |
|---------|------|---|
| 6.1 Introduction | 61 |
| 6.1.1 Contributions | 61 |
| 6.1.2 The Hierarchical Scheduling Framework | 62 |
| 6.2 Related Work | 62 |
| 6.2.1 Hierarchical Scheduling | 62 |
| 6.2.2 Implementations of Hierarchical Scheduling Framework | 63 |
| 6.3 System Model | 64 |
| 6.3.1 Subsystem Model | 64 |
| 6.3.2 Task Model | 65 |
| 6.3.3 Scheduling Policy | 65 |
| 6.4 FreeRTOS | 65 |
## 7 Paper C:
**Hard Real-time Support for Hierarchical Scheduling in FreeRTOS**  91

7.1 Introduction ........................................... 93
   7.1.1 Contributions ..................................... 93
   7.1.2 Resource Sharing in Hierarchical Scheduling Framework 94
7.2 Related Work ........................................... 95
   7.2.1 Local and Global Synchronization Protocols ............ 96
   7.2.2 Implementations of Resource Sharing in HSF .......... 97
7.3 Background ............................................. 98
   7.3.1 FreeRTOS ......................................... 98
   7.3.2 A Review of HSF Implementation in FreeRTOS .......... 99
   7.3.3 Resource Sharing in HSF ............................ 100
   7.3.4 Overrun Mechanisms ............................... 101
7.4 System Model ......................................... 102
   7.4.1 Subsystem Model ................................... 102
   7.4.2 Task Model ....................................... 103
   7.4.3 Scheduling Policy ................................. 103
   7.4.4 Design Considerations ............................. 103
7.5 Implementation ....................................... 104
   7.5.1 Support for Time-Triggered Periodic Tasks ............ 104
   7.5.2 Support for Legacy System .......................... 105
### Paper D:
**Run-Time Component Integration and Reuse in Cyber-Physical Systems**

<table>
<thead>
<tr>
<th>Section</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.1 Introduction</td>
<td>127</td>
</tr>
<tr>
<td>8.2 Related Work</td>
<td>129</td>
</tr>
<tr>
<td>8.2.1 AUTOSAR</td>
<td>129</td>
</tr>
<tr>
<td>8.2.2 Rubus</td>
<td>130</td>
</tr>
<tr>
<td>8.2.3 AADL</td>
<td>131</td>
</tr>
<tr>
<td>8.2.4 Deployment and Configuration specification</td>
<td>131</td>
</tr>
<tr>
<td>8.3 Background</td>
<td>132</td>
</tr>
<tr>
<td>8.3.1 ProCom Component Model</td>
<td>132</td>
</tr>
<tr>
<td>8.3.2 Hierarchical Scheduling Framework</td>
<td>135</td>
</tr>
<tr>
<td>8.3.3 FreeRTOS and its HSF Implementation</td>
<td>136</td>
</tr>
<tr>
<td>8.4 Runnable Virtual Node</td>
<td>137</td>
</tr>
<tr>
<td>8.4.1 Applying Virtual Node Concept to ProCom Component Model</td>
<td>137</td>
</tr>
<tr>
<td>8.4.2 The Synthesis of the Final Executables</td>
<td>138</td>
</tr>
<tr>
<td>8.5 Case Study: Cruise controller and an adaptive cruise controller</td>
<td>138</td>
</tr>
<tr>
<td>8.5.1 System design</td>
<td>139</td>
</tr>
<tr>
<td>8.5.2 Synthesis</td>
<td>141</td>
</tr>
<tr>
<td>8.5.3 Evaluation and Discussion</td>
<td>144</td>
</tr>
<tr>
<td>8.6 Conclusions</td>
<td>148</td>
</tr>
<tr>
<td>Bibliography</td>
<td>151</td>
</tr>
</tbody>
</table>
I

Thesis
Chapter 1
Introduction
In embedded real-time electronic systems, a continuous increasing trend in size and complexity of embedded software has been observed during the last decades. To battle this trend, modern software-development technologies are being adopted by the real-time industry. One such technology is Component-Based Software Engineering (CBSE), where the system is divided into a set of interconnected components [1]. Components have well-defined functional interfaces which define both provided and required services. However, the functional interfaces do not capture timing behavior or temporal requirements. Further, the advent of low cost and high performance 8, 16, and 32-bit micro-controllers, have made possible to integrate more than one complex real-time components on a single hardware node. For systems with real-time requirements, this integration poses new challenges.

The aim of this thesis is to investigate techniques for predictable integration of software components with real-time requirements. Further the real-time properties of the components should be maintained for reuse in real-time embedded systems.

1.1 Research Problem
Temporal behavior of real-time software components poses difficulties in their integration. When multiple components are deployed on the same hardware node, the emerging timing behavior is unpredictable. This means that a component that is found correct during unit test may fail, due to a change in temporal behavior.
Chapter 1

Introduction

In embedded real-time electronic 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. One such technology is Component-Based Software Engineering (CBSE), where the system is divided into a set of interconnected components [1]. Components have well-defined functional interfaces which define both provided and required services. However, the functional interfaces do not capture timing behavior or temporal requirements. Further, the advent of low cost and high performance 8, 16, and 32-bit micro-controllers, have made possible to integrate more than one complex real-time components on a single hardware node. For systems with real-time requirements, this integration poses new challenges.

The aim of this thesis is to investigate techniques for predictable integration of software components with real-time requirements. Further the real-time properties of the components should be maintained for reuse in real-time embedded systems.

1.1 Research Problem

Temporal behavior of real-time software components poses difficulties in their integration. When multiple components are deployed on the same hardware node, the emerging timing behavior is unpredictable. This means that a component that is found correct during unit test may fail, due to a change in tem-
poral 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.

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 control systems, methodologies and techniques are required to provide temporal isolation so that the run-time timing properties could be guaranteed.

1.2 Proposal

In this thesis we address the challenges of encapsulating real-time properties within the components, in order to make the integration of real-time components predictable, and to ease component reuse in new systems. The purpose is to preserve the timing properties within the components thus component integration and reuse can be made predictable.

To achieve this, the real-time properties are encapsulated into reusable components, and hierarchical scheduling is used to provide temporal isolation and predictable integration among the components that further leads to the increased reusability of the components [2, 3, 4].

1.2.1 Runnable Virtual Node

We propose the concept of a runnable virtual node, which is an execution-platform concept that preserves the temporal properties of software executed in it [3]. It introduces an intermediate level between the functional entities (e.g. components or tasks) and the physical nodes. Thereby it leads to a two-level deployment process instead of a single big-stepped deployment; i.e. first deploying functional entities to the virtual nodes and then deploying virtual nodes to the physical nodes.

The virtual node is intended for coarse-grained components for single node deployment and with potential internal multitasking. We envision a handful of components (less than 50) per physical node. Hierarchical scheduling technique is embedded within the runnable virtual node to encapsulate the timing requirements within the components. Using an Hierarchical Scheduling
Chapter 1. Introduction

poral 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.

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 control systems, methodologies and techniques are required to provide temporal isolation so that the run-time timing properties could be guaranteed.

1.2 Proposal

In this thesis we address the challenges of encapsulating real-time properties within the components, in order to make the integration of real-time components predictable, and to ease component reuse in new systems. The purpose is to preserve the timing properties within the components thus component integration and reuse can be made predictable.

To achieve this, the real-time properties are encapsulated into reusable components, and hierarchical scheduling is used to provide temporal isolation and predictable integration among the components that further leads to the increased reusability of the components [2, 3, 4].

1.2.1 Runnable Virtual Node

We propose the concept of a runnable virtual node, which is an execution-platform concept that preserves the temporal properties of software executed in it [3]. It introduces an intermediate level between the functional entities (e.g. components or tasks) and the physical nodes. Thereby it leads to a two-level deployment process instead of a single big-stepped deployment; i.e. first deploying functional entities to the virtual nodes and then deploying virtual nodes to the physical nodes.

The virtual node is intended for coarse-grained components for single node deployment and with potential internal multitasking. We envision a handful of components (less than 50) per physical node. Hierarchical scheduling technique is embedded within the runnable virtual node to encapsulate the timing requirements within the components. Using an Hierarchical Scheduling Framework (HSF) a subsystem (runnable virtual node in our case) are developed and analyzed in isolation, with its own local scheduler at first step of deployment and its temporal properties are validated. Then at the second step of deployment, multiple subsystems are integrated onto a physical node using a global scheduler without violating the temporal properties of the individual subsystems.

The runnable virtual node takes the advantages of both component-based software engineering and hierarchical scheduling approaches. It exploits encapsulation and reusability benefits of CBSE [1], and the temporal isolation and concurrent development and analysis of subsystems in isolation of HSF [5]. Moreover, combining the two approaches, results in the additional benefits of predictable integration and reuse of timing properties of the real-time components.

1.3 Contributions

The contributions presented in this thesis can be divided into two main parts:

HSF Implementation

HSF has attained a substantial importance since introduced in 1990 by Deng and Liu [6]. Numerous studies has been performed for the schedulability analysis of HSFs [7, 8] and processor models [9, 10, 11, 12] for independent subsystems. The main focus of this research has been on the schedulability analysis and not much work has been done to implement these theories.

We present our work towards an implementation of the hierarchical scheduling framework in an open source real-time operating system, FreeRTOS [13], to support temporal isolation among real-time components. We implement idling periodic and deferrable servers using fixed-priority preemptive scheduling at both local and global scheduling levels. We focus on being consistent with the underlying operating system and doing minimal changes to get better utilization of the system and keeping its API intact.

Allowing tasks from different subsystems to share logical resources imposes more complexity for the scheduling of subsystems. A proper synchronization protocol should be used to prevent unpredictable timing behavior of the real-time system. We extend the implementation of two-level hierarchical scheduling framework for FreeRTOS with the provision of resource sharing at two levels: (i) local resource sharing (among the tasks of the same subsystem)
using the Stack Resource Policy (SRP) [14], and (ii) global resource sharing using the Hierarchical Stack Resource Policy (HSRP) [15] with three different methods to handle overrun (with payback, without payback, and enhanced overrun) [16]. Moreover, we extend the HSF implementation to use in hard-real time applications, with the possibility to include legacy applications and components not explicitly developed for hard real-time or the HSF.

We test our implementation on EVK1100 AVR32UC3A0512 micro-controller [17]. To test the efficiency of the implementation, we measure the overheads imposed by the HSF implementation during heavy-load and over-load situations. Moreover, we evaluate the overheads and behavior for different alternative implementations of HSRP with overrun from experiments on the board. In addition, real-time scheduling analysis with models of the overheads of our implementation is presented.

**Presentation and Realization of Runnable Virtual Node Concept**

Runnable virtual node is proposed as a means to achieve predictable integration and reuse of software components. Runnable virtual node is a coarse-grained real-time component encapsulating the timing properties and with potential internal multitasking. We present to utilize the hierarchical scheduling within the component-based technology to retain temporal properties, increasing predictability during components integration that further leads to the increased reuse of the real-time components. We believe that our idea can be easily generalized. We present how it can be applied to other commercial component-based technologies like AUTOSAR and AADL.

As a specific example, we realize the idea of runnable virtual node using the ProCom component technology and validate that its internal temporal behavior is preserved when integrated with other components or when reused in a new environment. Our realization of runnable virtual node exploits the latest techniques for hierarchical scheduling to achieve temporal isolation, and the principles from component-based software-engineering to achieve functional isolation. It uses a two-level deployment process (instead of a single big-stepped deployment) i.e. deploying functional entities to the virtual nodes and then deploying virtual nodes to the physical nodes, thereby preserving the timing properties within the components in addition to their functional properties. We perform a proof-of-concept case study, implemented in the ProCom component-technology executing on top of FreeRTOS based hierarchical scheduling framework to validate the temporal isolation among components and to test the reuse of components.
1.4 Background

This section presents the background technologies our work uses. We provide an overview of the ProCom component technology, used to realize the runnable virtual node concept. It is followed by an introduction of the hierarchical scheduling framework.

1.4.1 ProCom Component Model

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. The ProCom component technology 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].

![Diagram of ProCom component model]

Figure 1.1: An overview of the deployment modelling formalisms and synthesis artefacts.

The ProCom component model can be described in two distinct realms: the modeling and the runnable realms as shown in Figure 1.1. In Modeling realm, the models are made using CBSE and MBE while in 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 1.1. 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 ProSave and ProSys allow composite components. For details on ProSave and ProSys, including the motivation for separating the two, see [18, 19].

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 required by those components. After that many virtual nodes could be mapped on a physical node i.e. an ECU (Electronic Control Unit). The relationship is again many-to-one. Details about the deployment modeling are provided in [4].

### The Runnable Realm

At the runnable realm, runnables/executables are synthesized from the ProCom model entities. The primitive ProSave components are represented as a simple C language source code in runnable form. From this C code, the ProSys runnables are generated which contain the collection of operating system tasks. Virtual nodes, called runnable virtual nodes here, implement the local scheduler and contain the tasks in a server. Hence a runnable virtual node 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 is generated by connecting different virtual nodes together with a global scheduler and using the middleware to provide intra-communications among the virtual node executables.
Deployment and Synthesis Activities

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:

- 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 (CPU allocation).

- The virtual nodes are then deployed on the physical nodes. The relationship is again many-to-one, which means that 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. Chapter 3 describes this further.

The PROGRESS Integrated Development Environment (PRIDE) tool [20] supports the automatic synthesis of the components at different levels [21]. At the ProSave level, the XML descriptions of the components is the input and the C files are generated containing the basic functionality. At the second level, ProSys components are assigned to the tasks to generate ProSys runnables. Since the tasks at this level are independent of the execution platform, therefore, the only attribute assigned at this stage is the period for each task; which they get from the clock frequency that is triggering the specific component. Other task attributes like priority are dependent on the underlying platform, hence assigned during later stages of the synthesis. A clock defines the periodic triggering of components with a specified frequency. Components are allocated to a task when (i) the components are triggered by the same event, (ii) when the components have precedence relation among them to be preserved.

1.4.2 Hierarchical Scheduling Framework

Hierarchical scheduling has shown to be a useful approach in supporting modularity of real-time software [22] by providing temporal partitioning among applications. A two-level hierarchical scheduling framework [23] is used to provide the temporal isolation among a set of subsystems. In hierarchical scheduling, the CPU time is partitioned among many subsystems (or servers),
that are scheduled by a global (system-level) scheduler. Each subsystem contains its own internal set of tasks that are scheduled by a local (subsystem-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 1.2. The parent node is a global scheduler that schedules subsystems. Each subsystem has its own local scheduler, that schedules the tasks within the subsystem. The subsystem integration involves a system-level schedulability test, verifying that all timing requirements are met.

The HSF gives the potential to develop and analyze subsystems in isolation from each other [24]. As each subsystem has its own local scheduler, after satisfying the temporal constraints, the temporal properties are saved within each

Figure 1.2: Two-level Hierarchical Scheduling Framework
subsystem. Later, a global scheduler is used to schedule all the subsystems together without violating the temporal constraints that are already analyzed and stored in the subsystems. 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.

1.5 Thesis Overview

The thesis is organized in two distinctive parts. Part-I gives a summary of the performed research. Chapter 1 describes the motivation and background of the research. Chapter 2 formulates the main research goal, describes the research method we used, and introduces research questions used as guideline to perform the research. Chapter 3 describes our approach of runnable virtual node, and some results of our research. Finally Chapter 4 concludes the thesis by summarizing the contributions and outlining the future work.

Part-II includes three peer-reviewed scientific papers and one technical report contributing to the research results. These papers are published and presented in international conference and workshop, or international journals and are presented in Chapters 5-7. The technical report is submitted for conference publishing and is presented in Chapter 8. A short description and contribution of these papers and the report is given as follows:


Short Summary: This paper presents an approach of two-level deployment process for component models used in the 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 these analysis is also reusable with the reuse of virtual node. Hence virtual node preserves real-time properties across reuse
and integration in different contexts.

We have presented the methods to realize the idea of virtual node concept within the ProCom, AUTOSAR, and AADL component models.

**Contribution:** I initiated this journal paper. I was involved in most parts of this paper. It has been a joint effort between me and all the authors.


**Short Summary:** This paper presents the implementation of hierarchical scheduling framework on an open source real-time operating system FreeRTOS to support the temporal isolation of a number of real-time components (or applications) on a single processor. The goal is to achieve predictable integration and reusability of independently developed components or tasks. It presents the initial results of the HSF implementation by running it on an AVR 32-bit board EVK1100.

The paper addresses the fixed-priority preemptive scheduling at both global and local scheduling levels. It describes the detailed design of HSF with the emphasis of doing minimal changes to the underlying FreeRTOS kernel and keeping its API intact. Finally it provides (and compares) the results for the performance measures of periodic and deferrable servers with respect to the overhead of the implementation.

**Contribution:** I was the initiator and author to all parts in this paper. I have contributed in the design of HSF implementation and have designed all the test cases and have performed the experiments. I supervised the students Mohammed and Sara who were responsible of the implementation part.

Short Summary: This paper presents extensions to the previous implementation of two-level Hierarchical Scheduling Framework (HSF) for FreeRTOS. The results presented here allow the use of HSF for FreeRTOS in hard-real time applications, with the possibility to include legacy applications and components not explicitly developed for hard real-time or the HSF.

Specifically, we present the implementations of (i) global and local resource sharing using the Hierarchical Stack Resource Policy and Stack Resource Policy respectively, (ii) kernel support for the periodic task model, and (iii) mapping of original FreeRTOS API to the extended FreeRTOS HSF API. We also present evaluations of overheads and behavior for different alternative implementations of HSRP with overrun from experiments on the AVR 32-bit board EVK1100. In addition, real-time scheduling analysis with models of the overheads of our implementation is presented.

Contribution: I was the initiator and the main author to majority parts in this paper. I have contributed in the design of HSF implementation and have designed all the test cases and have performed the experiments. Moris included the implementation overheads to the schedulability analysis and wrote that section.


Short Summary: This paper presents the concept of runnable virtual nodes as a means to achieve predictable integration and reuse of software components in cyber-physical systems. A runnable virtual node is a coarse-grained real-time component that provides functional and temporal isolation with respect to its environment. Its interaction with the environment is bounded both by a functional and a temporal interface, and the validity of its internal temporal behavior is preserved when integrated with other components or when reused in a new environment.

Our realization of runnable virtual nodes exploits the latest techniques for hierarchical scheduling to achieve temporal isolation, and the principles from component-based software-engineering to achieve functional isolation. In the paper we present a proof-of-concept case study, implemented in the ProCom
component-technology executing on top of FreeRTOS based hierarchical scheduling framework.

Contribution: I was the initiator and author to all parts in this paper. I have contributed in the design of the case study. Jiří was responsible of implementing the case study in Progress IDE, called PRIDE, and I executed it on the target platform using AVR Studio and performed all the tests and experiments.
Chapter 2

Research Overview

2.1 Research Goal

The aim of the thesis is to support management of real-time properties in component-based systems by achieving predictable integrations and reusability of those components. To achieve this, new methods and tools for embedded real-time component based systems has to be introduced where timing properties of the components can be preserved, and real-time components integration can be made predictable. Thus the main goal that the thesis aims is:

To provide methods to maintain the real-time properties of runtime components during their integration with other components and their reuse in a new environment.

The goal is broad and is sub-divided into smaller research problems to solve it, which collectively approach the main goal as explained in the following section.

2.2 Research Methodology and Research Guiding Questions

This research work is carried out following the methodology based on the steps proposed/described by Shaw [25]. The main research activities we followed are:
Figure 2.1: The main research steps.

1. Study of the current state-of-the-art and identification of the research problem on current trends from real-time and component-based communities and definition of the research goal.

2. Refining the problem to research settings and defining research guiding questions.

3. Analysis of the current state-of-the-art in the literature based on guiding questions and proposing a solution.

4. Solving the proposed sub-problem (either by implementing a prototype and/or by providing mathematical analysis) and presenting the research results.

5. Illustration and validation of the research results. This is done by performing experimental evaluations of the implementation, and by per-
forming a case study. In case of mathematical analysis, the validation is done by implementation/formal proofs.

In this work, a big goal is identified at step 1, which is sub-divided into smaller research goals, and solved one at a time using steps 2 to 5. These steps (2 - 5) are performed repeatedly unless the desired results for the overall goal are achieved as described in Figure 2.1.

In the following section the research performed is described in brief.

- We have first identified and defined the initial problem.

- From this research, we performed some preliminary studies and formulated some initial research questions. Since our main focus is on maintaining the real-time properties within the components to attain their predictable integration, we first investigated within the real-time community for the suitable technique. Then we explored the possibilities to combine this technique in the component-based systems. It resulted in proposing a preliminary idea to merge the real-time technique into the component-based systems to achieve predictable integrations and reusability of those components [2, 3, 4]. The research questions arise here are Q1 and Q2, presented at the end of this section.

- Detailed study of state-of-the-art in relevant areas resulted in writing Paper A [2, 3] in which we proposed to use HSF technique within the virtual node component. HSF is known as a technique for providing temporal isolation between applications in real-time community. By embedding HSF within the component technology, temporal isolation and predictable integration of components can be achieved. As a result of this research, the runnable virtual node concept was introduced in the ProCom component model in the synthesis realm.

- Lacking support for HSF for our intended target-platform, we realized the need to implement HSF. This lead us to question Q3. We first implemented hierarchical scheduling framework in FreeRTOS real-time operating system independently from software components. We performed a detailed experimental evaluation on the implementation to test its temporal behavior and performance measures on the target platform an AVR-based 32-bit EVK1100 board. It resulted in writing Paper B [26].

- From this implementation, we figured out the need for hard real-time support in the FreeRTOS operating system. Also the resource sharing
protocols for HSF were needed to be implemented in order to provide resource sharing among components. Hence we extended our implementation with these properties and tested on the same target platform. We also showed to include the overheads of our implementation in the schedulability analysis of HSF. It resulted in writing Paper C [27].

- To realize the runnable virtual node, HSF implementation is integrated within the ProCom component model as a proof of concept. We now needed to validate our approach, as formulated by questions Q4 and Q5. We performed an example case study on the platform and run experiments to validate the temporal isolation among runnable virtual nodes, their smooth integration and reusability. The resulted paper, Paper D [28] is in submission.

While performing the above mentioned steps, we iteratively identified the following research questions which we used as a guideline for our research:

Q1 Which techniques are used to achieve temporal isolations and predictable integration within the real-time community?

Q2 How can such a technique be integrated into component-technologies for embedded real-time systems?

Q3 How to efficiently implement hierarchical scheduling in a real-time operating system?

Q4 Can we demonstrate that we have achieved predictability and temporal isolations at run-time in real-time components and during their integration?

Q5 Can we demonstrate preservation of these temporal properties within the components when they are reused?
Chapter 3

Runnable Virtual Node

In this chapter, we describe the runnable virtual node concept in brief and how it is used within the ProCom technology. We also provide some details of our HSF implementation that is used within the runnable virtual node. Finally, we provide some of the results obtained by executing an example case study using ProCom model as a proof-of-concept.

3.1 Runnable Virtual Node Concept

The concept of runnable 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. Components’ reusability further reduces efforts related to software testing, validation and certification. This concept is based on a two-level deployment process as explained in Section 1.4.1. It indicates that the whole system is generated in two steps rather than a single big synthesis step. At the first level of deployment, the functional properties (functionality of components) are combined and preserved with their extra-functional properties (timing requirement) in the runnable virtual nodes. In this way it encapsulates the behavior with respect to timing and resource usage, and becomes a reusable component in addition to the design-time components. Followed by the second level of deployment, where these runnable virtual nodes are implemented on the physical platform along with a global scheduler.

A runnable virtual node includes the executable representation of the com-
ponents assigned to the tasks, a resource allocation (period and budget of server), and a real-time scheduler, to be executed within a server in the hierarchical scheduling framework.

### 3.2 HSF Implementation in FreeRTOS

In this work we use FreeRTOS as the operating system to execute both levels of the HSF. FreeRTOS is a portable open source real-time kernel with properties like small and scalable, support for 23 different hardware architectures, and ease to extend and maintain.

The official release of FreeRTOS only supports a single level fixed-priority scheduling. However, we have implemented a two-level HSF for FreeRTOS [26] with associated primitives for hard real-time sharing of resources both within and between servers [27]. The HSF supports reservations by associating a tuple \( < Q, P > \) to each server where \( P \) is the server period and \( Q (0 < Q \leq P) \) is the allocated portion of \( P \) and is called server budget. 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 [27].

The HSF implementation supports two kinds of servers, idling periodic [29] and deferrable servers [30]. An idle server is used in the system that has the lowest priority of all the other servers, i.e. 0. It will run when no other server in the system is ready to execute (in idling server). This is useful for maintaining and testing the temporal separation among servers and also useful in testing system behavior.

The implementation uses fixed priority preemptive scheduling (FPPS) for both global and local-level scheduling. The implementation is done with the considerations of being consistent with the underlying operating system, keeping the original API semantics, and doing minimal changes in FreeRTOS kernel to get minimal changes and better utilization of the system.

<table>
<thead>
<tr>
<th>Server</th>
<th>S1</th>
<th>S2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Period</td>
<td>20</td>
<td>40</td>
</tr>
<tr>
<td>Budget</td>
<td>10</td>
<td>15</td>
</tr>
</tbody>
</table>

Table 3.1: Servers used to test system behavior.

We have evaluated the performance measures for periodic and deferrable servers on an AVR 32-bit board EVK1100. To test the correctness of the
server’s behavior, the traces of the executions are visualized. The servers used to test the system are given in Table 3.1. Task properties and their assignments to the servers are given in Table 3.2. Note that higher number means higher
priority for both servers and tasks.

![Figure 3.2: Trace showing temporal isolation among idling servers](image)

The servers executions (according to their resource reservations) along with their task sets are presented in Figure 3.1 and Figure 3.2. In these Figures, the horizontal axis represents the execution time starting from 0. In the task’s visualization, the arrow represents task arrival and a gray rectangle means task execution. In the server’s visualization, the numbers along the vertical axis are the server’s capacity, the diagonal line represents the server execution while the horizontal line represents either the waiting time for the next activation (when budget has depleted) or the waiting for its turn to execute (when some other server is executing). Since these are idling periodic servers, all the servers in the system executes till budget depletion, if no task is ready then the idle task of that server executes till its budget depletion.

We have tested the system during normal load, and over loaded conditions. Especially, we tested the system for temporal isolation: The work load of one server was greater than 1, hence its lower priority task was missing its deadline. The same example is executed to perform this test but with the increased utilization of S1. The execution times of T1 and T2 are increased to 4 and 6 respectively, hence making the server S1 utilization greater than 1. Therefore the low priority task T1 misses its deadlines as shown by solid black lines in...
the Figure 3.2. S1 is never idling because it is overloaded. It is obvious from Figure 3.2, that the overload of S1 does not effect the behavior of S2 even though it has low priority.

The results for behavior testing for the deferrable server are given in Section 6.6.1. We also measure the overhead of the implementation, like tick handler, server context-switch and task context-switch. The detailed results can be found in Section 6.6.2.

For local resource-sharing (within a server) the Stack Resource Policy (SRP) [14] is used, and for global resource-sharing (between servers) the Hierarchical Stack Resource Policy (HSRP) protocol [15] is used with three different overrun mechanisms to deal with the server budget expiration within the critical section: (1) basic overrun without payback (BO), (2) overrun with payback (PO), and (3) enhanced overrun with payback (EO) [16]. The behavior test results for the resource sharing are provided in Section 7.7.1.

3.3 Applying Runnable Virtual Node in ProCom

In ProCom, a virtual node is an integrated model concept. It means that the virtual nodes exist both on the modeling level and on the synthesis level as explained in Section 1.4.1. In the synthesis realm they are called runnable virtual nodes.

3.3.1 Modeling Level

At modeling level, each virtual node contains a set of integrated ProSys components plus the execution resources (a period and budget) required for these ProSys components. The priorities of virtual nodes cannot be assigned at the modeling level. The priorities of a component are relative to other components in the system. Since virtual nodes are developed independently and are meant to be reused in different systems, therefore, the priorities are assigned to virtual nodes later during the synthesis process at the execution level.

3.3.2 Execution Level

At the execution level, the runnable virtual node contains a set of executable tasks, resources required to run those tasks and a real-time local scheduler to schedule these tasks. Note that the runnable virtual node is generated as a
result of first deployment step; it is platform independent and not executable as a stand-alone entity.

### 3.3.3 The Synthesis of the Final Executables

The final executables are generated by assigning priorities to the servers and tasks in the runnable virtual nodes, completing the task bodies with the user code, synthesizing communication among those nodes (if needed) and linking them together with the operating system. These executables then can be downloaded and executed on a physical node. As the real-time properties of runnable virtual nodes are preserved within the servers, therefore when integrated with other runnable virtual nodes on a physical node, the real-time properties of the whole integrated system will be guaranteed by the schedulability analysis of the whole system.

The communication among runnable virtual nodes is provided by a System server, which is automatically generated for inter-node communication (if needed), at this step. The main functionality of the server is to send and receive messages among the nodes. It contains two tasks to achieve this purpose: a sender task and a message-port updater task.

### 3.4 Evaluation Through a Case Study

The purpose of this case study is to evaluate and demonstrate the execution-time properties and reusability of the run-time components with real-time properties. The PROGRESS Integrated Development Environment (PRIDE) tool [20] supports development of systems using ProCom component models and it has been used for developing the examples of Cruise Controller (CC) and an Adaptive Cruise Controller (ACC) as shown in Figure 3.3.

First, the CC system was realized and exercised to test the temporal isolations among run-time components. Its basic functionality is to keep the vehicle at a constant speed. Then the ACC system extends this functionality by keeping a distance from the vehicle in front by autonomously adapting its speed to the speed of the preceding vehicle and by providing emergency brakes to avoid collisions. To evaluate the reusability of real-time components, the ACC system was realized by the reuse of some runnable virtual nodes from the CC system.

The design, automatic synthesis, and the generation of final binaries are described in details in Section 8.5. In the first step of deployment process,
two runnable virtual nodes are produced for both CC and ACC systems: one runnable virtual node for Virtual Node1 and one for Virtual Node2. These generated nodes contain tasks definitions in them. In the second step of the final synthesis/deployment part, the priorities are assigned to the runnable virtual nodes (also called servers now) and to the tasks in them. Four servers are generated for both examples. In addition to the system server and an idle server, two other servers CC, and VC are used for the CC application while ACC, and VC are used for the ACC application. In ACC application, the system and VC servers are reused. The priorities, periods and budgets for these servers are given in Table 3.3.

<table>
<thead>
<tr>
<th>Server</th>
<th>CC</th>
<th>ACC</th>
<th>VC</th>
<th>SYSTEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>Period</td>
<td>40</td>
<td>40</td>
<td>60</td>
<td>20</td>
</tr>
<tr>
<td>Budget</td>
<td>10</td>
<td>10</td>
<td>15</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 3.3: Servers used to test the CC and ACC systems behaviors.

The experiments are performed on the same EVK1100 board using the HSF implementation on FreeRTOS to test both the applications for the temporal isolations and reusability of the runnable virtual node and are provided in Section 8.5.3.

Figure 3.4 demonstrates the system execution under the overload situation to test the temporal isolation among the runnable virtual nodes. The execution times of tasks CCT1 and CCT2 are increased by adding the busy loops,
Figure 3.4: Trace showing temporal isolation during overload situation

hence making the $CC$ server’s utilization greater than 1. Therefore the low priority task $CCT2$ misses its deadlines at time 54. The $CC$ server is never idling because it is overloaded.

The overload of $CC$ server does not effect the behavior of any other server in the system as obvious from Figure 3.4. The $VC$ server has a lower prior-

Figure 3.5: Trace showing reusability of runnable virtual nodes in ACC system.

It shows a predictable timing behavior that eases their integration. It also manifests that the tempo-
eral errors are contained within the faulty runnable virtual node only and their effects are not propagated to the other nodes in the system.

Further, to test the reusability of the runnable virtual nodes, the ACC system is synthesized. It also contains four servers: the ACC server is synthesized with its task set while the other three servers are reused from the CC system.

The task set for the ACC server is different from that of CC server. It is clear from the Figure 3.5 that all the three reused servers sustain their timing behavior. For example, the VC server has a lower priority than ACC, still it’s behavior is not effected at all and remains similar to its behavior in the CC system. It confirms the predictable integration of real-time components on one hand, and demonstrates their reusability on the other hand.

We observed the same results on testing the ACC server with the changed timing properties, i.e. period 40 and budget 15. As long as the allocated budgets to servers (at the modeling level) are provided, the timing properties are guaranteed at the execution.

Hence, by the use of runnable virtual node components and two-level deployment process, the timing requirements are also encapsulated within the components along with their function requirements and the temporal partitioning is provided among the components (using HSF), that results in the increased predictability during component’s integration and making the runnable virtual nodes a reusable entity.
Chapter 4

Conclusions and Future Work

4.1 Summary

In this thesis we have presented an idea of the runnable virtual node using the two-level deployment process to meet the challenges of providing temporal properties, predictable integration and reusability of components with real-time properties. The runnable virtual node uses hierarchical scheduling mechanism to preserve temporal properties within the components as a means to achieve predictable integration and reuse of software in the real-time embedded systems. The runnable virtual node is intended as a coarse grained component for single node deployment and with potential internal multitasking. Each physical node is used to execute one or more virtual nodes.

We have implemented a two-level Hierarchical Scheduling Framework HSF in an open source real-time operating system, FreeRTOS, to support temporal isolation among real-time components. We have implemented idling periodic and deferrable servers using fixed-priority preemptive scheduling at both local and global scheduling levels of HSF. We have focused on being consistent with the underlying operating system and doing minimal changes to get better utilization of the system and kept the original FreeRTOS API semantics. We have added the Stack Resource Policy (SRP) to the FreeRTOS for efficient resource sharing by avoiding deadlocks. Further, we have implemented Hierarchical Stack Resource Policy (HSRP) and overrun mechanisms (with-
out payback, with payback, enhanced overrun) to share global resources in a two-level HFS implementation. We have provided a very simple and easy implementation to execute a legacy system in the HSF with the use of a single API.

We have tested our implementations and performed experimental evaluations on EVK1100 AVR based 32-bit micro-controller. We have validated the implementations during heavy-load and over-load situations. We have computed the overhead measurements (of tick handler, global scheduler, and task context-switch) and included them into the schedulability analysis.

The notion of two-level deployment process encapsulates the timing properties and uses the hierarchical scheduling within runnable virtual nodes that provides temporal isolation and increases the reuse of the component in different systems. Hence using runnable virtual nodes, a complex embedded system can be developed as a set of well defined reusable components encapsulating functional and timing properties.

Finally, we have performed a proof-of-concept case study which demonstrates temporal error containment within a virtual node as well as reuse of a virtual node in new environment without altering its temporal behavior. Our work is based on the ProCom component-technology running on HSF implementation on FreeRTOS. The case study was executed on an ECU with an AVR based 32-bit micro-controller. However, we believe that our concept is applicable also to commercial component technologies like AADL, AUTOSAR.

### 4.2 Questions Revisited

In this section we discuss: to which extent the research results and included papers fulfil the goals of Section 2.2. We also comment on the validity of our results.

Paper A addresses research questions Q1, Q2. We found that HSF is a known technique in real-time community that not only provide temporal isolation among subsystems, but also supports isolated and concurrent development of the subsystems. Further in this paper, we propose the idea of virtual node by integrating HSF into component technology to provide temporal isolations among components, that eventually leads to the predictable integration of components. We also present methods to integrate HSF within components and methods to incorporate the idea of virtual node in three component technologies, i.e. ProCom, AUTOSAR, and AADL.

**Validity:** We explain how to integrate it in only three component models. We
cannot claim that the idea of virtual node is applicable in general.

Research question Q3 has a broad scope. We implement HSF in a real-time operating system FreeRTOS; the details are described in Paper B and Paper C. To have an efficient implementation of hierarchical scheduling (less overhead of hierarchical scheduling) and to get better utilization of the system, a number of design considerations are made as explained in Sections 6.4.6 and 7.4.4 and are addressed in Sections 6.5.3 and 7.5.4 respectively. We check our implementation during heavy-load and over-load situations for correctness and efficiency. It is clear from the results of the overhead measurements (of tick handler, global scheduler, and task context-switch) that the design decisions made and the implementations are very efficient.

Validity: We test and validate the implementation by experimental results. Since other existing HSF implementations are either using Linux, VxWorks, or \( \mu \)C/OS-II (using simulator for results), our results are difficult to compare to them. We infer the efficiency of our results on the design decisions and on the implementation done. In this work we have not tried to evaluate different types of HSFs. For that reason we have implemented only two-level fixed-priority scheduling and one resource locking protocol (HSRP).

Paper D is a proof-of-concept paper for the realization of our idea of virtual node in the ProCom component model. It addresses Q4, Q5 by performing an example case study and visualizing the execution traces. We test the system for fault containments, predictability in component’s integration and reuse of components in a new environment. Temporal isolation and thus predictable integration has become possible because of the hierarchical scheduling and two-level deployment process. In the experiments, the task sets of each runnable virtual node is executed within the specified budget in a server and is scheduled by a local scheduler. The experimental results manifest that as long as the allocated budgets to servers (at the modeling level) are provided, the timing properties are guaranteed at the execution. Additionally it also provides fault containments (i.e. temporal errors are contained within the faulty runnable virtual node only and their effects are not propagated to the other nodes in the system). All these properties contribute to the predictability of runnable virtual nodes. The increased predictability during components integration further results in making the runnable virtual nodes a reusable entity as obvious from the results presented in Sections 3.4 (briefly) and 8.5.3 (in details) and Figures 3.4 and 3.5.

Validity: We realize our idea in only ProCom component model. Another limitation is the execution of an example case study (instead of a larger industrial one) due to the immaturity of the PRIDE tool.
4.3 Future Research Directions

This thesis work brings possibilities/issues to conduct further research in certain areas that are not thoroughly addressed and could be interesting to investigate in future. Some of these possibilities could be:

Starting from general issues, in this work we realize our concept of runnable virtual node in the ProCom component model using the PROGRESS Integrated Development Environment (PRIDE) tool by means of a proof-of-concept case study. Currently the PRIDE tool is evolving and the automatic synthesis part is not fully mature. It could be interesting to do some more work on this part and then conduct a larger industrial case study on it.

It could be interesting to support virtual communication-busses using server-based scheduling techniques for e.g. CAN [31] and Ethernet [32] in PRIDE tool. This will allow development, integration and reuse of distributed components using a set of virtual nodes and buses.

We believe that our concept is applicable also to commercial component technologies like AADL, AUTOSAR and it could be interesting to realize the concept in those component technologies too.

In the context of hierarchical scheduling, we assume that a system is executed in a single processor while many real-time applications are executed in a multi-processor or multi-core architecture. It could be interesting to extend the HSF implementation for the multi-processor systems. Further, we only worked on temporal isolation (CPU time division) among different sub-systems in our HSF implementation. Considering the memory isolation issues in the implementation could be another interesting direction.

Another interesting direction could be to make the hierarchical scheduling adaptive in nature by implementing mode switches into the hierarchical scheduling. We would like to start from adapting the CPU time and after that working on the memory issues.

Some of the possibilities to be investigated in future that are specific for each paper are presented in the papers.

Bibliography


4.3 Future Research Directions

This thesis work brings possibilities/issues to conduct further research in certain areas that are not thoroughly addressed and could be interesting to investigate in future. Some of these possibilities could be:

Starting from general issues, in this work we realize our concept of runnable virtual node in the ProCom component model using the PROGRESS Integrated Development Environment (PRIDE) tool by means of a proof-of-concept case study. Currently the PRIDE tool is evolving and the automatic synthesis part is not fully mature. It could be interesting to do some more work on this part and then conduct a larger industrial case study on it.

It could be interesting to support virtual communication-busses using server-based scheduling techniques for e.g. CAN [31] and Ethernet [32] in PRIDE tool. This will allow development, integration and reuse of distributed components using a set of virtual nodes and buses.

We believe that our concept is applicable also to commercial component technologies like AADL, AUTOSAR and it could be interesting to realize the concept in those component technologies too.

In the context of hierarchical scheduling, we assume that a system is executed in a single processor while many real-time applications are executed in a multi-processor or multi-core architecture. It could be interesting to extend the HSF implementation for the multi-processor systems. Further, we only worked on temporal isolation (CPU time division) among different sub-systems in our HSF implementation. Considering the memory isolation issues in the implementation could be another interesting direction.

Another interesting direction could be to make the hierarchical scheduling adaptive in nature by implementing mode switches into the hierarchical scheduling. We would like to start from adapting the CPU time and after that working on the memory issues.

Some of the possibilities to be investigated in future that are specific for each paper are presented in the papers.

Bibliography


