A hierarchical approach for reconfigurable and adaptive embedded systems*

Moris Behnam, Thomas Nolte, Insik Shin
Mälardalen Real-Time Research Centre
Mälardalen University, Västerås, SWEDEN

Abstract

Adaptive and reconfigurable embedded systems have been gaining an increasing interest in the past year from both academics and industry. This paper presents our work on hierarchical scheduling frameworks (HSF) intended as a backbone architecture facilitating the implementation of operating system support for adaptability and reconfigurability.

1 Introduction

The work presented in this paper is motivated by the needs of adaptability and reconfigurability in multiple embedded systems domains. In this paper we target mainly the automotive domain; however the approach is also suitable to other application domains such as robotics. We present our work based on the hierarchical scheduling framework (HSF) [15, 16] as backbone architecture for applications with high requirements on adaptation and reconfiguration.

Automotive software systems are traditionally rather static in terms of their provided functionality, how they are configured, and where they are physically located. However, recent trends indicate an increased interest towards dynamic automotive systems [2]. Due to high requirements on safety, predesigned modes are created to cover for all possible usage and failure scenarios. At the same time, cost, weight and complexity motivated trends investigate the possibility of integrating more and more software on fewer electronic control units (ECU) [17], which, in turn, potentially increases the risk of single point of failure scenarios. Having fewer ECUs means that failure of one ECU has the potential to bring several functions down, functions that used to be distributed over multiple ECUs.

For safety critical systems, in the case of failure, certain functionality must always be provided. Firstly, if a failure occurs while driving the car it must be possible to bring the car to a safe state. A scenario of a car being uncontrollable due to a software failure would be unacceptable. Secondly, if possible, it is desirable if the car under failure provides a limited limp back home functionality, i.e., a set of functions allowing the driver to bring the car to repair, even if parts of the system have failed.

From an adaptability and configurability point of view, if one part of the system fails due to, e.g., an accident/crash with the car or due to some internal failure, functionality can be migrated to other nodes, bringing the car to a safe state. This can be provided by either static predesigned modes or by a more dynamic adaptation and reconfiguration functionality with the possibility of coping with more complex scenarios.

Robotics is another targeted domain, and we distinguish robotics used for automation from robotics used in the field.

Robotics used for automation is typically found along assembly lines, and they have high requirements on uptime. If an assembly line would be stopped, this can cause large costs to the manufacturer. In other words, changing, maintaining and adding functionality to the assembly line should not require the whole system to be stalled. The system should provide the possibility of online reconfiguration, tuning and monitoring, in order to minimize downtime.

Field robotics often relies on sensors in order to react on its (sometimes) dynamic environment. These sensors trigger different functionality and actions to be taken depending on the current situation.

Modes can be designed for specific purposes as a reaction to the robot's environment. For example, a tracing robot searching for a specific target can be in different modes depending on if it is lost or if it knows where it is going. Also, the trigger of specific sensors might trigger a more sensitive motion control, whereas normal behaviour would be less sensitive.

These modes can be offline designed, in the case when all possible modes can be predicted and managed beforehand. Dealing with more complex scenarios, the design can be more dynamic as a result of using online modes.

In summary, looking at the abovementioned application domains, there is a great potential for protocols, mechanism and architectures providing adaptability and reconfigurability as a first class citizen.

*The work in this paper is supported by the Swedish Foundation for Strategic Research (SSF), via the research programme PROGRESS.

In the following sections, the HSF is presented and how it provides operating system support for adaptability including policies and algorithms for resource reconfiguration. Finally, our ongoing work on admission control functions is presented, and the paper is concluded.

2 The hierarchical scheduling framework

2.1 What is HSF?

The hierarchical scheduling framework (HSF) has been introduced to support hierarchical resource sharing among applications under different scheduling services. The hierarchical scheduling framework can be generally represented as a tree of nodes, where each node represents an application with its own scheduler for scheduling internal workloads (e.g., threads), and resources are allocated from a parent node to its children nodes.

From a general point of view, it is desirable that a hierarchical scheduling framework can support the following properties; (1) *independency*: i.e., that the fulfilment of temporal requirements (schedulability) of a subsystem can be analyzed independently of other subsystems as there will be no unpredictable interference among subsystems. (2) *abstraction*: i.e., that a subsystem imposes minimal temporal requirements on its environments in order to guarantee functional and extra-functional correctness. (3) *universality*: i.e., that any scheduler can be used within a subsystem, allowing for the most appropriate scheduler to be used for a specific function. (4) *flexibility*: i.e., it should enable adaptation and reconfiguration of its subsystems, implementing operating system support for adaptability including policies and algorithms for resource reconfiguration.

2.2 What are the benefits?

The HSF has been constructed with *modularity* as a main criterion. Component based design has been widely accepted as a methodology for designing and developing complex systems through systematic abstraction and composition. The HSF provides means for decomposing a complex system into well-defined parts, called *subsystems*, and for *interfaces* specifying the relevant properties of these subsystems precisely, such that subsystems can be independently developed and assembled in different environments. The HSF provides a means for composing subsystems into a subsystem assembly, or *composite*, according to the properties specified by their interfaces, facilitating the reuse of subsystems. A challenging problem in composing subsystems is to support the principle of *composability* such that properties established at the subsystem level also hold at the composite level. The HSF can be effectively useful in supporting composability on timing properties in the design and analysis of real-time systems, since it allows the system-level timing property to be established by combining the subsystem-level timing properties (specified by individual subsystem interfaces).

The HSF can be used to support multiple applications while guaranteeing independent execution of those applications. This can be correctly achieved when the system provides *partitioning*, where the applications may be separated functionally for fault containment and for compositional verification, validation and certification. The HSF provides such a partitioning, preventing one partitioned function from causing a failure of another partitioned function in the time domain.

The HSF is particularly useful in the domain of open environments, where applications may be developed and validated independently in different environments. For example, the HSF allows an application to be developed with its own scheduling algorithm internal to the application and then later included in a system that has a different meta-level scheduler for scheduling applications.

2.3 Enabling adaptability and reconfigurability

The HSF is very useful when it comes to the implementation of operating system support for adaptability and reconfigurability needed in dynamic open systems, where applications (one or more subsystems) may be allowed to join and/or leave the system during runtime. In allowing such functionality, a proper *admission control* (AC) must be provided. Also, the HSF allows for a convenient implementation of quality of service management policies, allowing for a dynamic allocation of resources to subsystems.

2.4 Related work on HSFs

Over the years, there has been a growing attention to HSFs for real-time systems. Since Deng and Liu [6] proposed a two-level HSF for open systems, several studies have followed proposing its schedulability analysis [8, 9]. Various processor reource models, such as bounded-delay [12] and periodic [10, 15], have been proposed for multi-level HSFs, and schedulability analysis techniques have been developed for the proposed processor models [1, 5, 7, 10, 14]. The work in this paper extends [15, 16].

3 Admission control

The admission control (AC) applies one or more algorithms to determine if a new application (consisting of one or multiple subsystems) can be allowed to join the system and start execution (admission) without violating the requirements of the already existing applications (or the requirements of the whole system). The decision of the AC depends on the state of the system resources and the resources required by the new application asking for admission. If there are enough resources available in the system, the application will be admitted; otherwise the application will be rejected.

In general, since the AC uses online algorithms the complexity and overhead of implementing these algorithms should be very low for several reasons, such as maintaining scalability of the AC and minimizing its interference on the system. Hence, one objective in designing the AC concerns keeping the input to these algorithms as simple as possible, e.g., the resource requirement for each individual task could be abstracted to the subsystem level. Another objective concerns minimizing interference between the AC and the system online, making it desirable to perform as much work as possible offline.

3.1 Resources

The *resources* considered by the AC may include, but are not limited to, *CPU* resources, *memory* resources, *network* resource and *energy* resources. Initially, we have been focusing on CPU and network resources, and are now also looking at memory resources.

CPU resources When using the HSF, traditional schedulability algorithms can be used in order to check the CPU resources, e.g., by using the global schedulability test in the HSF [15, 16]. This algorithm depends on the type of system level scheduler used, e.g., EDF, FPS, etc. The AC checks the schedulability condition of the system including the new subsystem. If the system is still schedulable, the new subsystem will pass this test; otherwise the new application will be rejected. In using this test, it is guaranteed that all hard real time requirements will be met. The input to the algorithm is the subsystem interface (subsystem budget and period) of each running subsystem together with the interface of the new subsystem. Note that these parameters are evaluated and determined during the development of the subsystem (offline).

Memory resources When allowing for a new application to enter the system, the AC should guarantee that there is sufficient memory space to be used by all subsystems. Otherwise, unexpected problems may happen during run time. In a similar way as for CPU resources, the maximum memory space required by each subsystem is evaluated during its development. In the AC test, a simple algorithm can be used to check if there is enough memory space available in the system, by checking if the summation of the maximum memory space for all subsystems is less than or equal to the memory space provided by the platform. Such an algorithm is very simple; however, the accuracy of the result is not high as all applications will not likely need their specified maximum memory space at the same time. Higher efficiency can be achieved by the usage of algorithms such as the approximated algorithm presented in [4].

Energy resources Most of the modern processors support changing the frequency and voltage of the CPU during runtime, in controlling the CPU's power consumption. The HSF can use this feature to select the lowest frequency/voltage that guarantees the hard real time requirements of the system. Decreasing the frequency of a processor will increase the worst-case execution time (WCET) of its tasks. In doing this, more CPU resources should be allocated to subsystems in order to ensure that all hard real time tasks will meet their deadlines. Looking at the HSF, if predefined levels of frequencies are used, we can find a subsystem interface for each frequency level for all subsystems. Then, during runtime, the AC will make sure that the processor is working with the lowest frequency keeping the schedulability of the current set of subsystems. When it is required to add a new subsystem, the AC will check the schedulability condition with the current processor frequency; if the system is deemed not schedulable, then the AC will try with higher frequencies. When a subsystem is removed from the system, the AC will try to reduce the frequency of the CPU in order to reduce its power consumption.

Network resources This type of resource is important in distributed systems where there typically exist communications between nodes in the network. The network resource is different from the other resources previously described in the sense that the network resource is shared by all nodes, while the other resources are local to each node. When the AC is faced with a request for adding a subsystem, it should check if the communications requirements will be met, i.e., check if all important messages will be delivered in proper time [13]. Selecting an algorithm that checks this resource is more complex as there are many different requirements, communication protocols, network types, etc. Covering all these aspects might not be necessary but as an illustration consider a simple algorithm which relies on the communication bandwidth. During the development of each subsystem, their maximum communication bandwidth requirements should be evaluated such that the AC can use it in order to check if the summation of required bandwidth for all subsystems is less than 100%.

3.2 Admission control in distributed systems

Implementing the AC in distributed systems is more complex than doing so for a single CPU. The main reason for this is that the information needed by the AC algorithms must be consistent. For example, when using the network resources, awareness of all network users must be maintained by the AC, and these users are typically located on many nodes throughout the distributed system. Commonly, information on the current state is kept at one place, managing the information needed by the AC. Also, when an application consists of more than one subsystem, and these subsystems are located at different nodes, all these subsystems should pass the AC tests before admitting the application.

In designing the AC we have identified 3 different approaches based on where the AC test will be implemented.

- A special **Master Node (MA)** will implement the AC tests of all resources in the system. Only the MA will have information about resources in the system. Hence, consistency is not a problem, it is easy to determine the order between AC requests, and the AC does not have to contact multiple nodes in getting the current system state as only a single AC request to the MA is needed. On the downside the MA is a single system level point of failure.

- **All Nodes (AN)** will implement the AC tests of all resources in the system. Each node should have the consistent information of all resources that are used by the system. Hence, in this fully distributed approach the AC test can be performed without having to communicate with other nodes. Also, the approach is tolerant to failures. On the downside, consistency must be maintained between all nodes, and ordering is more complex compared with MA. Also, more memory is needed in maintaining all resource state replicas.

- There will be **One Node (ON)** implementing the AC test of each resource in the system. Each node will maintain information about the resources that is responsible for. Hence, there will be no issues with respect to data consistencies between replicas of the same information, but a single AC request might have to communicate with a number of nodes in order to get a valid system state. Also, ordering among AC requests must be solved, and each resource owner will be a single resource level point of failure.

As a first step, we consider the MA approach that we believe possess strong properties in terms of flexibility, promoting the evaluation of multiple algorithms. Also, this approach fits very well with the HSF.

4 Summary

To summarise we are currently active in 3 areas related to the motivation of this paper. Firstly, we have done work in the area of synchronization algorithms for HSFs [3]. When multiple subsystems are sharing the same CPU it is likely that they also share logical resources that must be protected by the usage of a proper synchronization protocol. Secondly, we are implementing the HSF on a commercial operating system on an application given by one of our industrial partners. This implementation will have an important role in evaluating the efficiency of the HSF itself, as well as the AC approaches presented in this paper. Thirdly, we are designing the admission control and quality of service manager. These will also be implemented in the HSF implementation. In summary we believe that the result of these three areas will provide important knowledge towards adaptive and reconfigurable systems; results that have both high industrial relevance as well as academic relevance.

Acknowledgements

The authors wish to express their gratitude to the anonymous reviewers for their helpful comments.

References

[1] L. Almeida and P. Pedreiras. Scheduling within temporal partitions: response-time analysis and server design. In *EMSOFT '04*, 2004.

[2] R. Anthony, A. Leonhardi, C. Ekelin, D. Chen, M. Törngren, G. de Boer, I. Jahnich, S. Burton, O. Redell, A. Weber, and V. Vollmer. A future dynamically reconfigurable automotive software system. In *Elektronik im Kraftfahrzeug*, June 2007.

[3] M. Behnam, I. Shin, T. Nolte, and M. Nolin. Sirap: A synchronization protocol for hierarchical resource sharing in real-time open systems. In *EMSOFT'07*, 2007.

[4] M. Bohlin, K. Hänninen, J. Mäki-Turja, J. Carlson, and M. Nolin. Safe shared stack bounds in systems with offsets and precedences. Technical Report, Mälardalen University, January 2008.

[5] R. I. Davis and A. Burns. Hierarchical fixed priority preemptive scheduling. In *RTSS*, 2005.

[6] Z. Deng and J. W.-S. Liu. Scheduling real-time applications in an open environment. In *RTSS '97*, 1997.

[7] X. A. Feng and A. K. Mok. A model of hierarchical real-time virtual resources. In *RTSS*, 2002.

[8] T.-W. Kuo and C.-H. Li. A fixed-priority-driven open environment for real-time applications. In *RTSS*, 1999.

[9] G. Lipari and S. Baruah. Efficient scheduling of real-time multi-task applications in dynamic systems. In *RTAS '00*, 2000.

[10] G. Lipari and E. Bini. Resource partitioning among real-time applications. In *ECRTS*, 2003.

[11] G. Lipari, J. Carpenter, and S. Baruah. A framework for achieving inter-application isolation in multiprogrammed hard-real-time environments. In *RTSS '00*, 2000.

[12] A. Mok, X. Feng, and D. Chen. Resource partition for real-time systems. In *RTAS '01*, 2001.

[13] T. Nolte. *Share-Driven Scheduling of Embedded Networks*. PhD thesis, Department of Computer and Science and Electronics, Mälardalen University, Sweden, May 2006.

[14] S. Saewong, R. R. Rajkumar, J. P. Lehoczky, and M. H. Klein. Analysis of hierar hical fixed-priority scheduling. In *ECRTS '02*, 2002.

[15] I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In *RTSS '03*, 2003.

[16] I. Shin and I. Lee. Compositional real-time scheduling framework. In *RTSS '04*, 2004.

[17] G. Spiegelberg. The impact of new gateways and busses - are these the answers for furhter innovations?, Podiums Discussion at the Embedded Systems Week, Salzburg, Austria, October 2007.