Attaining Flexible Real-Time Systems by Bringing Together Component Technologies and Real-Time Systems Theory

Johan Fredriksson, Mikael Åkerholm, Kristian Sandström and Radu Dobrin
Department of Computer Science and Engineering, Mälardalen University, Västerås, Sweden
{johan.fredriksson, mikael.akerholm, kristian.sandstrom, radu.dobrin}@mdh.se

Abstract

In this paper we propose a component model and run-time mechanisms, gathering benefits provided by both Real Time Systems (RTS) and Component Based Software Engineering (CBSE). In particular, we show that the proposed model is a suitable package for efficient utilization of the multiple version paradigm. The purpose of using a multiple version technique is to ensure a minimum level of quality while providing run-time flexibility.

1 Introduction

Computer control systems are embedded in a large and growing group of products such as automotive vehicles, aircrafts, and industrial robots. A reason for establishing a CBSE discipline for such systems is that the systems are becoming increasingly more complex due to the inclusion of more functionality. At the same time the product cycles are becoming shorter, leading to requirements of shorter time to market. Moreover, the industry strives to enable cost effective implementation of new functionality. Although the motivation for utilizing CBSE methods for development of embedded control systems are essentially the same as for general purpose software, the requirements on them differ. For embedded control systems a component model must focus on extra-functional requirements, such as reliability, timing, resource usage and linkage to specific hardware.

This paper propose a component model that adopt *The Multiple Versions Paradigm* [11], which is a method that can be used to achieve more efficient resource usage. The Multiple Versions Paradigm uses different versions of implementations for solving a problem. This could involve different implementations of a task monitoring varying numbers of parameters, more or less deep iterations in numerical approximations (imprecise computing)within different versions. The proposed component model is based on the model described in [13]. Their work defines a *Prediction Enabled Component Technology* (PECT) [6].

The contribution of this work is the introduction of a component model with real-time attributes, which is a suitable package for utilisation of the multiple version paradigm. To use the multiple version paradigm, *the Adaptive Threshold Algorithm* [4] is adopted.

A second contribution is an adaptation of the Adaptive Threshold Algorithm to also handle aperiodic tasks.

The rest of the paper is outlined as follows. In section 2 we present real-time basics. In section 3 the component model is presented. The paper proceeds in section 4 by describing the run-time mechanisms. The 5th and final section concludes the paper and contains suggestions for future work.

2 Real-Time Systems

Real-time systems are computer systems in which the correctness of the system depends not only on the logical correctness of the computations performed, but also on time factors [10]. What is common for most real-time tasks is the *Worst Case Execution Time* (WCET), which has to be calculated in order to make predictions about the system behaviour, e.g., to guarantee timing requirements.

Real-time systems can be classified into two major categories: *hard* and *soft* real-time systems, depending on the consequences of a *deadline* miss. The deadline is derived from the latest point in time when a response to an event must be generated. Hard real-time systems are computer systems in which all task deadlines must be met. On the other hand, in soft real-time systems, a number of deadlines can be missed without serious consequences. This paper primarily focuses on hard real-time systems.

The choice of scheduling technique used in order to achieve different requirements has been well analyzed and discussed [14]. If the main goal is to achieve run-time flexibility, the approach typically used is priority driven scheduling. Priority driven (on-line) scheduling can be divided in two main categories: *Fixed Priority Scheduling* (FPS) or *dynamic priority scheduling* such as *Earliest Deadline First* (EDF). Common for both categories is that the scheduling decision for individual tasks is made during run-time, based on the priority of the tasks. This results in a flexible system with a potentially higher ability to cope with run-time events. In fixed priority based systems, guarantees for temporal behaviour are achieved by performing response time analysis [1], [2],[10]. In dynamic priority based systems, e.g., EDF, the guarantee that all tasks will meet their deadlines is based on the processor utilization [8].

3 Component Model

This section briefly outlines the proposed component model, by defining a single component and how to compose components. Below is some of the basic component properties listed.

- A basic property of a component is that the implementation is not reachable by a third party; a component is a pre-compiled black-box. The only way to communicate with a component is through its interfaces.
- Components can be composed from other components, denoted *composed components* or *composites*.
- A service provided by a component can be implemented in different versions. As in the Multiple Version Paradigm, the different versions are denoted *quality levels*. Each quality level is assigned a value.
- A service provided by a component can be *active*, or *passive*. An active service is scheduled by the run-time system, while a passive service is not directly in contact with the run-time mechanisms.

3.1 Component Description

In Figure 1, a UML meta-model of a component is shown. A component provides one or more services, which in some sense are related. As *Port Based Objects* (PBO) [12], the services provide in and out ports for exchanging data with each others. An active service has an associated descriptor containing all parameters

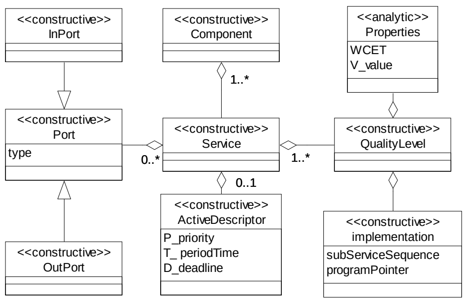

Figure 1. UML meta-model of a component

needed by the run-time system. Furthermore, a service has one or more quality levels and the number of provided quality levels is free for the developer to specify. However, at least one quality level is required. Each quality level has an implementation. A function pointer represents the implementation of the quality level for a basic component, but a quality level provided by a composite has a sequence of sub-services that should be executed upon an invocation. A quality level also has a WCET and a value associated with itself. Theories regarding WCET estimation have been presented, in e.g., [5],[9]. Another analytic property is the value V that can be set to an arbitrary number, and is used by the run-time system to choose a quality level. The run-time system tries to schedule the service with highest value, the run-time treatment of the value is introduced in formula (3), section 4.1.

A component has three interfaces, which are the specification of its access points, as in [3]. *Data interfaces*, are port based, and contain information about existing ports and data type definitions. *Control interfaces* provide methods for invocation of the different encapsulated services. If a service is defined as active, it consists of parameters and structures required by the run-time system. *Analytic interfaces* provide parameters concerning different quality levels of a service. Here the number of levels with corresponding WCET and value are included, but the analytical interface can be extended with other attributes, e.g., memory consumption.

3.2 Assembling Components

Assembling components or parts of components into composed components, and assembling systems through exchanging data between components, is carried out by utilizing different interfaces. Hence, data exchange and component composition are independent to each others. Therefore, data can cross component boundaries utilizing the same interfaces as within a component.

Exchanging data between services is carried out by

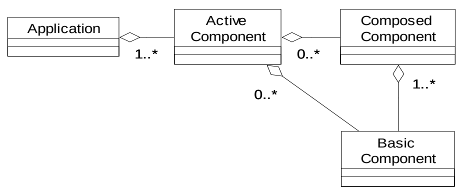

Figure 2. Components building an application

connecting in ports and out ports. A smallest requirement on the type level is that the interacting in and out ports uses the same data types. The ports of a particular service is accessible trough the data interface provided by the component hosting the service. When a service is launched, it begins with reading data from all its in ports (zero or more) and, when its execution is finished, data is written to its out ports (zero or more). Composing a service within a component from services provided by other components is achieved through the definition of a sequence. The sequence is constructed by using the control interface provided by the subcomponents. Assembly of applications may be viewed as a hierarchy. Basically, as in Figure 2, we can distinguish between, basic components, composed components or composites, active components and applications.

4. Component Technology

In the model proposed in section 3, a service has a number of quality levels. One of the levels is a *basic level* that has been analysed pre-runtime by a schedulability test. In our approach, a schedulability analysis is performed in order to guarantee the basic level for each periodic service. There are many ways of formally guaranteeing that a set of services will complete before their deadlines [1],[2],[8].

4.1 On-Line Service Scheduler

The Adaptive Threshold Algorithm is modified and applied to the component model. We also define how to handle run-time events with incompletely known parameters, i.e., aperiodic services.

For executing components at a higher quality level during runtime, the Adaptive Threshold algorithm adopts two techniques, *resource reclaiming* and *Slack-stealing*[7]. In some cases it is of greater value to the system to execute an aperiodic service, rather than executing a periodic service at a higher quality level. However, all the periodic services have to be guaranteed to complete their basic quality level before their deadlines. Terms that will be used in the following sections are listed below.

Residual time L_l is the difference in time between the estimated WCET and *Actual Case Execution Time* (ACET) of a quality level l of a service. Using this residual time is called resource reclaiming.

Slack time S_l represents slack time for a quality level l. The slack is the extra interference that a service can be subjected to without missing its deadline. This can be performed with the slackstealing algorithm.

Spare time is the smallest of the residual and the slack time. It is the maximum time that any lower prioritized service can be subjected to without missing its next deadline.

Available time $A_s(t)$ is the spare time together with the scheduled time for the specified service.

Each quality level l, in a service s has a worst-case execution time $WCET_s^l$ and a value V_s^l which is set pre runtime. The service incorporates a deadline D_s, and a release time R_s. The basic quality level of a service is indexed with P ($WCET_s^P$). The values of aperiodic services are decided the same way as the periodic services.

In [4], an expression (1) for calculating the available time is proposed. $L_l(t, D_s)$ is the residual time available at priority level l in the interval $[t, t + D_s)$. $min_{\forall j \in lp(l)} S_j(t)$ is the extra interference that any lower prioritized service can be subjected to without missing its next deadline, assuming it will use its basic quality level.

$$A_s(t) = WCET_s^p + min[L_l(t, D_s), min_{\forall j \in lp(l)} S_j(t)] \quad (1)$$

The run-time system must decide if there is enough available time for a specific quality level before it is scheduled. The system must also determine if the aperiodic service should be run at all. Since the periodic services must be guaranteed to run, the aperiodic services are not guaranteed pre run-time. Hence, the available time for the aperiodic services is not the same as for the periodic, thus (2).

$$A_s(t) = min[L_l(t, D_s), min_{\forall j \in lp(l)} S_j(t)] \quad (2)$$

As Figure 3 shows, at time t, the time interval available for executing a medium priority service (MP) is L_l. That is the additional execution time, i.e., the time not used by any service. The reclaimed time can be used for any service ready to execute at time t. However, the maximum amount of time that can be used by MP is until its deadline D_{MP}.

In addition to the values V_s^l corresponding to each service, there is also a global system value V^{SYS}. The V^{SYS} value is the mean of all executed periodic services. In [4], a value-density based strategy has been

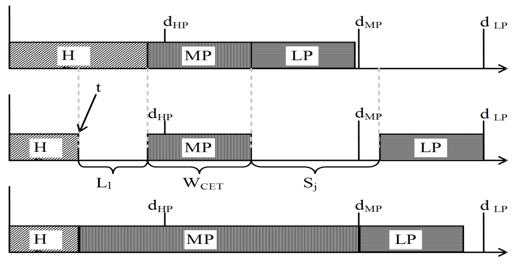

Figure 3. Visual explanation of Formula 1

proposed. The strategy chooses the quality level with the highest value density W_s^l. The value-density W_s^l is given by (3). This approach also applies to aperiodic services since V^{SYS} is the mean value of all periodic services. Hence it is easy to determine if the system accrues a higher value by scheduling the aperiodic service.

$$\forall l[WCET_s^l \leq A_s(t)]:$$
$$W_s^l = \frac{V_s^l \cdot WCET_s^l + [A_s(t) - WCET_s^l] \cdot V^{SYS}}{A_s(t)} \quad (3)$$

However, an interesting discussion is which services should contribute to the V^{SYS}. Should it be only periodic services, or aperiodic services as well. In a system where aperiodic services are rare and come in bursts, the V^{SYS} may become very unstable and fluctuating, perhaps misguiding the system. However, if aperiodic services come regularly and with some even distribution, they should very much be taken into account to promote high value aperiodic services before higher quality levels of periodic services.

5. Conclusions and Future Work

In this paper we have shown that a real-time component is a suitable package for the multiple versions paradigm. We have proposed a component model with a run-time mechanism that gives the developer possibilities for issuing real-time guarantees with additional flexibility through implementing multiple versions.

Furthermore, the Adaptive Threshold Algorithm is used and adapted to also cater for aperiodic services. This is important in order to provide flexibility in many real-time systems.

A prototype implementation of the proposal with development tools and possibility to compile for execution upon some commercial real-time operating system is the next step towards a realization of the model. Trying to utilize such a prototype in the development of an embedded control system, would result in useful input for future development of the component model.

Acknowledgements We would like to thank all anonymous reviewers for constructive and helpful comments. Special thanks to Jukka Mäki-Turja and Tomas Lennvall for peer reviewing this paper.

References

[1] N. C. Audsley. Optimal Priority Assignment and Feasibility of Static Priority Tasks with Arbitrary Start Times. Technical report, Department of Computer Science, University of York, 1991.

[2] N. C. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings. Applying New Scheduling theory to Static Priority Pre-Emptive Scheduling. In *Software Engineering Journal*, pages 284–292, 1993.

[3] I. Crnkovic and M. Larsson. *Building Reliable Component-based Software Systems*. Number ISBN 1-58053-327-2. Artech House computing libraray, 2002.

[4] R. Davis, S. Punnekkat, N. Audsley, and A. Burns. Flexible Scheduling for Adaptable Real-Time Systems. *1080-1812/95 IEEE*, 1995.

[5] J. Engblom, A. Ermedahl, M. Sjödin, J. Gustafsson, and H. Hansson. Execution-Time Analysis for Embedded Real-Time Systems. In *In proceedings of Software Tools for Technology Transfer*, 2000.

[6] S. A. Hissam, G. A. Moreno, J. Stafford, and K. C. Wallnau. Packaging Predictable Assembly with Prediction-Enabled Component Technology. Technical Report CMU/SEI-2001-TR-024 ESC-TR-2001-024, Software Engineering Institute, Caregie Mellon University, 2001.

[7] J. P. Lehoczky and S. Ramos-Thuel. An Optimal Algorith for Scheduling Soft-Aperiodic Tasks Fixed Priority Preemptive Systems. In *Proceedings Real-Time System Symposium*, pages 110–123, 1992.

[8] C. I. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM*, 20(1), 1973.

[9] P. Puschner. A Tool for High-Level Language Analysis of Worst-Case Execution Times. In *Proceedings of 10th Euromicro Workshop on Real-Time Systems*, pages 130 –137, 1998.

[10] L. Sha, R. Rajkumar, and J. Lehoczky. Task Period Selection and Schedulability in Real-Time Systems. *IEEE Transactions on Computer*, 39(9), 1990.

[11] J. A. Stankovic and K. Ramamritham. What is Predictability for Real-Time Systems? *Real-Time Systems*, 2(4), 1990.

[12] D. B. Stewart, R. A. Volpe, and P. Khosla. Design of Dynamically Reconfigurable Real-Time Software using Port-Based Objects. *IEEE Transactions on Software Engineering*, 23(12), 1997.

[13] A. Wall, M. Larsson, and C. Norstrom. Towards an Impact Analysis for Component Based Real-Time Product Line Architectures. In *Proceedings of the 28th Euromicro Conference*, pages 81–88, 2002.

[14] J. Xu and D. L. Parnas. Priority Scheduling versus Pre-run-time Scheduling. *Journal of Real-Time Systems*, 2000.