Ensuring Real-Time Performance Guarantees in Dynamically Reconfigurable Embedded Systems *

Aleksandra Tešanović[1], Mehdi Amirijoo[1], Daniel Nilsson[1], Henrik Norin[1], and Jörgen Hansson[2,1]

[1] Linköping University, Department of Computer Science, Linköping, Sweden
{alete,meham,jorha}@ida.liu.se
[2] Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA
hansson@sei.cmu.edu

Abstract. In this paper we present a quality of service (QoS) adaptive framework for dynamic reconfiguration of component-based real-time systems. Our framework is light-weighted enabling reconfiguration in resource constrained embedded environments. Furthermore, it is possible to reconfigure both components and aspects of a system, hence, enabling finer tuning of a real-time system. Real-time QoS guarantees are maintained in the system and under reconfiguration by employing feedback-based scheduling methods.

1 Introduction

A large majority of computational activities in modern society are performed within embedded and real-time systems. For a real-time system, it is essential that results produced by the system are both produced correctly and in a timely manner. To ensure timeliness, tasks[3] in a real-time system are associated with deadlines, and a number of real-time scheduling techniques have been developed ensuring that tasks meet their respective deadlines (see [1]).

Successful deployment of real-time and embedded systems strongly depends on low development costs, a short time to market, and a high degree of reconfigurability. One way to meet these requirements is to adopt the component-based software development (CBSD) paradigm for real-time systems. This way systems are developed out of pre-defined software components to fit a specific real-time application. Component-based real-time system approaches, e.g., [2–4], do not provide efficient support for crosscutting features such as concurrency control or scheduling algorithms that typically crosscut the structure of the overall system. Aspect-oriented software development (AOSD) [5] has emerged as a new principle for software development that provides an efficient way of modularizing crosscutting concerns in software systems in "modules", called aspects. Using AOSD, systems can be built to contain only the required functionality, while

* This work is supported by the Swedish Foundation for Strategic Research (SSF), the Swedish National Graduate School in Computer Science (CUGS), and the Center for Industrial Information Technology (CENIIT) under contract 01.07.

[3] Real-time systems are typically constructed out of concurrent programs called tasks.

other functional and non-functional crosscutting features encapsulated into aspects can be added to the system in a process called aspect weaving.

In an effort to integrate the two software engineering techniques, namely AOSD and CBSD, into real-time system development we have developed an approach to aspectual component-based real-time system development, ACCORD [6]. Given some system requirements ACCORD enables efficient system configuration from components and aspects from a library.

Most real-time component-based software systems, including our earlier work on ACCORD, are pre-compiled. This implies that the resulting running system is monolithic and not dynamically reconfigurable. Consequently, when updating or maintaining these systems, they need to be shut down for recompilation. However, reconfiguring a system on-line is desirable for embedded real-time systems that require continuous hardware and software upgrades in response to technological advancements, environmental change, or alteration of system goals during system operation [7, 3]. For example, small embedded real-time sensor-based control systems must be designed and developed such that software resources, e.g., controllers and device drivers, change on the fly. Hence, a reconfiguration mechanism, enabling adding removing, and exchanging components on-line, is needed to ensure that the software is updated without interrupting the execution of the system.

However, dynamic reconfiguration of a real-time system changes the temporal behavior of the system. For example, exchanging a component for a new version could result in a task execution time that is higher than the execution time obtained when using the existing component in the system. This in turn influences the real-time performance of the system negatively. Hence, the dynamic reconfiguration mechanism should be capable of ensuring that the system satisfies real-time performance guarantees, in terms of quality of service (QoS).

In this paper we present a QoS-adaptive framework for dynamic reconfiguration of real-time systems. The QoS-adaptive framework is an extension of the previous work on ACCORD to support exchange of components and aspects during run-time while preserving real-time performance guarantees. The main benefits introduced by this framework are as follows.

- The framework enables dynamic reconfiguration of real-time systems in embedded environments. We have identified a list of requirements that dynamically reconfigurable real-time and embedded systems need to satisfy, and we show that our framework fulfills these requirements.
- The framework is novel in that it supports dynamic reconfiguration in two dimensions: reconfiguration of components and reconfiguration of aspects. Dynamic reconfiguration of both components and aspects provides higher reusability and flexibility of a real-time system.
- The framework ensures that QoS guarantees are maintained even under reconfiguration. By employing feedback control structure we ensure that a real-time system adapts to the changes in the temporal behavior of components which occur during dynamic reconfiguration.

The paper is organized as follows. We formulate the problem in section 2. Our QoS-adaptive framework for dynamic reconfiguration is presented in section 3. The paper finishes with main conclusions in section 4.

2 Problem Formulation

In this section we identify a number of requirements that the reconfiguration framework needs to fulfill to ensure dynamic reconfigurability in a real-time system and discuss to what degree these requirements are fulfilled by existing approaches.

R1 A reconfiguration framework of a real-time system should be light-weight, implying that in normal operation of the system this framework must not introduce any significant overhead in the task execution and memory consumption of the system. This is to ensure that the system is usable in an embedded and real-time environment with sparse recourses in terms of CPU and memory.

R2 There should not be a restriction on the number of components that could be exchanged or added/removed from the system since reconfiguration may affect the entire system.

R3 Reconfiguration may be requested at any time, and the system has no a priori knowledge of the possible components that are to be reconfigured in the system.

R4 Reconfiguration must be carried out efficiently in terms of the granularity of exchangeable parts, implying that flexibility for fine-tuning of an embedded real-time application should be ensured. One should be able to exchange both functionality that is encapsulated into components and real-time algorithms that typically crosscut several components.

Accurate temporal characteristics of software components that are being added, removed, or exchanged in the system are not always available. Moreover, if temporal characteristics are available, the problem of determining exact temporal characteristics of software on a target platform is known to be difficult due to, e.g., cache and branch prediction [8]. Therefore, we need to ensure that, when reconfiguring the real-time system, the reconfiguration does not affect the performance of the system negatively. That is, a real-time system under reconfiguration should consider the varying temporal behavior of the software components being reconfigured and adapt accordingly. The adaptation should be such that specified performance requirements expressed in terms of a desired QoS, bounded worst-case QoS, and timely adaptation are satisfied. Hence, the following additional requirements on dynamically reconfigurable system arise.

R5 The system user must be able to specify the desired system QoS during steady state, i.e., the state in which no reconfiguration is applied.

R6 In the face of a transient state during which reconfiguration is applied, the worst-case system QoS and temporal adaptation must comply with the specified requirements. More specifically, the QoS must satisfy worst-case

system QoS requirements and the QoS must converge towards and reach the desired QoS within a certain specified time interval (also known as settling time).

The majority of approaches enabling development of configurable real-time systems is concerned with configuration of a real-time system in the design and compilation phase and does not provide flexibility by dynamically reconfiguring the system [9, 4, 3]. Existing approaches [10, 11, 2] enabling dynamic reconfiguration do not fulfill all of the identified requirements. The approach presented in [11] does not satisfy requirement R1, and is not suitable for embedded environments. Approaches that do comply with R1, e.g., [10, 2], do not provide flexible way of reconfiguration, e.g., system reconfiguration is done by pre-compiling possible configurations into the monolithic system, violating requirement R3 [2]. The named approaches to dynamic real-time reconfiguration do not provide means for exchanging algorithms and other crosscutting concerns in the system, and, therefore, are not fulfilling requirement R4. There has been work on maintaining performance guarantees under reconfiguration of real-time systems, addressing requirements R5 and R6, but this has been done assuming that the worst-case temporal behavior of components is known prior to the reconfiguration and/or that the reconfiguration is done off-line [12, 11].

3 Approach

In this section we present our QoS-adaptive framework for dynamic reconfiguration of real-time systems that fulfills identified requirements R1-R6. We first review main characteristic of the ACCORD approach, as they were presented in our previous work. Then we present extensions to the approach that enable dynamic reconfiguration of a real-time system and discuss how QoS guarantees are maintained under reconfiguration.

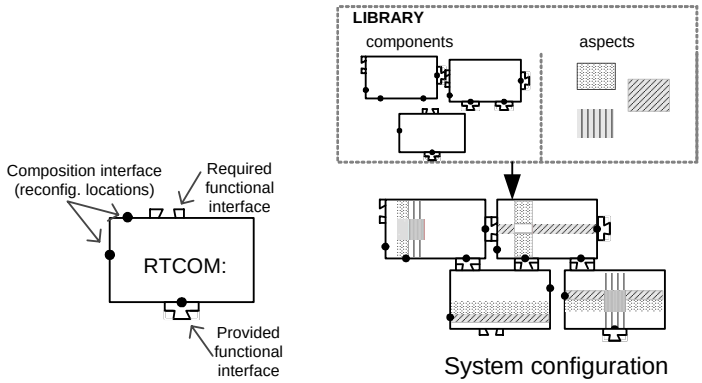
3.1 ACCORD

ACCORD [6] prescribes that real-time systems should first be decomposed into a set of components followed by decomposition into a set of aspects. Aspects are properties of a system affecting its performance or semantics, and crosscutting the functionality of a system.

ACCORD provides a real-time component model, denoted RTCOM, to support reconfigurability [6]. RTCOM components are "grey" as they are encapsulated in interfaces, while changes to their behavior can be performed in a controlled way by aspect weaving. The process of weaving is typically done off-line and it generates the modified source code of a component as specified by aspects. An aspect consist of pointcuts, describing places in the code of components where aspect should be woven, and advices, defining code that is to be woven.

Each RTCOM component has two types of functional interfaces: provided and required (see figure 1(a)). Provided interfaces reflect a set of operations

that a component provides to other components, while required interfaces reflect a set of operations that the component requires from other components. Composition interfaces define reconfiguration locations in the component code. Reconfiguration locations define the points in the component code where additional modification of components can be done by aspect weaving. These points can be used by the component user (or component developer) to reconfigure a component for a specific application. Reconfiguration locations are also used for analysis purposes.



(a) (b)

Fig. 1. (a) Reconfigurable real-time component model (RTCOM), and (b) ACCORD system configuration

Figure 1(b) illustrates the way a system is configured and deployed in the underlying environment. Here, the composition and configuration of the system via aspects and components is done at compile time. The final system deployed in the environment is monolithic and cannot be reconfigured on-line.

3.2 Enabling Dynamic Reconfiguration

To enable dynamic reconfiguration we extend the original ACCORD approach in two ways. Firstly, we extend the RTCOM component model to ensure preservation of component and aspect states during reconfiguration. Secondly, we introduce a middleware layer, denoted ACCORD component framework, which handles communication among components and aspects.

Note that, in the context of ensuring real-time performance, an exchange of a component encompasses all the dimensions of the dynamic reconfiguration as it includes also a removal of the old version of a component and the addition of

a new version. Therefore, in the following, we focus primarily on explaining the component exchange[4] part of the dynamic reconfiguration.

Extensions to RTCOM To be able to preserve the internal state of components and aspects under reconfiguration, RTCOM is extended as follows. The provided interface of a component is extended with two mandatory operations, `export` and `import`. The `export` operation enables a component under reconfiguration to export its state, i.e., to store its state outside its data space (in the ACCORD component framework). The `import` operation ensures that the exported state of a component, which is to be replaced, is correctly imported into a new version of the component.

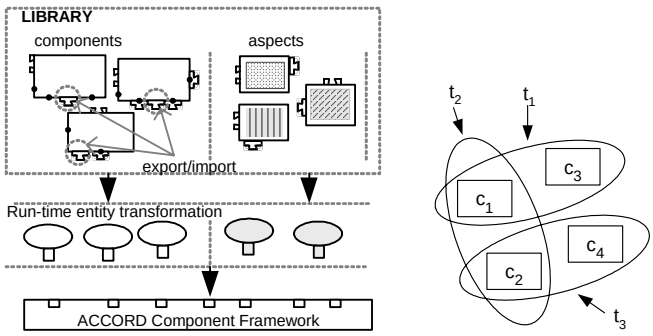
Enabling aspect exchange implies changing the way aspects are implemented within RTCOM. To conform to the dynamic aspect reconfiguration, a feature not supported by current aspect languages, and still enable general use of our framework with any of the aspect and/or component languages, we augment aspects with provided and required interfaces. These interfaces play an important role in communication and reconfiguration as we describe later in this section. Observe also that having interfaces defined for each aspect provides better encapsulation of aspect functionality and still preserves the crosscutting nature of an aspect.

System composition out of components and aspects now consists of the following steps (see figure 2(a)). First, aspects are woven off-line into components they affect. Then, using information stored in interfaces of components and aspects, these are translated into run-time entities (shared objects) recognized by the component framework. These run-time entities are then deployed onto the component framework. Conceptually, this means that weaving and run-time entity generation are done on a separate computer, and the obtained run-time entities are uploaded in the system running on another computer. Having components and aspects mapped into run-time entities that communicate via the component framework enables easy reconfiguration of both components and aspects, satisfying requirement R4.

ACCORD Component Framework The ACCORD component framework is a middleware layer between a run-time environment and the components, which handles communication among components and aspects using jump tables (inspired by [10]). The jump table contains the list of pointers to provided and required interfaces of components and aspects. When exchanging a component or an aspect, the jump table entries for the functional interface of a component or an aspect are re-pointed to the new version of the component/aspect. As mentioned, during reconfiguration the component framework temporarily stores component and aspect internal states, using the `export` and `import` operations.

The component framework provides a user interface for reconfiguration. This reconfiguration user interface consists of operations that enable adding, remov-

[4] A component exchange in the literature is also referred to as a live update of a system.



(a) (b)

Fig. 2. (a) ACCORD system configuration revisited, and (b) an example of the relation between tasks and components

ing, or exchanging components (see figure 3). During the system lifetime, reconfiguration can be requested at any time using these operations, thus, fulfilling requirement R3.

Before allowing reconfiguration to take place, the component framework always makes sure that the system can undergo reconfiguration without interrupting the task execution (line 2 in figures 3(a-c)). Tasks, which rely on operations of components that are to be reconfigured, have to be completed before the reconfiguration is carried out. This is exemplified in figure 2(b), where there are four components denoted by $c_1, \ldots, c_4$, and three tasks denoted by $t_1, \ldots, t_3$. Task t_1 uses operations of component c_1 and c_3, while t_2 uses c_1 and c_2. Both t_1 and t_2 have to complete executing before c_1 can be exchanged. However, only t_1 needs to be completed before c_3 can be exchanged. Once the system can undergo reconfiguration, the actual exchange of components is initiated.

```
1 exchange(c1,c2){
2   makeSystemReady();
3   remove(c1);
4   redirect(c1,c2);
5   add(c2);
6 }
```

(a) exchange

```
1 remove(c){
2   makeSystemReady();
3   if (c.hasState(){
4     state=c.export();
5     statePresent=true;
6   }
7 }
```

(b) remove

```
1 add(c){
2   makeSystemReady();
3   if (statePresent){
4     c.import(state);
5     statePresent=false;
6   }
7 }
```

(c) add

Fig. 3. Reconfiguration of a component

For example, the exchange of an old version of a component c, denoted c^1, with a new version, c^2, is carried out after the system is prepared for exchange, by first removing the component from the system (line 3 in 3(a)). The removal is carried out by exporting the state of (an old version) of a component into the

framework (lines 3-5 in figure 3(a)). Then the jump table is re-pointed to the functional interfaces of the new version, c^2, of the component (line 4 in figure 3(b)). Finally, the new component is added to the system (line 5 in figure 3(a)). The operation `add` restores states in the new version of the component using the `import` operation (line 4 in figure 3(c)). The role of the `statePresent` variable in `remove` and `add` operations is to ensure that the state of a component is restored only if it is needed. Described reconfiguration mechanism is hidden in the component framework and it enables fast and light-weight reconfiguration, satisfying requirement R1.

The reconfiguration of any number of components can be done by a user, which can either be the system user (human) or an application (requirement R2). The component framework is aware of the version of a component, via the version number of a component, in the current configuration. When a new version is compiled into the run-time directory (the component is loaded in the memory by the framework), this is detected during application self-inspection and the component is exchanged with another version using the `exchange` operation of the component framework (see figure 3(a)). If an arbitrary number of components needs to be exchanged this can be done by invoking `exchange` operation for each of the components. This also implies that components with dependencies can be exchanged safely, in a sequence of appropriate `exchange` calls, since the tasks using these components will be completed before the exchange takes place.

When wanting to exchange one of the algorithms that crosscut the overall system, the reconfiguration can be done by exchanging aspects. Exchange is done by first weaving the desired new aspect into affected components off-line, and then employing the reconfiguration of affected components as described above[5]. Note that it is not possible to exchange aspects if they are not encapsulated into interfaces and translated into corresponding real-time entities. Although the dynamic reconfiguration of aspects is done via component exchange, the benefits of aspect-orientation are still fully retained as changes to the code that crosscut many components are still done in an automated, efficient, and modular way via aspect weaving.

3.3 Enabling QoS Adaptiveness

Recall requirements R5 and R6, stating that the system administrator must be able to specify desired system QoS and transient state system QoS in terms of worst-case QoS and how fast the QoS should converge toward the desired QoS. To fulfill R5 and R6 and, therefore, provide QoS guarantees under system reconfiguration we employ feedback control [14]. Applying feedback to control the QoS of computer systems is referred to as feedback control scheduling (FCS), and this technique has been introduced as a promising foundation for QoS control of complex real-time systems [13–15].

The desired nominal system QoS is expressed in terms of a reference QoS, which gives the level of QoS that the system must provide when it is in the steady

[5] An aspect exchange can be, from the dynamic reconfiguration perspective, regarded as exchange of a number of components with new (woven) versions.

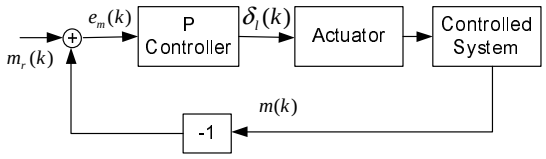


Fig. 4. Feedback control structure

state, i.e., when no reconfiguration is currently taking place and any effects of previous reconfiguration have passed. When a reconfiguration is taking place the system alternates to the transient state, which is characterized by fluctuations in QoS. The desired transient state QoS is usually expressed in terms of the maximum overshoot and the settling time [14]. The maximum overshoot M_p is the worst-case system QoS in a transient system state (under reconfiguration) and it is given in percentage. Hence, the maximum performance degradation of the system under reconfiguration is bounded by M_p. The settling time T_s is the time for the transient overshoot to decay and reach the steady state QoS. Hence, the settling time is a measure of system adaptability, i.e., how fast the system converges toward the desired QoS in the face of reconfiguration.

Typically, one is interested in controlling the performance of real-time systems using the metric deadline miss ratio [15], which gives the ratio of tasks that have missed their deadlines. We have employed a feedback-based QoS management method, referred to as FC-M [15], in the ACCORD dynamic reconfiguration framework. Using FC-M deadline miss ratio is controlled by modifying the admitted load. We say that a task is terminated when it has completed or missed its deadline. Let $missedTasks(k)$ be the number of tasks that have missed their deadline and $terminatedTasks(k)$ be the number of terminated admitted tasks in the time interval $[(k-1)T, kT]$. The deadline miss ratio, $m(k) = \frac{missedTasks(k)}{terminatedTasks(k)}$, denotes the ratio of tasks that have missed their deadlines.

In figure 4, the performance error, $e_m(k) = m_r(k) - m(k)$, is computed to quantize the difference between the desired deadline miss ratio, given by the reference $m_r(k)$, and the measured deadline miss ratio $m(k)$. The change to the load $\delta_l(k)$, which we denote as the manipulated variable, is derived using a P controller, hence, $\delta_l(k) = K_P e_m(k)$, where K_P is a tunable variable. The load target $l(k)$ is the integration of $\delta_l(k)$, i.e., $l(k+1) = l(k) + \delta_l(k)$. Admission control is then used to carry out the change in load, where a task is admitted into the system if the sum of the load of the task (that is waiting to be admitted) and the load of the already admitted tasks is less than the load target $l(k)$.

Consider the following example where the deadline miss ratio reference is set to 0.1 and the load threshold at the 10^{th} sampling instant is set to 0.9, i.e., $m_r = 0.1$ and $l(10) = 0.9$. A component exchange during the previous sampling interval has resulted in an increase in the execution time of the tasks, and consequently the deadline miss ratio has increased to $m(10) = 0.2$. Clearly, the component exchange has degraded the performance of the system. Therefore,

we need to reduce the deadline miss ratio to the reference value $m_r = 0.1$. This is done by taking the measured value of the deadline miss ratio from the system and computing the performance error $e_m(10) = m_r(10) - m(10) = (0.1 - 0.2) = -0.1$ and then computing the change in load, i.e., $\delta_l(10) = -0.1K_P$; following the steps in the feedback structure from figure 4. The load threshold during the next sampling interval is changed to $l(11) = l(10) + \delta_l(10) = 0.9 - 0.1K_P$. The admitted load is reduced as a result of a decrease in the load threshold and, consequently, the deadline miss ratio for the 11^{th} sampling instant is reduced.

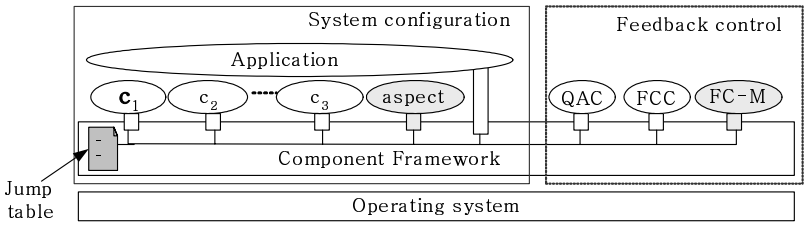


Fig. 5. ACCORD system configuration revisited

QoS management related components and aspects are implemented according to the extended RTCOM model and deployed into the ACCORD component framework. The structure of the dynamically reconfigurable system that guarantees QoS is depicted in figure 5. QoS guarantees are now satisfied in the framework by having an actuator component (AC) that implements changes in the manipulated variable and the feedback control component (FCC), which implements the control loop by measuring the controlled variable and computing the manipulated variable. The FC-M QoS algorithm is implemented as an aspect, crosscutting AC and FCC. For more details on how feedback loop can be designed using aspects and components and configured statically, as well as what types of possible QoS algorithms are supported, we refer interested readers to [16].

4 Summary

The majority of the component-based real-time systems focuses on static reconfiguration of a real-time system. However, real-time systems that are tightly coupled with their environment cannot be halted for reconfiguration to take place as their continuous operation is of paramount importance. Even if a reconfiguration takes place, affecting the temporal behavior of the system, the QoS provided by these systems must remain unaltered.

We have addressed this problem by presenting a QoS-adaptive framework for dynamic reconfiguration of real-time systems. Our framework is founded on aspectual component-based real-time system development, an approach that combines component-based and aspect-oriented software development with real-time system development. Hence, the framework supports reconfiguration in

terms of components and aspects, which increases the granularity of exchangeable parts. The framework also ensures that real-time performance guarantees are maintained even when the system undergoes dynamic reconfiguration. Using the QoS-adaptive framework for dynamic reconfiguration results in efficient development and software upgrading of real-time systems, conforming to strict QoS requirements that are applied to performance-critical real-time systems.

References

1. Buttazzo, G.C.: Hard Real-Time Computing Systems. Kluwer (1997)
2. van Ommering, R.: Building product populations with software components. In: Proceedings of the 24th ACM International Conference on Software Engineering (2002) 255–265
3. Stewart, D.B., Volpe, R., Khosla, P.K.: Design of dynamically reconfigurable real-time software using port-based objects. IEEE Transactions on Software Engineering **23** (1997)
4. Sandström, K., Fredriksson, J., Åkerholm, M.: Introducing a component technology for safety critical embedded realtime systems. In: Proceedings of the International Symposium on Component-based Software Engineering, Springer-Verlag (2004)
5. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.M., Irwin, J.: Aspect-oriented programming. In: Proceedings of the ECOOP. Volume 1241 of Lecture Notes in Computer Science., Springer-Verlag (1997) 220–242
6. Tešanović, A., Nyström, D., Hansson, J., Norström, C.: Aspects and components in real-time system development: Towards reconfigurable and reusable software. Journal of Embedded Computing **1** (2004)
7. Cerpa, A., Estrin, D.: ASCENT: Adaptive Self-Configuring sEnsor Networks Topologies. IEEE Transactions on Mobile Computing **3** (2004) 272–285
8. Engblom, J.: Analysis of the execution time unpredictability caused by dynamic branch prediction. In: Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS). (2003) 152–159
9. Isovic, D., Lindgren, M., Crnkovic, I.: System development with real-time components. In: Proceedings of ECOOP Workshop - Pervasive Component-Based Systems (2000)
10. ITEA project: (ROBOCOP deliverable 1.5) ROBOCOP ITEA project.
11. Hissam, S., Moreno, G., Stafford, J., Wallnau, K.: Enabling predictable assembly. The Journal of Systems and Software **65** (2003) 185–198
12. Díaz, M., Garrido, D., Llopis, L.M., Rus, F., Troya, J.M.: Integrating real-time analysis in a component model for embedded systems. In: Proceedings of the 30th IEEE Euromicro conference (2004)
13. Amirijoo, M., Hansson, J., Gunnarsson, S., Son, S.H.: Enhancing feedback control scheduling performance by on-line quantification and suppression of measurement disturbance. In: Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS). (2005)
14. Hellerstein, J.L., Diao, Y., Parekh, S., Tilbury, D.M.: Feedback Control of Computing Systems. Wiley-IEEE Press (2004)
15. Lu, C., Stankovic, J.A., Tao, G., Son, S.H.: Feedback control real-time scheduling: Framework, modeling and algorithms. Real-time Systems **23** (2002)
16. Tešanović, A., Amirijoo, M., Björk, M., Hansson, J.: Empowering configurable QoS management in real-time systems. In: Proceedings of the Fourth ACM SIG International Conference on Aspect-Oriented Software Development (2005)