Providing Configurable QoS Management in Real-Time Systems with QoS Aspect Packages

Aleksandra Tešanović[1], Mehdi Amirijoo[1], and Jörgen Hansson[1,2]

[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. Current quality of service (QoS) management approaches in real-time systems lack support for configurability and reusability as they cannot be configured for a target application or reused across many applications. In this paper we present the concept of a QoS aspect package that enables developing configurable QoS management for real-time systems. A QoS aspect package represents both the specification and the implementation of a set of aspects and components that provide a number of QoS policies. A QoS aspect package enables upgrades of already existing systems to support QoS performance assurance by adding aspects and components from the package. Furthermore, a family of real-time systems can easily be developed by adding aspects from the QoS aspect package into an existing system configuration. We illustrate the way a family of real-time database systems is developed using the QoS aspect package with a case study of an embedded real-time database system, called COMET. Our experiments with the COMET database have shown that it is indeed possible to design a real-time system without QoS management and then with a reasonable effort add the QoS dimension to the system using a QoS aspect package.

1 Introduction

Real-time systems are characterized by rigid requirements on quality of service (QoS). Namely, the failure to deliver a correct response in a timely manner results in deterioration of system performance and, in the worst case, a catastrophe. For example, in multimedia applications decoding of frames (images or sound) has to be done in a timely manner, otherwise the results of the computations are of no value [1]. Other examples can be found in telecommunications where a user request (e.g., placing a call) has to be addressed as soon as possible and packets have to arrive in a timely manner to avoid poor voice quality [2]. Furthermore, the engine of a modern car is controlled by an electronic control unit (ECU) that continuously monitors and controls the engine using actuators [3]. The control computations, e.g., fuel injection, must be finished within a certain time frame, otherwise the performance of the engine decays and can lead to an engine breakdown, potentially resulting in a catastrophe for the driver

A. Rashid and M. Aksit (Eds.): Transactions on AOSD II, LNCS 4242, pp. 256–288, 2006.

[3, 4]. For these reasons, the majority of real-time research over the years has focused primarily on delivering appropriate methods for ensuring real-time QoS, e.g., [5, 6, 7, 8, 9, 10, 11, 12]. In recent years, however, with a dramatic increase in the use of real-time systems, the requirements on low development costs, short time to market, and high degree of configurability have become increasingly important in real-time system development [13].

Since QoS management is a core issue of real-time computing, configurability and reusability of software cannot be achieved without first ensuring configurability and reusability of QoS management. However, in our study of existing work we have observed that approaches for QoS management in real-time systems [5, 6, 7, 8, 9, 10, 11, 12] do not comply with essential software engineering principles that enable configuration and reuse.

In this paper we address the software engineering challenges in developing configurable QoS management for real-time systems by proposing the concept of a *QoS aspect package*, which represents a way of packaging the specification and implementation of real-time QoS management for reuse and configuration. At an abstract level, a QoS aspect package represents a way of specifying configurable QoS management, where a real-time QoS management policy is specified independently of an application by means of aspects and components. At a concrete level, a QoS aspect package consists of aspects and components implementing a variety of QoS management policies.

When studying existing QoS management approaches, we observed that a majority of approaches assume that a real-time system has a QoS management infrastructure upon which algorithms implementing specific QoS policies are implemented. The infrastructure is implemented as an integral part of the system and consists of services, functions, or methods for adjusting the system load. Algorithms implementing QoS policies crosscut the overall system, and use the services provided by the infrastructure to ensure performance guarantees.

The concept of a QoS aspect package provides a way of decomposing and implementing QoS management for a family of real-time applications. The decomposition is done such that the main functional infrastructure of QoS management is encapsulated into a set of components and various QoS policies are encapsulated into aspects. For a family of real-time applications a QoS aspect package has a unique set of components and possibly a great number of aspects implementing various QoS policies.

A QoS aspect package enables the following:

- specification and implementation of QoS policies independent of a real-time application,
- upgrades of existing systems to support QoS performance assurance by simply adding aspects that implement different QoS policies from the QoS aspect package,
- development of families of real-time systems with variations in QoS management, where variations (aspects from the package) are injected into the QoS management infrastructure at explicitly declared joinpoints of the components,

- offline and online configuration of QoS management as aspects can either be woven into the resulting system offline and then deployed into the runtime environment (preferred option in most real-time systems), or can be deployed into the running system online [14] (preferred when a high degree of availability is required),
- reusability of QoS management as aspects implemented within a QoS aspect package can be reused in multiple applications,
- configurability and reusability of QoS management for a large class of real-time systems across many application areas as the QoS aspect package concept can be used in any real-time system conforming to a set of requirements elaborated in this paper, e.g., available joinpoints in the code and conformance to an aspect language.

Since a QoS aspect package could consist of many aspects and components, the designer might need assistance in choosing the relevant subset of aspects and components for configuring QoS management of a system. Therefore, we provide appropriate tools for configuring QoS management. The tools also assist in determining if a suggested QoS configuration, i.e., adding of a QoS aspect package, is feasible for the target application.

We present a proof of concept implementation of a QoS aspect package for an embedded real-time database, called COMET [15, 16]. We report our experiences in using a QoS aspect package for real-time system development. We believe that they are valuable for current and future implementors of real-time systems that would like to use aspects for system development.

The paper is organized as follows. In Sect. 2 we present a background to real-time systems and discuss how QoS is maintained. The problem formulation is then presented in Sect. 3, where we identify problems in current QoS management approaches for real-time systems. We propose, in Sect. 4, a QoS aspect package as a possible solution to the identified problems. In Sect. 5 we present the COMET database, to which QoS management has been added using a QoS aspect package. Experiences from using aspect-orientation in real-time system design and implementation are discussed in Sect. 6. The paper finishes with the main conclusions in Sect. 7.

2 QoS Management

We now review the main characteristics of real-time systems and then discuss the key properties of QoS management in these systems.

2.1 Real-Time System Model

Real-time systems are traditionally constructed of concurrent programs, called tasks. A task is characterized by a number of temporal constraints amalgamating a task model. One of the most important temporal constraints a task needs to satisfy is a deadline, which represents the time point by which the task needs to be completed.

Depending on the consequence of a missed deadline, real-time systems can be classified as hard or soft. In hard real-time systems, e.g., aircraft and train control, the consequences of missing a deadline can be catastrophic. In soft real-time systems, e.g., video streaming and mobile services, missing a deadline does not cause catastrophic damage to the system but affects performance negatively.

Tasks in a system are scheduled by a real-time scheduler that determines an order of task execution that allows tasks to meet their respective deadlines [7]. Depending on the type of a real-time system and the scheduling policy used, different task models are applicable. For example, in hard real-time systems scheduling policies typically assume periodic tasks or sporadic tasks (aperiodic tasks with a minimum interarrival time), with known worst-case execution times [7]. Knowing the interarrival times and the worst-case execution times of the tasks enables a designer to check whether deadlines are met. Since no deadline misses can be tolerated, analysis of the system is normally done before system deployment, implying that the utilization of the system is fixed beforehand.

The assumptions taken for hard real-time systems are relaxed for soft real-time systems, i.e., for soft real-time systems it is assumed that the tasks can arrive aperiodically and that worst-case execution times are not available.[1] Instead, tasks in such systems are associated with an estimate of the average execution time. The execution time is typically estimated by profiling the code or by running experiments and monitoring the execution time of a task.

Since tasks in a soft real-time system arrive with unknown interarrival times and inaccurate worst-case execution times, the workload submitted to the system is unpredictable and can in the worst case cause the system to be overloaded. This means that it is difficult to adjust the utilization to a certain level beforehand (since admitted workload and utilization are related) and missing deadlines is inevitable. The latter follows from the fact that there is a correlation between the utilization of a system and deadline misses [7]. Rather than striving to achieve a certain utilization or meet deadlines, the focus in the resource allocation and task scheduling in soft real-time systems lies in mechanisms for ensuring QoS predictability, i.e., guaranteeing that the utilization does not exceed a certain threshold and no more than a certain number of tasks miss their deadlines during a period of time.

In recent years, a new set of requirements focusing on reuse and reconfiguration of real-time systems have emerged, resulting in the introduction of a number of approaches to development of reconfigurable real-time software using component-based and/or aspect-oriented software development, e.g., [16, 18, 19, 20]. In these approaches, components (possibly woven with aspects) are executed by tasks in a run-time environment. A task typically executes one or multiple operations a

[1] Note that in practice deriving accurate worst-case execution times is difficult as execution times depend on branches in the code, cache, processor pipelining, shared resources (forcing tasks to wait for allocated resources), and scheduling. This also implies that hard real-time system premises, e.g., the existence of accurate worst-case execution time estimates, are not realistic for a majority of recently developed intricate and large-scale real-time systems [17].

component. Which component is going to be executed by which task is determined before the system is deployed, in a process called component-to-task mapping, depending on the available resources in the underlying real-time environment. In this work we allow tasks and components to be mapped arbitrarily, i.e., without loss of generality a task can execute several components, and/or a component can consist of multiple tasks. We omit details about the mapping in this paper due to its length, and instead refer the interested reader to [20, 21] for further details.

2.2 QoS Management in Real-Time Systems

In this paper we focus primarily on methods for QoS management in soft real-time systems. This is because soft real-time systems are founded on more realistic assumptions than hard, i.e., worst-case execution times do not need to be known and aperiodic tasks are supported. Also, soft real-time systems constitute a wide spectrum of existing and emerging real-time applications, such as telecommunication, web servers, video streaming, and automatic and vehicle control.

Most of the research dealing with QoS management in real-time systems today uses some form of feedback for controlling the QoS, as it has been shown that feedback control is highly effective to support the specified performance of soft real-time systems that exhibit unpredictable workloads. For example, it has been shown that exact estimates of task execution times and arrival patterns are not required when feedback control is used [8, 22]. This implies that we can apply feedback control-based QoS management techniques on a wide spectrum of large-scale and complex systems where, e.g., execution time estimates are not available. We therefore primarily focus on QoS management using feedback control and refer to it as feedback-based QoS management.

In the remainder of this section, we introduce the general feedback control structure that has been used in all feedback-based QoS management approaches, and elaborate on representative QoS management approaches in real-time systems.

Feedback Control Structure. A typical structure of a feedback control system is given in Fig. 1 along with the control-related variables. A sampled variable $a(k)$ refers to the value of the variable a at time kT, where T is the sampling period and k is the sampling instant. In the remainder of the paper we omit k where the notion of time is not of primary interest.

Input to the controller is the difference between the reference $y_r(k)$, representing the desired state of the controlled system, and the actual system state given by the controlled variable $y(k)$, which is measured using the sensor. Based on the performance error, $y_r(k) - y(k)$, the controller changes the behavior of the controlled system via the manipulated variable $\delta_u(k)$ and the actuator. The objective of the control is to compute $\delta_u(k)$ such that the difference between the desired state and the actual state is minimized, i.e., we want to minimize $(y_r(k) - y(k))^2$. This minimization results in a reliable performance and system adaptability as the actual system performance is closer to the desired performance.

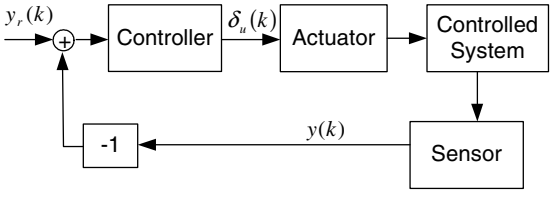

Fig. 1. An architecture of the real-time system using feedback control structure

Feedback-Based QoS Management of Real-Time Systems. Here we review the main feedback-based QoS management policies used in various real-time applications. The goal is to illustrate the variety of sensors, actuators, and controllers used in QoS management of real-time systems and to further motivate our work.

In controlling the utilization in soft real-time systems [6, 8, 11, 23, 24], the sensor periodically measures the utilization of the system. A controller compares the utilization reference with the actual measured utilization and computes a change to the manipulated variable. The actuator then carries out the change in utilization, in one of the following ways:

- The precision or the quality of the result of the tasks can be modified by applying imprecise computation techniques [25], which have been introduced to allow flexibility in operation and to provide means for achieving graceful degradation during transient overloads, e.g., a task is decomposed into a mandatory part producing a nonoptimal but satisfactory result, and optional parts that refine the initial result. This class of techniques enables trading CPU resource needs for the precision of requested service results. Using these techniques, the utilization is decreased by reducing the execution time of the tasks. This in turn implies that the precision of the task results is lowered. Conversely, the result precision increases as the utilization increases.
- The utilization is directly related to the the interarrival times of the tasks, i.e., the utilization increases with decreasing interarrival times. Hence, the utilization can be easily adjusted by changing the interarrival times of the tasks [8, 11].
- Since the utilization of the system increases with number of admitted tasks, it can be changed by enforcing admission control, where a subset of the tasks are allowed to execute [22].

In a number of real-time applications, e.g., video streaming and signal processing, tasks can deliver results of less precision in exchange for timely delivery of results. For example, during overloads alternative filters with less output quality and execution time may be used, ensuring that task deadlines are met. In general, the precision of the results increases with the execution time given to a task, calling for the use of a feedback structure to control the precision [5, 12, 26]. In such a structure, the sensor is used to estimate the output quality of the tasks, while a controller forces the tasks to maintain an output precision equal to the

reference. The execution time given to individual tasks is controlled by the actuator, thereby, ensuring that the output precision is maintained at the desired level.

In telecommunication and web servers, arriving packets and requests are inserted into queues, where the requests wait to be processed. The time it takes to process requests once they arrive at a server is proportional to the length of the queue, i.e., the processing time increases with the length of the queue. Controlling the queue length is the key to guarantee timely processing of requests. If the queue length is too long, then the time it takes to process a request may be unacceptable as there are time constraints on the requests. Typically, a feedback controller is used to adjust the queue length such that the length equals its reference [10, 27, 28, 29, 30]. Ways of manipulating the queue length include changing the admission rate of the requests. Namely, by admitting more arriving requests the queue length increases, thus, increasing the latency time of the requests.

Controlling the execution times of tasks is important in real-time systems with constraints on energy consumption. Efforts have been carried out trying to reduce energy consumption in real-time systems, while preserving timely completion of tasks [31]. In this case execution times are monitored and the voltage and, thus, frequency of the CPU is varied such that the power consumption is reduced and tasks are executed in a timely manner. Hence, the sensor is used to measure the execution time of the tasks and the actuator is used to carry out the change in the voltage or the frequency of the CPU.

By studying the examples above, we note that there are many ways to implement sensors and actuators. This shows that the choice of a sensor and an actuator depends highly on the type of an application and its constraints. Control-related variables also vary depending on the application being controlled. Furthermore, controllers are implemented using a particular metric and a controlling algorithm, explicitly tuned for a specific application.

In the feedback-based policies mentioned in this section, traditional control algorithms such as PID, state-feedback, and Lead-Lag are used (details on these algorithms can be found in [32]). These algorithms assume that the controlled system does not change during run time, i.e., the behavior of the controlled system is time-invariant. Hence, the control algorithm parameters are tuned offline and may not be altered during run time. However, the behavior of some real-time systems is time-varying due to significant changes in load and/or execution times. This is addressed by employing adaptive control algorithms [33] where the behavior of the controlled system is monitored at run time and the control algorithm parameters are adapted accordingly. Hence, different types of controllers may be employed and, as such, there is a need for configurability of the controllers.

As can be observed, there are many different types of control algorithms that are applied to specific QoS management policies. The choice of a control algorithm depends on the control performance, run-time complexity, memory footprint, and adaptiveness.

3 Problem Formulation

As discussed in Sect. 2, a number of QoS management approaches have been explicitly developed to suit the needs of a particular application. Specifically, if feedback-based QoS management is considered, a QoS controller (see Fig. 1) is typically developed to control a certain type of metric, and it cannot be reused for another metric unless extensive modifications are made to the system and the controller. Furthermore, QoS policies used in existing approaches to real-time QoS management [5, 6, 8, 9, 10, 11, 12, 23, 24, 26, 27, 28, 29, 30, 31, 34, 35, 36] cannot be exchanged or modified separately from the system. Hence, QoS management is specific to a real-time application and a QoS management approach developed for one application cannot easily be reused in another application.

Additional limitation of these QoS approaches is that the architecture of the system is developed such that it is tightly integrated with QoS management, yielding in a system that has a fixed, monolithic, and nonevolutionary architecture. Modifications of QoS management require complex modifications of the code and the structure of the overall system. Moreover, current approaches do not enable taking existing systems without QoS management and adapting them to be used in an application with specific QoS needs. The trend in the vehicular industry, for example, is to incorporate QoS guarantees in vehicle control systems [15, 37]. In this case a cost-effective way of building QoS-aware vehicle control systems is to efficiently incorporate QoS mechanisms into already existing systems.

Software engineering research has shown that the implementation of QoS management should be reusable across applications and highly configurable [38, 39]. A number of software engineering approaches that provide configurable QoS management do exist. In these approaches QoS policies are encapsulated into aspects, developed independently of the application, and injected into the distributed middleware system when needed. In such approaches *qoskets* are frequently used as the means of encapsulating the code of a QoS policy [38, 39, 40]. Qoskets represent a set of QoS class specifications and their implementations. They are used for monitoring and adapting the system to, e.g., varying network loads. Separating QoS policies from the application and encapsulating them into reusable and exchangeable units has proven to be conducive to cost-effective development of software systems that can be more easily maintained and evolved [38, 39, 40]. Moreover, it has been shown that CoSMIC [41] is useful for facilitating automated system configuration and deployment [39, 41]. CoSMIC is a domain-specific tool based on the concept of model-driven architectures, developed to aid in configuration of QoS in the domain of distributed real-time CORBA-based middleware systems.

While software engineering approaches and tools for reusable and configurable QoS management [38, 39, 40] enable extending existing systems by adding qoskets, they assume a CORBA-based (including real-time versions) architecture. The domain of the tools developed for assisting in configuring QoS is also limited to CORBA-based systems [41]. This type of evolution and tool support is not sufficient for a large class of real-time systems as most of the existing systems

do not have CORBA architecture, e.g., vehicle systems, control systems, and mobile devices.

4 Enabling Configurable Real-Time QoS

In this section we first elaborate on the requirements that a system needs to fulfill in order to use a QoS aspect package for configuring QoS. Then, we present the main idea behind the concept of a QoS aspect package using an example of feedback-based QoS management, and outline the procedure for configuring QoS management. This is followed by a discussion on how tools can be used to support configuration of QoS.

4.1 System Requirements

The concept of a QoS aspect package as the means of configuring and extending an existing system applies both to the class of traditional (monolithic) real-time systems and to the class of component-based real-time systems, provided that they conform to the requirements we identified for each class.

Traditional real-time systems should: (1) be written in a language that has a corresponding aspect language; (2) have the source code of the system available; and (3) have the code structured in fine-grained pieces that perform well-defined functions, i.e., good coding practice is employed.

Component-based real-time systems should be built using "glass box" or "gray box" component models. These models imply that components have well-defined interfaces, but also internals are accessible for manipulation by the software developer. Examples include Koala [18], RTCOM [16], PBO [19], AutoComp [20], and Rubus-based component models [42].

In addition, both monolithic and component-based real-time systems should have functions for controlling the system load. Recall from Sect. 2.2 that there are multiple ways of controlling the load in the system, e.g., by changing the output quality of the tasks [25], modifying the period of the tasks [8, 11], admission control [22], and changing the frequency of the CPU [31]. This implies that the tasks must be scheduled by an online scheduler [7], e.g., earliest deadline first or rate monotonic [43] (in contrast to systems with, for example, a cyclic executive).

Given that a system conforms to the named requirements, the concepts and tools for configuration that we present next can be used for adding and configuring QoS management.

4.2 The QoS Aspect Package

At the design and specification level, the QoS aspect package prescribes a hierarchical QoS management architecture that, at the highest level, consists of two classes of entities: the QoS component and the QoS aspect types (see Fig. 2). The top level of the QoS management architecture can be used for any QoS management. In the remainder of the paper we discuss feedback-based QoS management,

where components include a QoS actuator component, a feedback controller component, and a sensor component. The aspect types include the following types of aspects (see Fig. 2):

- QoS management policy aspects,
- QoS task model aspects, and
- QoS composition aspects.

The QoS component is defined as a gray box component implementing functions that provide an infrastructure for the QoS management polices. As such, a component has an interface that contains the following information: (1) functionality, in terms of functions, procedures, or methods that a component requires (uses) from the system, (2) functionality that a component provides to the system, and (3) the list of explicitly declared joinpoints where changes can be made in the component functionality. Explicitly declared joinpoints are especially useful in the context of producing families of systems with variations in their QoS management. One other motivation for explicitly declaring these joinpoints is to ensure that the evolution of the QoS aspect package and the system can be done as efficiently as possible. Namely, having access to the points in the component structure, the aspect developer when developing new QoS aspects does not necessarily have to have full knowledge of the component code to develop aspects quickly and successfully (as we elaborate further in our experience report in Sect. 6).

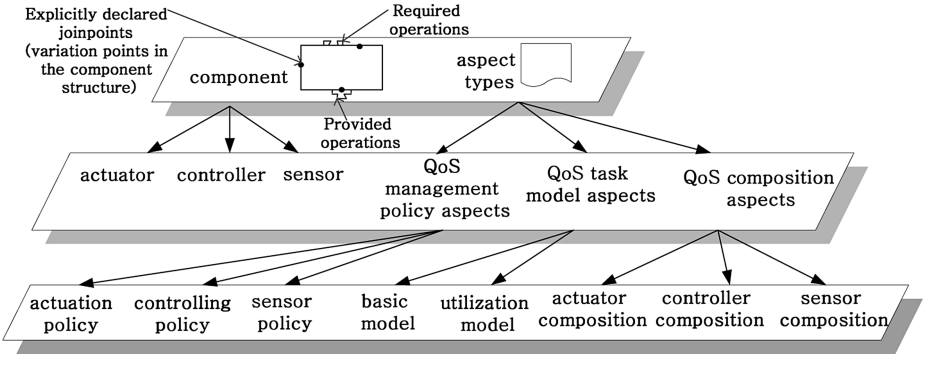

Fig. 2. A QoS aspect package at the specification level

The actuator is a QoS component and, in its simplest form, acts as a simple admission controller. It publishes a list of joinpoints in its interfaces where different actuator policies can be woven. Similarly, the feedback controller is by default designed with a simple control algorithm and identified joinpoints in the component structure, such that the controller can be extended to support more sophisticated control algorithms, e.g., adaptive control [33]. The sensor component collects necessary data and possibly aggregates it to form the metric

representing the controlled variable. In its simplest form the sensor component measures utilization, which is commonly used as a controlled variable [8]. The sensor component publishes a set of joinpoints where it is possible to change the measured metric.

The QoS management policy aspects, namely actuator policy, controller policy, and the sensor policy, adapt the system to provide a variety of QoS management policies. Depending on the target application, the aspects modify the controller to support an appropriate QoS policy, and also change the actuator and the sensor component according to the choice of manipulated variable and controlled variable, respectively. For example, if the deadline miss ratio is to be controlled by changing the computation result quality of the tasks, then a QoS policy aspect measuring the deadline miss ratio is chosen. The actuator is modified by the aspect, exchanging the admission policy for an actuation mechanism where the quality of the tasks' results are modified.

The QoS task model aspects adapt the task model of a real-time system to the model used by QoS policies, e.g., a utilization task model needs to be used for a feedback-based QoS policy where utilization of the system is controlled. There can be a number of aspects defined to ensure enrichments or modifications of the task model, e.g., by adding various attributes to tasks, so that the resulting task model is suitable for distinct QoS or applications' needs. Concrete examples of task models are given in Sect. 5.3.

The QoS composition aspects facilitate composition of a real-time system with the QoS-related components: controller, actuator, and sensor. As we illustrate in Sect. 5.5, these aspects are rather simple and straightforward to implement.

Once a QoS aspect package is specified as described above, i.e., components and aspect types are identified, it needs to be populated with the actual implementations of aspects and components, which enable a system designer to develop a family of applications. Therefore, each family of applications would have its own QoS aspect package.

Now, an existing system that complies with requirements from Sect. 4.1 is configured for the specific real-time QoS management as follows.

- The architecture of the existing system is inspected and joinpoints are identified.
- The new context in which the system is going to be used, i.e., the application domain, is determined.
- Given the new usage context of the system, a suitable QoS policy consisting of the sensor policy, controlling policy, and the actuation policy, is identified.
- If the aspects implementing the QoS policy do not exist in the given QoS aspect package, then the corresponding aspects are defined, developed, and added to the package. Aspects use joinpoints in the existing system to inject the QoS policy. Similarly, the QoS aspect package has to be populated with a sensor component, controller component, and an actuator component, if these do not already exist in the package.

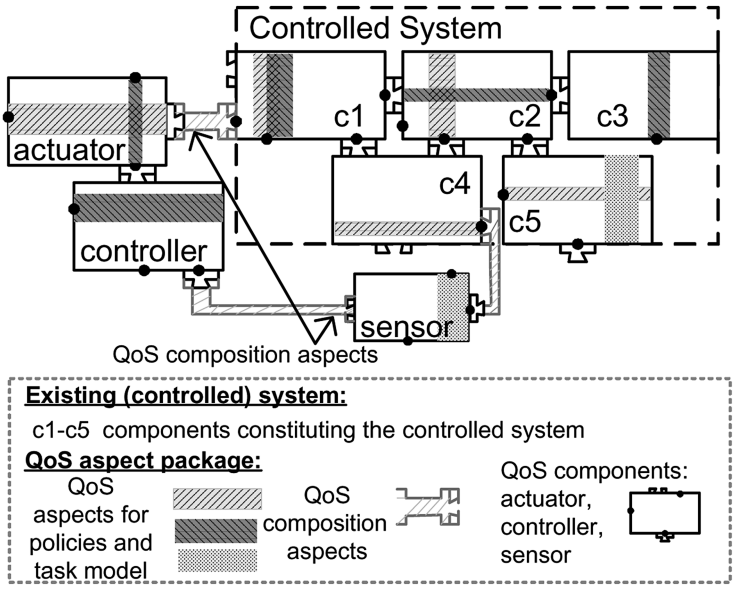

Fig. 3. A real-time system where a QoS aspect package is used for configuring QoS management

– Aspects and components for a specific QoS configuration are chosen from the QoS aspect package and woven into the existing system.

A real-time system where a QoS aspect package is applied is shown in Fig. 3. In this figure, the controlled (existing) system is a component-based system that consists of gray box or glass box components $c1, \ldots, c5$, which have well-defined interfaces and internals accessible for modification (corresponding to the second type of real-time systems discussed in Sect. 4.1).[2]

QoS composition aspects from a QoS aspect package are used to add the sensor, controller, and actuator to the parts/components of the system where needed; these aspects are represented with gray dashed lines between component connections in Fig. 3. Additionally, QoS composition aspects offer significant flexibility in the system design as the feedback loop can easily be placed "outside" the system and between any components in the system by simply adding QoS composition aspects. QoS management policies are thus added to the system using appropriate aspects from the package. Moreover, QoS policies can easily be exchanged by adding or changing aspects within the QoS management policy type. Hence, a QoS aspect package ensures that QoS management policies are modifiable and configurable, depending on the application requirements.

[2] The explanation and the main points are the same as if the depicted controlled system would be a traditional monolithic real-time system conforming to the requirements listed in Sect. 4.1.

4.3 Tool Support

In this section we describe tool support for choosing relevant aspects and components from a QoS aspect package for a specific QoS configuration.

The tool supporting QoS configuration is a part of a tool called ACCORD modeling environment (ACCORD-ME). ACCORD-ME is, in turn, an integral part of the ACCORD tool development environment, which provides support for the development of reconfigurable real-time systems [44].[3] ACCORD-ME is a model-based tool, implemented using the generic modeling environment (GME), a toolkit for creating domain-specific modeling environments [45]. In general, the creation of a GME-based tool is accomplished by defining metamodels that specify the modeling paradigm (modeling language) of the application domain. In our case, based on the content of the QoS aspect package, modeling paradigms for QoS management are defined for ACCORD-ME. QoS modeling paradigms are given as UML diagrams and they define the relationship between components and aspects in the package, their possible relationship to other components of the system, and possibilities of combining components and aspects into different configurations. Note that a modeling paradigm needs to be specified for each new domain, i.e., each family of real-time systems.

The GME environment also enables specifying multiple tool plug-ins, which can be developed independently of the environment and then integrated with the GME to be used for different modeling and/or analysis purposes. Exploiting this, ACCORD-ME is developed with a number of subtools, namely the configurator, memory and worst-case execution time (M&W) analyzer, and formal verifier (see Fig. 4).

Next we explain the configurator part of ACCORD-ME and its role in configuration of QoS management. We omit detailed description of ACCORD-ME analysis tools (M&W analyzer and formal verifier) and refer the interested reader to [44].

When configuring QoS management of a system, the inputs to ACCORD-ME are the QoS requirements of the application that are placed on the system. QoS requirements can include specification of the controlled and manipulated variables, and the parameters characterizing the task model. The requirements can also be QoS policies that need to be enforced by the system, e.g., the utilization policy, the deadline miss ratio policy, and dynamic adaptation of the control algorithms. Based on the requirements and the predefined QoS modeling paradigms, the system developer can activate the configurator subtool to get adequate support for choosing relevant aspects and components from the QoS aspect package. To suggest a set of appropriate aspects and components, the configurator compares the requirements with the available configurations. Based on this analysis, configurator suggests the components and aspects that are found in suitable configurations.

[3] The name ACCORD-ME originates from the name of the approach for aspectual component-based real-time system development (ACCORD) [16] for which this tool was initially developed.

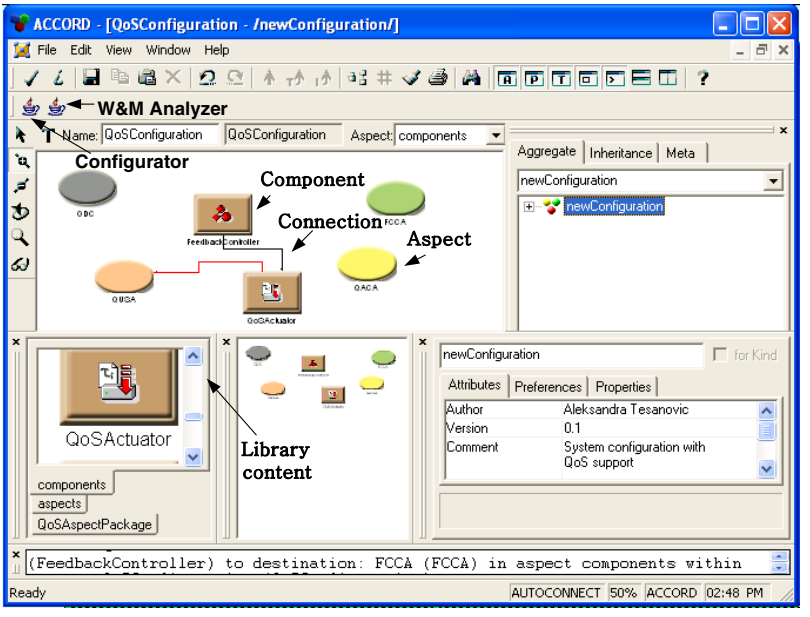

Fig. 4. The editing window of ACCORD-ME, which is a part of the ACCORD Development Environment

The configurator provides three levels of support, expert, configuration, and requirement-based, based on the expertise and preferences of the developer. The expert option is used by developers familiar with the content of the QoS aspect package and all of its provided functionality and policies. The configuration-based option gives a list of possible QoS configurations of the system that can be created from the QoS aspect package. The requirement-based option provides the system developer with a list of possible choices of parameters in the feedback control structure, e.g., controlled and manipulated variables, from which the developer can choose a relevant subset suitable for a particular application. Thus, developers do not need to know the contents of the QoS aspect package to be able to configure QoS management.

After one of the above options is chosen, the configurator loads the relevant components and aspects into the ACCORD-ME editing area, where the designer can assemble the system by connecting the components and aspects, as shown in Fig. 4. Each aspect and component contains information stating with which it can be connected to form a functionally correct QoS management configuration.

The obtained QoS configuration can be analyzed by other subtools available in ACCORD-ME. Specifically, formal analysis can be performed to ensure that adding constituents of the QoS aspect package preserves already proven properties of the original system [46], thereby ensuring that functionality of the original system is preserved or modified in a well-understood way. For the purposes of

formal analysis, a QoS configuration is transformed from a UML-based model to a timed automata-based model.

The output of ACCORD-ME is a configuration of a system that satisfies a specific QoS need of the application and that is functionally correctly assembled. The configuration, i.e., components, aspects, and their relationships, is stored in an XML file. This XML file is then fed to the tool of the ACCORD development environment, called the configuration compiler, which takes as input: (1) the information obtained from ACCORD-ME about the created QoS configuration, and (2) the source code of the needed aspects and components from the QoS aspect package (the source code of the system that is being controlled is also required). Based on this input, the configuration compiler generates a compilation file, which is used to compile source code of aspects and components into the final system. The configuration compiler also provides documentation about the generated QoS configuration that can later be used for maintaining the system.

5 Case Study: COMET Database

In this section we present a case study on a component-based embedded real-time database, called COMET. Initially, we developed COMET to be suitable for hard real-time applications in vehicular systems [15, 16]. Thus, the initial COMET implementation does not provide QoS guarantees. To adopt COMET to real-time systems with performance guarantees we developed a COMET QoS aspect package. In order to understand the choices made when developing different aspects for COMET QoS, we give an overview of COMET, and present the QoS management policies used for implementation of the QoS aspect package. We then discuss the data and transaction models used in various COMET configurations. We also describe a number of COMET QoS configurations and present the implementation of representative components and aspects. Finally, we demonstrate experimentally that COMET with the QoS extensions indeed provides the required QoS guarantees.

5.1 COMET Overview

The architecture of the COMET database consists of a number of components (see Fig. 5a): the user interface component, the transaction manager component, the index manager component, and the memory manager component. The user interface component provides a database interface to the application, which enables a user (i.e., an application using the database) to query and manipulate data elements. User requests are parsed by the user interface and are then converted into an execution plan. The transaction manager component is responsible for executing incoming execution plans, thereby performing the actual manipulation of data. The index manager component is responsible for maintaining an index of all tuples in the database. The COMET configuration containing the named components provides only basic functionality of the database, and this basic COMET configuration is especially suitable for small embedded vehicular systems [15].

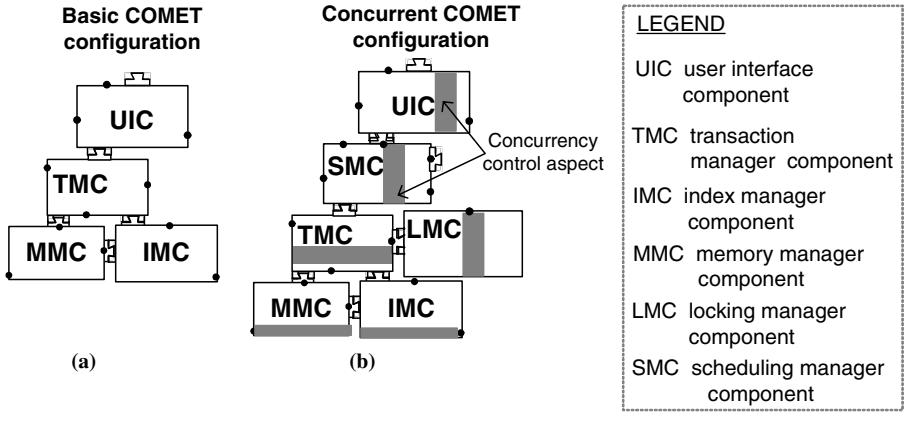

Fig. 5. Basic and concurrent COMET configurations

Depending on the application with which the database is to be integrated, additional aspects and components can be added to the basic COMET configuration. For example, to enable concurrent access to the database two additional components, the locking manager component and the scheduling manager component, are needed (see Fig. 5b). The scheduling manager component is responsible for registering new transactions, i.e., tasks in a database system, to the system and scheduling them according to the chosen scheduling policy, e.g., earliest deadline first [47]. The locking manager component is responsible for obtaining and releasing locks on data items accessed by transactions. Concurrency control aspects, providing algorithms for detecting and resolving conflicts among transactions, can also be woven to the system. The concurrent COMET configuration is out of scope of this paper and we refer interested readers to [16].

Each component has been developed according to RTCOM [16] component model we developed previously. RTCOM is compliant with the QoS component discussed in Sect. 4.2 as it has interfaces where it publishes (provided and required) operations and declares joinpoints where variations can be introduced. For example, the transaction manager component executes transactions by executing an operation `getResult()` declared in its provided interface. The scheduling manager declares an operation `createNew()` in its provided interface, which registers transactions. Moreover, transactions in the scheduling manager component are scheduled using method `scheduleRecord()`, which is explicitly declared in the component interface as the joinpoint where variations in the structure of the scheduling manager component can be done.

5.2 QoS Policies

Given that we want to use the COMET database with applications that require performance guarantees, we need to choose some existing QoS policies and develop the QoS aspect package to integrate them into the database. Hence, in

this section we give an overview of three feedback-based QoS management policies that we use in our case study; we found that these are especially suitable for ensuring performance guarantees in real-time systems. First we describe the feedback miss ratio control (FC-M) [8], where deadline miss ratio is controlled by modifying the number of admitted transactions. This is followed by a description of QMF [48], which is a QoS-sensitive approach for miss ratio and freshness guarantees used for managing QoS in real-time databases. Finally, we give a brief description of two adaptive QoS algorithms, the self-tuning regulator and the least squares regression algorithm [33].

FC-M uses a control loop to control the deadline miss ratio by adjusting the utilization in the system. We say that a transaction is terminated when it has completed or missed its deadline. Let $missedTransactions(k)$ be the number of transactions that have missed their deadline and $admittedTransactions(k)$ be the number of terminated admitted transactions in the time interval $[(k-1)T, kT]$. The deadline miss ratio,

$$m(k) = \frac{missedTransactions(k)}{admittedTransactions(k)} \tag{1}$$

denotes the ratio of transactions that have missed their deadlines. The performance error, $e_m(k) = m_r(k) - m(k)$, is computed to quantize the difference between the desired deadline miss ratio $m_r(k)$ and the measured deadline miss ratio $m(k)$. Note that $m(k)$ is a controlled variable, corresponding to $y(k)$ in Fig. 1, while $m_r(k)$ is a reference, corresponding to $y_r(k)$. The change to the utilization $\delta u(k)$, which we denote as the manipulated variable, is derived using a P-controller [32], hence, $\delta u(k) = K_{\mathrm{P}} e_m(k)$, where K_{P} is a tunable variable. The utilization target $u(k)$ is the integration of $\delta u(k)$. Admission control is then used to carry out the change in utilization.

Another way to change the requested utilization is to apply the policy used in QMF [48], where a feedback controller, similar to that of FC-M, is used to control the deadline miss ratio. The actuator in QMF manipulates the quality of data in real-time databases in combination with admission control to carry out changes in the controlled systems. If the database contains rarely requested data items, then updating them continuously is unnecessary, i.e., they can be updated on-demand. On the other hand, frequently requested data items should be updated continuously, because updating them on-demand would cause serious delays and possibly deadline overruns. When a lower utilization is requested via the deadline miss ratio controller, some of the least accessed data objects are classified as on-demand, thus, reducing the utilization. In contrast, if a greater utilization is requested then the data items that were previously updated on-demand, and have a relatively higher number of accesses, are moved from on-demand to immediate update, meaning that they are updated continuously. This way the utilization is changed according to the system performance.

The QoS management approaches presented so far in this section are not adaptive as they use linear feedback control and assume that the real-time system is time-invariant, which implies that the controller is tuned and fixed for that particular environment setting. For time-varying real-time systems it is

beneficial to use adaptive QoS management that enables the controller in the feedback loop to dynamically adjust its control algorithm parameters such that the overall performance of the system is improved. Two representative adaptive QoS approaches are the self-tuning regulator and the least squares regression model [33].

5.3 Data and Transaction Model

We consider a main memory database model, where there is one CPU as the main processing element. We consider the following data and transaction models.

In the basic configuration of COMET we have a *basic data model* and a *basic transaction model*. The basic data model contains metadata used for concurrency control algorithms in databases. The basic transaction model characterizes each transaction τ_i only with a period p_i and a relative deadline d_i. However, QoS algorithms like FC-M, QMF, and adaptive algorithms require more complex data and transaction models that capture, e.g., metadata that express temporal constraints, such as mean interarrival and execution times.

In the *differentiated data model*, data objects are classified into two classes, temporal and nontemporal [49]. Nontemporal data constitutes data objects with no temporal requirements, e.g., arrival times and deadlines. For temporal data we only consider base data, i.e., data objects that hold the view of the real-world and are updated by sensors. A base data object b_i is considered temporally inconsistent or stale if the current time is later than the timestamp of b_i followed by the absolute validity interval avi_i of b_i, i.e., $currenttime > timestamp_i + avi_i$.

Both FC-M and QMF require a transaction model where transaction τ_i is classified as either an update or a user transaction. Update transactions arrive periodically and may only write to base (temporal) data objects. User transactions arrive aperiodically and may read temporal and read/write nontemporal data. In this model, denoted the *utilization transaction model*, each transaction has the following characteristics:

- the period p_i (update transactions),
- the estimated mean interarrival time $r_{\mathrm{E},i}$ (user transactions),
- the actual mean interarrival time $r_{\mathrm{A},i}$ (user transactions),
- the estimated execution time $x_{\mathrm{E},i}$,
- the actual execution time $x_{\mathrm{A},i}$,
- the relative deadline d_i,
- the estimated utilization[4] $u_{\mathrm{E},i}$, and
- the actual utilization $u_{\mathrm{A},i}$.

Table 1 presents the complete utilization transaction model. Upon arrival, a transaction presents the estimated average utilization $u_{\mathrm{E},i}$ and the relative deadline d_i to the system. The actual utilization of the transaction $u_{\mathrm{A},i}$ is not known in advance due to variations in the execution time.

[4] Utilization is also referred to as load.

Table 1. The utilization transaction model

Attribute	Periodic transactions	Aperiodic transactions
d_i	$d_i = p_i$	$d_i = r_{\mathrm{A},i}$
$u_{\mathrm{E},i}$	$u_{\mathrm{E},i} = x_{\mathrm{E},i}/p_i$	$u_{\mathrm{E},i} = x_{\mathrm{E},i}/r_{\mathrm{E},i}$
$u_{\mathrm{A},i}$	$u_{\mathrm{A},i} = x_{\mathrm{A},i}/p_i$	$u_{\mathrm{A},i} = x_{\mathrm{A},i}/r_{\mathrm{A},i}$

5.4 A Family of COMET QoS Configurations

We developed a QoS aspect package for COMET to enable the database to be used in applications that have uncertain workloads and where requirements for data freshness are essential, e.g., a new generation of vehicle control systems [3]. The aspects within the package are implemented using AspectC++ [50]. The COMET QoS aspect package consists of the actuator and controller components and the following aspects:

- QoS management policy aspects: actuator utilization policy, missed deadline monitor, missed deadline controller, scheduling strategy, data access monitor, QoS through update scheduling aspect, self-tuning regulator aspect, and adaptive regression model aspect;
- QoS transaction and data model aspects: utilization transaction model aspect and data differentiation aspect; and
- QoS composition aspects: actuator composition and controller composition aspects.

Figure 6 shows the family of COMET feedback-based QoS configurations. Each configuration is obtained by adding aspects and possibly components from the package to the previous configuration. The initial configuration is the concurrent COMET configuration. Table 2 indicates which elements of the QoS aspect package are used in different configurations.

The tool described in Sect. 4.3 can be used for specifying the QoS requirements of an application with which COMET is to be integrated. Once the requirements are specified, the tool suggests the subset of aspects and components from the aspect package that can be used for making a COMET QoS configuration. The tool decides on the relevant subset of components and aspects by cross-examining the requirements with the previously developed QoS modeling paradigms for the domain of embedded real-time databases.

To simplify the presentation, in the following we discuss only five distinct configurations that provide admission control, FC-M, QMF, self-tuning, and least squares regression QoS. Note however that depending on the chosen aspects and components from the package, the number of variants in COMET QoS family is higher (see Fig. 6).

The admission control configuration includes one component from the QoS aspect package, the actuator. The configuration also requires the actuator composition aspect to ensure adding the actuator to the controlled system, and the utilization transaction model aspect to extend the transaction model (see Table 2). This configuration is simple as it only provides facilities for admission control.

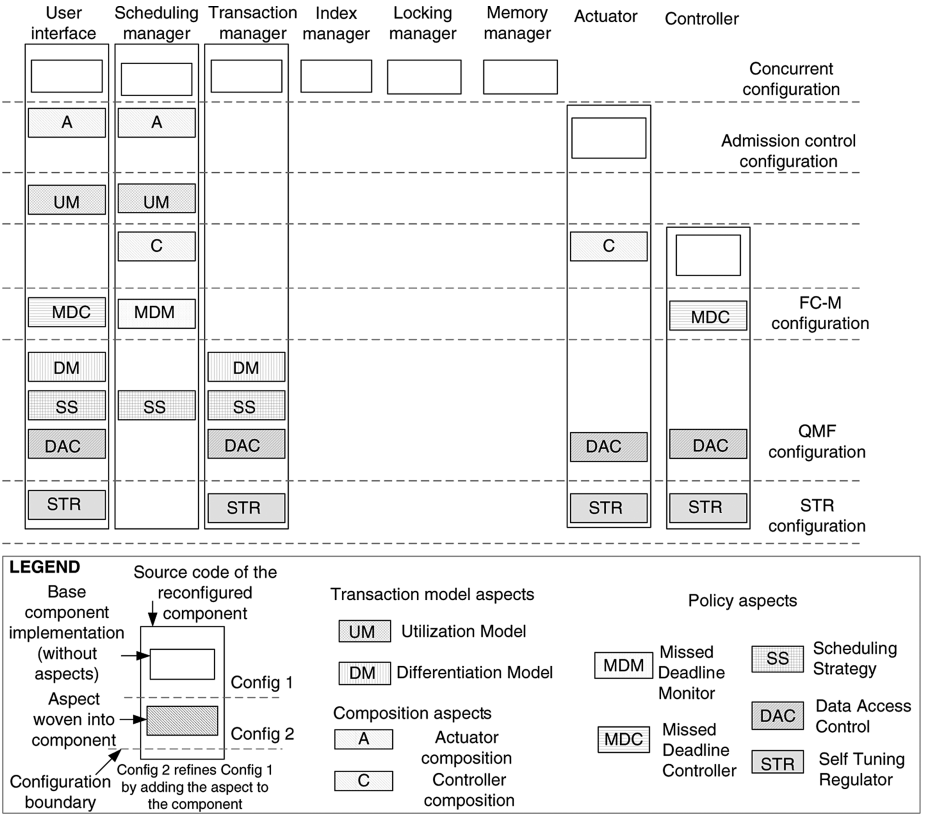

Fig. 6. Creating a family of real-time systems from the COMET QoS aspect package

The miss ratio feedback configuration (COMET FC-M) provides QoS guarantees based on the FC-M policy. The configuration includes the actuator and controller components and their corresponding composition aspects, the utilization transaction model aspect, the missed deadline monitor aspect, and the missed deadline controller aspect (see Table 2). These aspects modify the policy of the scheduling manager component and the controller to ensure that QoS with respect to deadline misses is enforced.

The update scheduling configuration (COMET QMF) provides the QoS guarantees based on the QMF policy. Here the data differentiation aspect and scheduling strategy aspect are used to further enrich the transaction model. Moreover, the data access monitor aspect is required to ensure the metric used in QMF, and the QoS through update scheduling aspect to further adjust the policy of the actuator to suit the QMF algorithm.

The self-tuning regulator configuration (COMET STR) provides adaptive QoS control where the control algorithm parameters are adjusted by using

Table 2. The COMET QoS aspect package constituents and their usage in various COMET QoS configurations

QoS aspect package		COMET configurations				
		Admission control	COMET FC-M	COMET QMF	COMET STR	COMET RM
Policy aspects	Actuator utilization policy	X	X	X	X	X
	Missed deadline monitor		X	X	X	X
	Missed deadline controller		X	X	X	X
	Scheduling strategy			X		
	Data access monitor			X		
	QoS through update scheduling			X		
	Self-tuning regulator				X	
	Adaptive regression model					X
Transaction model aspects	Utilization transaction model	X	X	X	X	X
	Data differentiation			X		
Composition aspects	Actuator composition	X	X	X	X	X
	Controller composition		X	X	X	X
Components	Actuator	X	X	X	X	X
	Controller		X	X	X	X

X in the table means that an aspect (or a component) is a part of a configuration
COMET FCM the miss ratio feedback configuration; *COMET STR* the self-tuning regulator configuration
COMET QMF the update scheduling configuration; *COMET RM* the regression model configuration

the self-tuning regulator. This aspect is added to the aspects and components constituting the COMET FC-M configuration to ensure the adaptability of the control already provided by the COMET FC-M configuration.

The regression model configuration (COMET RM) provides the adaptive QoS control where the control algorithm parameters are adjusted by using the least squares technique and the regression model. This aspect also requires all the aspects needed for the FC-M configuration to ensure adaptability of QoS management.

5.5 Implementation Details

To illustrate the way aspects and components can be implemented within an aspect package, in this section we elaborate on the implementation of the main constituents of the COMET FC-M configuration. Recall that the FC-M configuration includes the actuator and controller components, composition aspects for these components, as well as the utilization transaction model aspect, the missed deadline monitor aspect, and the missed deadline controller aspect.

The actuator is a component that, based on an admission policy, decides whether to allow new transactions into the system. Operations provided by the actuator are `admit()`, which performs the admission test, and `adjust()`, which adjusts the number of transactions that can be admitted. The default admission policy is to allow all transactions to be admitted to the system. This admission policy of the actuator can be changed by weaving specific QoS actuator policy aspects.

The controller is a component that computes the input to the admission policy of the actuator at regular intervals. By default, an input of zero is generated, but by using QoS controlling policy aspects various feedback control policies can be used. The controller provides only one operation, `init()`, that initializes the controller component. The controller calls `adjust()` operation of the actuator after computing the manipulated variable.

The *utilization transaction model aspect* augments the basic COMET transaction model so that it suits the utilization transaction model described in Sect. 5.3. This is done using an intertype declaration that adds new parameters to the basic model, e.g., estimated utilization $u_{\mathrm{E},i}$ and estimated execution time $x_{\mathrm{E},i}$.

```
1: aspect actuator_composition{
2:   // Insert actuator between user interface and scheduler.
3:   advice call("bool Scheduler_CreateNew(...)") : around() {
4:     if (Actuator_Admit(*(scheduleRecord *)tjp->arg(0)))
5:       tjp->proceed();
6:     else
7:       *(bool *)tjp->result() = false;
8:   }
9: };
```

Fig. 7. The actuator composition aspect

The *actuator composition aspect* enables the actuator to intercept requests to create new transactions that are posed by the user interface to the scheduling manager component. This is done via an advice of type around which is executed when the operation `createNew()` of the scheduler manager is called (lines 3–8 in Fig. 7). Since this operation of the scheduler manager component is in charge of registering a new transaction to the system, the advice ensures that an admission test is made by the actuator before the transaction is actually registered (line 4). If the transaction can be admitted, transaction registration is resumed; the `proceed()` in line 5 enables the normal continuation of the joinpoint `createNew()` (an explicitly declared joinpoint in the scheduling manager component). If the transaction is to be aborted, then the around advice replaces the execution of the transaction registration in full and, thus, ensures that the transaction is rejected from the system (line 7).

The *actuator utilization policy aspect* shown in Fig. 8 replaces, via the around advice (lines 5–13), the default admission policy of actuator with an admission policy based on utilization (lines 9–12). The current transaction examined for admission in the system is denoted `ct` in Fig. 8.

```
1: aspect actuator_utilization_policy{
2:   // Add a utilization reference to the system
3:   advice "SystemParameters" : float utilizationRef;
4:   // Changes the policy of the actuator to the utilization
5:  advice execution("% Actuator_Admit(...)") : around() {
6:    // Get the current estimated total utilization
7:    totalUtilization = GetTotalEstimatedUtilization();
8:   // Check if the current transaction ct can be admitted
9:   if (utilizationTarget > totalUtilization + ct->utilization)
10:    { (*(bool *)tjp->result()) = true; }
11:      else
12:    {  (*(bool *)tjp->result()) = false; }
13:  }
```

Fig. 8. The actuator utilization policy aspect

The *controller composition aspect* facilitates the composition of the controller with all other components in the system by ensuring that the controller is properly initialized during the system initialization.

```
1: aspect missed_deadline_monitor {
2:   advice call("% Scheduler_CreateNew(...)") : after(){
3:       if (*(bool *)tjp->result()) { admittedTransactions++; }
4:   }
5:   advice call("% Scheduler_Completed(...)") : before(){
6:     ScheduleRecord *sr = (ScheduleRecord *)tjp->arg(0);
7:      _getTime(&currentTime);
8:      node = findNode(ActiveQueue_root, sr->id);
9:      if ((node != NULL) && (_compareTimes(&currentTime,
10:                              &(node->data->deadline))))
11:       { missedTransactions++; }
12:  }
13:  advice call("% Scheduler_Aborted(...)") : before(){…
14:     admittedTransactions--;}
15:  advice call("% Scheduler_RejectLeastValuableTransaction(...)") :  after(){
16:     if (*(bool *)tjp->result()) { admittedTransactions--;}
17:  }
18:  advice call("% getTimeToDeadline(...)") && within("%
19:    getNextToExecute(...)") : after() {… missedTransactions++;}
20: }
```

Fig. 9. Missed deadline monitor aspect

The *missed deadline monitor aspect* modifies the scheduling manager component to keep track of transactions that have missed their deadlines, $missedTransactions$, and transactions that have been admitted to the system, $admittedTransactions$. This is done by having a number of advices of different types intercepting operations of the scheduling manager component that handles completion and abortion of transactions (see Fig. 9). For example, the advice of type after that intercepts the call to `createNew()` increments the number of admitted

transactions once the transactions have been admitted to the system (lines 2–4). Similarly, before the transaction has completed, the advice in lines 5–12 checks if the number of transactions with missed deadlines should be incremented, i.e., before invoking the operation `completed()` of the scheduler manager component.

```
1: aspect missed_deadline_control{
2:   // Initialize the new variables need for contrb
3:   advice call("% UserInterface_init(...)") : after() {
4:         SystemParameters *sp =
5:                 (SystemParameters *)tjp->arg(0);
6:     if (*(bool *)tjp->result()) {
7:      missRatioReference = sp->missRatioReference;
8:      missRatioControlVariableP =
9:                 sp->missRatioControlVariableP;
10:      ...
11:   }
12:   // Modify the calculation of the control output
13:   advice call("% calculateOutput(...)") : after(){
14:     missRatioOutputHm =
15:       calculateMissRatioOutput(Scheduler_GetDeadlineMissRatio());
16:    *((float *)tjp->result()) = missRatioOutputHm;
17:   }
18: }
```

Fig. 10. Missed deadline controller aspect

The *missed deadline controller aspect*, illustrated in Fig. 10, is an instance of the feedback control policy aspect and it modifies the scheduler manager component to keep track of the deadline miss ratio, using Eq. (1). The aspect does so with two advices of type after. One is executed after the initialization of the user interface (lines 3–11), thus, ensuring that the appropriate variables needed for controller policy are initialized. The other modifies the output of the controller to suit the chosen feedback control policy, which is deadline miss ratio in this case (lines 13–17).

5.6 Experimental Evaluation

In this section we present an experiment made on the COMET platform with and without the QoS aspect package. The goal of the experiment is to show that QoS management in COMET performs as expected and thereby show that, when adding the COMET QoS aspect package, we indeed achieve configurability in QoS management with required performance guarantees. It should be noted that we have performed several other experiments to show that we achieve the desired behavior under different COMET QoS configurations [51].

For doing the experiment we have chosen the following experimental setup. The database consists of eight relations, each containing ten tuples. Note that this relatively low data volume is acceptable for the experiments as the experimental results do not depend on the data volume but the load, i.e., number

of transactions, that is imposed on the database. To that end, we ensured that a constant stream of transaction requests is used in the experiments. Update transactions arrive periodically, whereas user transactions arrive aperiodically. To vary the load on the system, the interarrival times between transactions are altered. The deadline miss ratio reference, i.e., the desired QoS, is set to 0.1.

The experiment is applied to the COMET FC-M configuration, where the load applied on the database is varied. This way we can determine the behavior of the system under increasing load. We use the behavior of the concurrent COMET configuration without the QoS aspect package as a baseline. For all the experiment data, we have taken the average of 10 runs, each consisting of 1,500 transactions. We have derived 95% confidence intervals based on the samples obtained from each run using the *t*-distribution [52]. We have found that the ten runs are sufficient as we have obtained tight confidence intervals (shown later in this section). Figure 11 shows the deadline miss ratio of concurrent COMET and COMET with the FC-M configuration. The dotted line indicates the deadline miss ratio reference. We vary the applied load from 0 to 130%. This interval of load captures the transition from an underloaded system to an overloaded system since at 130% applied load we capture the case when the system is overloaded. We refer to [5, 6] for extensive evaluations of how loads greater than 130% affect the performance of the system.

Starting with concurrent COMET, the deadline miss ratio starts increasing at approximately 0.85 load. However, the deadline miss ratio increases more than the desired deadline miss ratio and, hence, concurrent COMET does not provide any QoS guarantees. Studying the results obtained when using the FC-M configuration we see that the deadline miss ratio for loads $0.90, \ldots, 1.30$

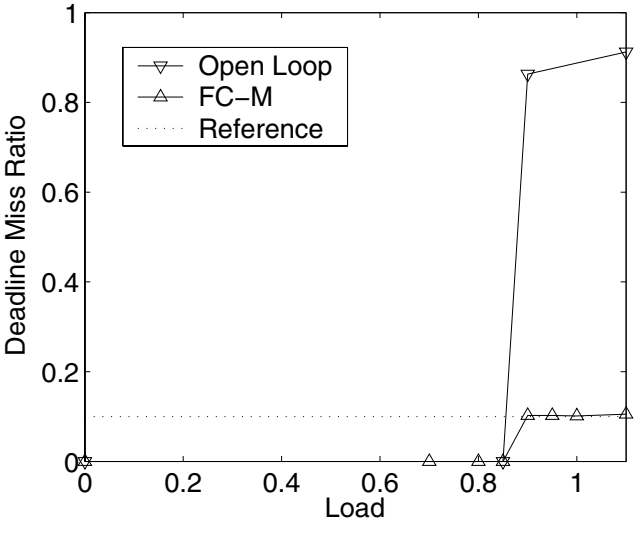

Fig. 11. Deadline miss ratio as a function of load

are $0.1025 \pm 0.0070, 0.1023 \pm 0.0027, 0.1012 \pm 0.0025, 0.1052 \pm 0.0030$, and 0.1082 ± 0.0011. In contrast with concurrent COMET, the added FC-M configuration manages to keep the deadline miss ratio at the reference, even during high loads. This is in line with our earlier observations where feedback control has shown to be very effective in guaranteeing QoS [5, 53, 54]. Hence, the results of our experiment show that the COMET FM-C configuration is able to provide QoS guarantees under varying load.

6 Experience Report

This section contains observations we made with respect to use of aspect-oriented software development in general and the QoS aspect package in particular for ensuring configurability and reusability of QoS management in real-time software.

Lesson 1: *Both aspects and components are needed in the QoS aspect package to ensure configurability.* We already observed in Sect. 1 that many QoS management approaches use a similar QoS infrastructure but provide distinct QoS policies, concluding that this is the reason why a QoS aspect package has both components (providing infrastructure) and aspects (providing QoS policies). Here we would like to reaffirm that without components in a QoS aspect package high configurability and reusability would be difficult to achieve. Namely, if all QoS management approaches were implemented only using aspects, each aspect would have to contain the necessary functionality of the infrastructure for QoS management. This is obviously not conducive to the development of families of real-time systems with distinct needs on the QoS configuration. Furthermore, the footprint of the system would be increased with additional code. Another solution would be to implement one basic aspect that would contain the functionality of an infrastructure and possibly a QoS policy. However, this option is not desirable as it implies dependencies among aspects that could lead to dependency problems in the implementation phase. This in turn could induce decreased configurability (as fewer aspects could be combined). Having components that provide functionality used by aspects decreases dependency issues and the overall memory footprint for the system, and increases reusability of aspects and configurability of QoS management.

Lesson 2: *Explicitly declared joinpoints in component interfaces lead to an efficient and analyzable product-line architecture.* We have observed that for developing a variety of system configurations using the same set of components in combination with various aspects, it is beneficial to have explicitly defined places in the architecture where extensions can be made, i.e., aspects woven. Although this approach restricts the joinpoint model of the aspect language, we obtain clear points in the components and the system architecture where variations can be made. The system developer extending and reconfiguring an existing system now does not need to have a complete insight into the system or component internals to perform successful reconfiguration. Therefore, in our components within a QoS aspect package we enforce that the places where pos-

sible extensions can be done are explicitly declared in the component interfaces. It is our experience, confirmed by the experiences of third-party COMET users, that these points are not difficult to identify in the design and implementation phases of component development. For example, the declared joinpoints for the actuator component and the controller component from the COMET QoS aspect package were straightforwardly determined in the design phase by taking into consideration a few possible policies that could be applied to these components.

Hence, relevant joinpoints in the components should be identified in the design phase of the QoS aspect package development with regard to possible policy variations. In development of the COMET database we have experienced that these points are relatively low in number and, once identified, are even suitable for aspects that are developed later on. Moreover, the explicitly declared joinpoints in the component code are desirable in the real-time domain. This is because they provide predefined places where code modifications can be done and, therefore, the system can be analyzed during the design phase to establish if it satisfies temporal constraints and has desired functional characteristics [16, 46]. We can conclude that by using the notion of a QoS aspect package we can efficiently develop an analyzable product-line architecture of a real-time system that has the ability to satisfy specified QoS needs.

Lesson 3: *There is a tradeoff between configurability, reusability, and maintenance.* Having a large number of aspects leads to high demands on maintainability of the aspects and the system, while fewer aspects lead to better maintainability at the expense of limiting configurability and reusability of aspects in the system. This further confirms previous observations [55] where a tradeoff between requirements for configurability and maintenance when using aspects in embedded software systems was identified. In the case of developing a QoS aspect package for COMET, our primary goal was to ensure reuse of QoS-related aspects and to increase system configurability. Therefore, we have chosen to separate concerns such that we have a great number of aspects that each can be used in multiple COMET configurations. For example, the missed deadline monitor aspect is decoupled from the missed deadline controller aspect (both are part of the QoS policy aspects and implement FC-M policy) to allow the missed deadline monitor aspect to be used in a combination with another controller policy. In the case when there is a focus on maintainability, the missed deadline monitor aspect and the missed deadline control aspect could be combined into one aspect that both monitors and controls the deadline misses. The same is true for the scheduling strategy aspect and the QoS through the update scheduling aspect that both implement parts of the QMF algorithm. We have chosen to have these in different aspects to enable them to be exchanged independently from the configuration and from each other.

Hence, if reusability and configurability is of foremost concern, as it is typically the case in the context of creating families of real-time systems, QoS policies should be decomposed into greater number of aspects; thus, trading maintainability for reusability and configurability. To deal with maintainability issues, an

efficient way of organizing aspects and components for easier access and modification within a QoS aspect package is needed.

Lesson 4: *Aspects can be reused in various phases of the system development.* We found that aspects implemented in a QoS aspect package can be reused in various phases of the system development. Namely, due to the nature of QoS policies, one or several aspects constituting a policy normally control the load of the system and in some way monitor the system performance. Hence, in addition to reuse of these aspect in a number of QoS configurations, they can be reused in the testing and evaluation phase for performance evaluation and gathering statistics. As a simple illustration, the missed deadline monitor aspect within the COMET QoS aspect package is used in the design and implementation phase of the system as a part of a QoS management to implement a specific QoS policy, and is later used in the evaluation phase of the system development for performance evaluations (presented in Sect. 5.6).

Lesson 5: *Aspect languages are a means of dealing with legacy software.* Since we initially targeted the COMET database system to hard real-time systems in the vehicular industry [4], the programming language used for the development of the basic database functionality (described in Sect. 5.1) needed to be suited for software that already existed in a vehicular control system. Moreover, analysis techniques that have been used in the existing vehicle control system should be applicable to our basic database components. This lead to developing the COMET basic configuration using C programming language. Aspects provided an efficient means for introducing extensions to the system; we used the AspectC++ weaver since a weaver for C language [56] is not yet publicly available. Since existing real-time systems are typically developed in a non-object-oriented language such as C, aspects provide a great value for evolution of the system without reconstructing the code of the system.

Lesson 6: *Less is more when it comes to aspect languages for embedded and real-time systems.* When developing aspect languages for real-time systems operating in resource-constrained environments, the focus should be on providing basic aspect language features that facilitate encapsulating and weaving aspects into the code of components in the simplest and most memory-efficient way possible. We believe that minimizing memory usage should be of primary importance for aspect languages suitable for these types of systems. Given that most real-time computing systems are developed using non-object-oriented languages, the intertype declaration could be kept as simple as possible, e.g., allowing weaving of single members in structs. We also observe that due to the nature of many real-time operating systems, e.g., Rubus [57] and MicroC [58], the advice and pointcut model could be simplified. Namely, the pointcut syntax in most cases does not need to be elaborate as it is in current aspect languages (e.g., AspectJ and AspectC++). We have developed most of the COMET aspects using `call` and `execution` pointcuts, and occasionally `within`.

7 Summary

In this paper we have presented the concept of a QoS aspect package that enables reconfigurability in QoS management of real-time systems. A QoS aspect package facilitates the development of real-time systems with a flexible QoS management architecture consisting of aspects and components, where parts of the architecture can be modified, changed, or added depending on the target application's QoS requirements. Furthermore, QoS policies within a QoS aspect package are encapsulated into aspects and can be exchanged and modified independently of the real-time system. This improves reusability of QoS management and ensures applicability across many applications. Applying the concept of a QoS aspect package enables existing real-time systems, without QoS guarantees, to be used in applications that require specific performance guarantees. By exchanging aspects within the QoS aspect package one can efficiently tailor QoS management of a real-time system based on the application requirements. We have shown how the concept can be applied in practice by describing the way we have adapted the COMET database platform. Initially, COMET was developed without mechanisms for QoS guarantees, but by adding the QoS aspect package COMET has been extended to support a variety of QoS policies.

Acknowledgments

The authors would like to thank anonymous reviewers for valuable comments on the manuscript. This work is financially supported by the Swedish National Graduate School in Computer Science (CUGS) and the Center for Industrial Information Technology (CENIIT) under contract 01.07.

References

[1] Chen X., Cheng A.M.K. An imprecise algorithm for real-time compressed image and video transmission. In: *Proceedings of the International Conference on Computer Communications and Networks (ICCCN)*, pp. 390–397, 1997

[2] Curescu C., Nadjm-Tehrani S. Time-aware utility-based resource allocation in wireless networks. *IEEE Transactions on Parallel and Distributed Systems*, 16:624–636, 2005

[3] Gustafsson T., Hansson J. Data management in real-time systems: a case of on-demand updates in vehicle control systems. In: *Proceedings of Tenth IEEE Real-Time Applications Symposium (RTAS'04)*, IEEE Computer Society Press, pp. 182–191, 2004

[4] Nyström D., Tešanović A., Norström C., Hansson J., Bånkestad N.E. Data management issues in vehicle control systems: a case study. In: *Proceedings of the 14th IEEE Euromicro International Conference on Real-Time Systems (ECRTS'02)*, IEEE Computer Society Press, pp. 249–256, 2002

[5] Amirijoo M., Hansson J., Son S.H., Gunnarsson S. Robust quality management for differentiated imprecise data services. In: *Proceedings of the 25th IEEE International Real-Time Systems Symposium (RTSS'04)*, IEEE Computer Society Press, pp. 265–275, 2004

[6] 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 11th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'05)*, IEEE Computer Society Press, pp. 2–11, 2005

[7] Buttazzo G.C. Hard Real-Time Computing Systems, Kluwer Academic, Dordrecht, 1997

[8] Lu C., Stankovic J.A., Tao G., Son S.H. Feedback control real-time scheduling: framework, modeling and algorithms. *Journal of Real-Time Systems*, 23, 2002

[9] Lu Y., Saxena A., Abdelzaher T.F. Differentiated caching services: a control-theoretical approach. In: *Proceedings of the 21st IEEE International Conference on Distributed Computing Systems (ICDCS'01)*, IEEE Computer Society Press, pp. 615–622, 2001

[10] Parekh S., Gandhi N., Hellerstein J., Tilbury D., Jayram T., Bigus J. Using control theory to achieve service level objectives in performance management. *Journal of Real-Time Systems*, 23, 2002

[11] Cervin A., Eker J., Bernhardsson B., Årzén K. Feedback-feedforward scheduling of control tasks. *Real-Time Systems Journal*, 23, 2002, Special Issue on Control-Theoretical Approaches to Real-Time Computing

[12] Li B., Nahrstedt K. A control theoretical model for quality of service adaptations. In: *Proceedings of the Sixth IEEE International Workshop on Quality of Service*, pp. 145–153, 1998

[13] Stankovic J. VEST: a toolset for constructing and analyzing component based operating systems for embedded and real-time systems. In: *Proceedings of the First International Conference on Embedded Software, (EMSOFT'01), LNCS vol. 2211*, Springer, Berlin Heidelberg New York, pp. 390–402, 2001

[14] Tesanovic A., Amirijoo M., Nilsson D., Norin H., Hansson J. Ensuring real-time perfomance guarantees in dynamically reconfigurable embedded systems. In: *Proceedings of the IFIP International Conference on Embedded and Ubiquitous Computing, LNCS vol. 3824*, Springer, Berlin Heidelberg New York, pp. 131–141, 2005

[15] Nyström D., Tešanović A., Nolin M., Norström C., Hansson J. COMET: a component-based real-time database for automotive systems. In: *Proceedings of the IEEE Workshop on Software Engineering for Automotive Systems*, pp. 1–8, 2004

[16] 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

[17] 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'03)*, pp. 152–159, 2003

[18] van Ommering R. Building product populations with software components. In: *Proceedings of the 24th International Conference on Software Engineering*, ACM, New York, pp. 255–265, 2002

[19] 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

[20] 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 (CBSE7)*, Scotland, Springer, Berlin Heidelberg New York, pp. 194–208, 2004

[21] 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 (AOSD'05)*, ACM, New York, pp. 39–50, 2005

[22] Amirijoo M., Hansson J., Son S.H., Gunnarsson S. Generalized performance management of multi class real-time imprecise data services. In: *Proceedings of the 26th IEEE International Real-Time Systems Symposium (RTSS'05)*, pp. 38–49, 2005

[23] Abdelzaher T.F., Shin K.G., Bhatti N. Performance guarantees for web server end-systems: a control-theoretical approach. *IEEE Transactions on Parallel and Distributed Systems*, 13:80–96, 2002

[24] Koutsoukos X., Tekumalla R., Natarajan B., Lu C. Hybrid supervisory utilization control of real-time systems. In: *Proceedings of 11th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'05)*, IEEE Computer Society Press, pp. 12–21, 2005

[25] Liu J.W.S., Shih W.K., Lin K.J., Bettati R., Chung J.Y. Imprecise computations. IEEE Comput, 82, 1994

[26] Amirijoo M., Hansson J., Son S.H. Algorithms for managing QoS for real-time data services using imprecise computation. In: *Proceedings of the Conference on Real-Time and Embedded Computing Systems and Applications (RTCSA'03), LNCS vol. 2968*, Springer, Berlin Heidelberg New York, pp. 136–157, 2004

[27] Abdelzaher T.F., Stankovic J.A., Lu C., Zhang R., Lu Y. Feedback performance control in software services. *IEEE Control Systems Magazine*, 23:74–90, 2003

[28] Sha L., Liu X., Lu Y., Abdelzaher T. Queuing model based network server performance control. In: *Proceedings of 23rd IEEE Real-Time Systems Symposium (RTSS'02)*, IEEE Computer Society Press, 2002

[29] Abdelzaher T., Lu Y., Zhang R., Henriksson D. Practical application of control theory to web services. In: *Proceedings of American Control Conference (ACC)*, 2004

[30] Robertson A., Wittenmark B., Kihl M. Analysis and design of admission control in web-server systems. In: *Proceedings of American Control Conference (ACC)*, 2003

[31] Zhu Y., Mueller F. Feedback EDF scheduling exploiting dynamic voltage scaling. In: *Proceedings of the Tenth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'04)*, IEEE Computer Society Press, pp. 84–93, 2004

[32] Franklin G.F., Powell J.D., Workman M. Digital Control of Dynamic Systems, 3rd edn., Addison-Wesley, New York, 1998

[33] Åström K.J., Wittenmark B. Adaptive Control, 2nd edn., Addison-Wesley, New York, 1995

[34] Kang K.D., Son S.H., Stankovic J.A. Managing deadline miss ratio and sensor data freshness in real-time databases. *IEEE Transactions on Knowledge and Data Engineering*, 16:1200–1216, 2004

[35] Kang K.D., Son S.H., Stankovic J.A. Service differentiation in real-time main memory databases. In: *Proceedings of the Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, IEEE Computer Society Press, pp. 119–128, 2002

[36] Sharma V., Thomas A., Abdelzaher T., Skadron K., Lu Z. Power-aware QoS management in web servers. In: *Proceedings of 24th IEEE Real-Time Systems Symposium (RTSS)*, IEEE Computer Society Press, pp. 63–71, 2003

[37] Sanfridson M. Problem formulations for qos management in automatic control. Technical Report TRITA-MMK 2000:3, ISSN 1400-1179, ISRN KTH/MMK–00/3–SE, Mechatronics Lab KTH, Royal Institue of Technology (KTH), Sweden, 2000

[38] Schantz R., Loyall J., Atighetchi M., Pall P. Packaging quality of service control behaviors for reuse. In: *Proceedings of the Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'02)*, Washington, DC, USA, IEEE Computer Society, pp. 375, 2002

[39] Wang N., Gill C., Schmidt D.C., Subramonian V. Configuring real-time aspects in component middleware. In: *Proceedings of the OTM Confederated International Conferences, LNCS vol. 3291*, Springer, Berlin Heidelberg New York, pp. 1520–1537, 2004

[40] Duzan G., Loyall J., Schantz R., Shapiro R., Zinky J. Building adaptive distributed applications with middleware and aspects. In: *Proceedings of the Third ACM International Conference on Aspect-Oriented Software Development (AOSD'04)*, New York, NY, USA, ACM, pp. 66–73, 2004

[41] Gokhale A.S., Schmidt D.C., Lu T., Natarajan B., Wang N. CoSMIC: an MDA generative tool for distributed real-time and embedded applications. In: *Proceedings of the First Workshop on Model-Driven Approaches to Middleware Applications Development (MAMAD 2003)*, pp. 300–306, 2003

[42] Isovic D., Norström C. Components in real-time systems. In: *Proceedings of the Eighth IEEE International Conference on Real-Time Computing Systems and Applications (RTCSA'02)*, Tokyo, Japan, pp. 135–139, 2002

[43] Liu C.L., Layland J.W. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20:46–61, 1973

[44] Tešanović A., Mu P., Hansson J. Development environment for configuration and analysis of embedded real-time systems. In: *Proceedings of the Fourth International Workshop on Aspects, Components, and Patterns for Infrastructure Software (ACP4IS'05)*, 2005

[45] GME. The generic modeling environment. Institute for Software Integrated Systems, Vanderbilt University, `http://www.isis.vanderbilt.edu/Projects/gme/`, 2004

[46] Tešanović A., Nadjm-Tehrani S., Hansson J. Modular verification of reconfigurable components. In: *Component-Based Software Development for Embedded Systems – An Overview on Current Research Trends, LNCS vol. 3778*, Springer, Berlin Heidelberg New York, pp. 59–81, 2005

[47] Liu C.L., Layland J.W. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20:46–61, 1973

[48] Kang K.D., Son S.H., Stankovic J.A., Abdelzaher T.F. A QoS-sensitive approach for timeliness and freshness guarantees in real-time databases. In: *Proceedings of the 14th IEEE Euromicro Conference on Real-Time Systems (ECRTS'02)*, IEEE Computer Society Press, pp. 203–212, 2002

[49] Ramamritham K. Real-time databases. *International Journal of Distributed and Parallel Databases*, 1993

[50] Spinczyk O., Gal A., Schröder-Preikschat W. AspectC++: an aspect-oriented extension to C++. In: *Proceedings of the 40th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS'02)*, Sydney, Australia, Australian Computer Society, 2002

[51] Björk M. QoS management in configurable real-time databases. Master's thesis, Department of Computer Science, Linköping University, Sweden, 2004

[52] DeGroot M.H., Schervish M.J. Probability and Statistics, 3rd edn., Addison-Wesley, New York, 2002
[53] Amirijoo M., Hansson J., Son S.H. Error-driven QoS management in imprecise real-time databases. In: *Proceedings of the 15th IEEE Euromicro Conference on Real-Time Systems (ECRTS'03)*, IEEE Computer Society Press, pp. 63–72, 2003
[54] Amirijoo M., Hansson J., Son S.H. Specification and management of QoS in imprecise real-time databases. In: *Proceedings of the Seventh IEEE International Database Engineering and Applications Symposium (IDEAS'03)*, IEEE Computer society Press, pp. 192–201, 2003
[55] Tešanović A., Sheng K., Hansson J. Application-tailored database systems: a case of aspects in an embedded database. In: *Proceedings of the Eighth IEEE International Database Engineering and Applications Symposium (IDEAS'04)*, IEEE Computer Society, pp. 291–301, 2004
[56] Coady Y., Kiczales G. Back to the future: a retroactive study of aspect evolution in operating system code. In: *Proceedings of the Second ACM International Conference on Aspect-Oriented Software Development (AOSD'03)*, Boston, USA, ACM, pp. 50–59, 2003
[57] Articus Systems. Rubus OS – Reference Manual, 1996
[58] Labrosse J.J. MicroC/OS-II the Real-Time Kernel, CMPBooks, 2002