Empowering Configurable QoS Management in Real-Time Systems*

Aleksandra Tesanovic, Mehdi Amirijoo, Mikael Björk, and Jörgen Hansson
Department of Computer Science
Linköping University
Linköping, Sweden
{alete,meham,jorha}@ida.liu.se

ABSTRACT

Current Quality of Service (QoS) management methods in real-time systems using feedback control loop lack support for configurability and reusability as they cannot be configured for a target application or reused across different applications. In this paper we present a method for developing reconfigurable feedback-based QoS management for real-time systems, denoted Re-QoS. By combining component-based design with aspect-oriented software development Re-QoS enables successful handling of crosscutting nature of QoS policies, as well as evolutionary design of real-time systems and QoS management architectures. Re-QoS defines a QoS aspect package, which is an implementation of a set of aspects and components that provide a number of different QoS policies. By adding a QoS aspect package to an existing system without QoS guarantees, we are able to use the same system in unpredictable environments where performance guarantees are essential. Furthermore, by exchanging aspects within the QoS aspect package one can efficiently tailor the QoS management of a real-time system based on the application requirements. We demonstrate the usefulness of the concept on a case study of an embedded real-time database system, called COMET. Using the COMET example we show how a real-time database system can be adapted to be used in different applications with distinct QoS needs.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures; D.3.3 [**Programming Languages**]: Language Constructs and Features—*Aspects, Components*

*This work is supported by the Swedish Foundation for Strategic Research (SSF) via the SAVE project (SAfety critical component-based VEhicular systems), the Swedish National Graduate School in Computer Science (CUGS), and the Center for Industrial Information Technology (CENIIT) under contract 01.07

General Terms

Design, Performance, Languages

1. INTRODUCTION

A large majority of computational activities in modern society are performed within embedded and real-time systems. Real-time systems are typically constructed out of concurrent programs, called tasks, due to the inherent nature of the environment with which these systems interact. The most common type of temporal constraint that a real-time system must satisfy is the completion of task deadlines. There exist a number of real-time scheduling techniques to ensure that tasks meet their respective deadlines (see [8]), and these typically require the knowledge of the worst-case execution time of a task.

Depending on the consequences of a missed deadline, real-time systems can be classified as hard or soft. In a *hard real-time system* consequences of missing a deadline can be catastrophic, while in a *soft real-time system* missing a deadline does not cause catastrophic damage to the system, but may affect performance negatively.

In recent years, the domain of real-time computing has expanded from hard real-time application areas, e.g., avionics and automotive applications, where performance relies on worst-case guarantees (so-called closed applications) to soft application areas operating in open and unpredictable environments where arrival patterns and resource requirements of tasks are generally unknown, e.g., mobile computing systems, web-servers, and e-commerce. Further, these applications are becoming more complex and at the same time performance guarantees are required.

Traditional approaches providing hard real-time guarantees rely on worst-case execution times and worst-case arrival patterns of tasks, and are not effective for a large class of soft real-time systems as they result in highly underutilized systems. Feedback control has been introduced as a promising foundation for performance control of real-time systems that are both resource insufficient and exhibit unpredictable workloads [5, 2, 18, 19, 22, 9, 16]. Feedback-based QoS management is attractive as it enables the designer or system operator to explicitly specify the performance of the system in terms of desired steady-state and transient-state system performance.

In this paper we use the following terminology. A *QoS management architecture* denotes the way the feedback control structure is implemented with the real-time system. A *QoS management policy* refers to the way the system is controlled and performance guaranteed. A *QoS management method* refers to both a QoS management architecture and a policy as well as the relationship between the two. When the distinction between the QoS management architecture, policy, and method is not of interest the simple term QoS management is used.

We observe that existing feedback-based QoS management methods for real-time systems [5, 2, 18, 19, 22, 9, 16], do not address some of the most important QoS design challenges, such as [11, 6]:

- enabling configurability in QoS management by allowing the designer to choose the QoS policy depending on the application requirements; and
- supporting reusability of QoS policies and architectures across application areas.

The requirements for configurability and reusability in real-time computing systems have been addressed in approaches that combine component-based software development technologies with real-time systems, e.g., [34, 27, 29, 25]. These approaches provide configurability of the system by enabling a system to be developed out of pre-defined software components with well defined interfaces. These, however, do not provide efficient support for crosscutting features, such as QoS algorithms that typically crosscut the structure of the overall system. In an effort to integrate the two software engineering techniques, aspect-oriented and component-based software development, into real-time system development we have developed aspectual component-based real-time system development, ACCORD [32]. ACCORD enables efficient system configuration from components and aspects from the library based on the system requirements.

In this paper we address the essential QoS challenges by providing a method for building reconfigurable QoS management (Re-QoS). The Re-QoS method is founded on ACCORD, and thereby has components and aspects as constituents of the QoS management architecture and policies. Hence, by adapting the ACCORD notion of combining component-based design with aspect-oriented software development, Re-QoS enables successful handling of crosscutting QoS policies, as well as the evolutionary design of real-time systems and QoS management. The Re-QoS method is general and applicable to a large class of real-time systems across different domains and application areas that conform to a set of requirements elaborated in the paper, e.g., available and well-structured code, and conformance to an aspect language.

Re-QoS defines the concept of a QoS aspect package, which represents the implementation of a plethora of different QoS management policies. Hence, a QoS aspect package consists of a number of components and aspects implementing the policies. By adding a QoS aspect package to an existing system without QoS guarantees, we are able to use the same system in unpredictable environments where performance guarantees are essential. Furthermore, by exchanging aspects within the QoS aspect package one can efficiently tailor the QoS management of a real-time system based on the application requirements.

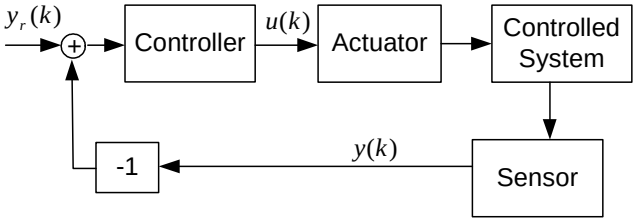

Figure 1: An architecture of the real-time system using feedback control structure

We evaluate Re-QoS by doing a case study of an embedded real-time database, called COMET [20, 31], where we illustrate how a real-time database can be used in different applications, each with distinct QoS needs. We also report the experiences collected from this case study for applying aspect languages in general, and Re-QoS in particular, for real-time system development.

The paper is organized as follows. In section 2 we identify the problems in current QoS management implementations with respect to lack of configurability in QoS management and application dependence of the QoS methods. ACCORD is then briefly discussed in section 3. We propose, in section 4, Re-QoS as a method that addresses the identified problems. We present a case study using the COMET database in section 5, where we demonstrate the successful application of the Re-QoS method. The paper finishes with main conclusions and directions for future work in section 6.

2. QOS MANAGEMENT

We first review main characteristics of feedback-based QoS management policies and then identify main problems of current QoS management (section 2.1). We discuss the goals and methodology adopted in this paper to address the identified problems in section 2.2.

2.1 Feedback-Based QoS Management

A typical structure of a feedback control system is given in figure 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 $u(k)$ and the actuator. The objective of the control is to compute $u(k)$ 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 more reliable performance and system adaptability as the actual system performance is closer to the desired system performance.

We have identified the following problems with current feedback-based QoS management methods.

(P1) QoS methods are *specific to the domain, application, and real-time system*, since a QoS method developed for one real-time system in one domain and one application cannot easily be applied to the same domain with a different application. Here a domain refers to the domain in which a real-time system is used. An application refers to the application area or an environment in which a real-time system resides. For example, a real-time database is a real-time system that is used in the database domain to ensure efficient manipulation of data. It can reside in applications such as vehicle control applications, Internet applications, mobile, web services etc. Currently, there exist a number of QoS management methods specifically developed to suit either the needs of a specific real-time system or a specific application, e.g., real-time databases [5], control applications [9], and web services [1].

(P2) QoS policies employed in existing methods [5, 2, 18, 19, 22, 9, 16] are *metric-centric*, since a QoS controller (see figure 1) is typically developed to control a certain type of metric, and without extensive modifications to the system and the controller, the QoS controller cannot be reused for another metric.

(P3) QoS policies used in [5, 2, 18, 19, 22, 9, 16] are *crosscutting*, since they are integrated with parts of a real-time system crosscutting its overall structure and, thus, cannot be exchanged or modified separately.

(P4) QoS management architectures [5, 2, 18, 19, 22, 9, 16] are *non-evolutionary*, since they are fixed and monolithic, and modifications in the QoS management require complex modifications in the code of the overall system. Moreover, current methods 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 [26, 20]. In this case a cost-effective way of building QoS-aware vehicle control systems is to efficiently incorporate QoS mechanisms into already existing systems.

All the identified problems (P1)-(P4) result in two additional drawbacks of the existing feedback-based QoS methods, namely the methods are (P5) *not configurable*, and (P6) *not reusable*. Thus, existing QoS methods for real-time systems due to shortcomings (P5) and (P6) do not comply with essential QoS principles of enabling configuration and reuse [6].

As the problems (P1)-(P4) induce the problems (P5) and (P6), it is apparent that providing a QoS method developed specifically to addresses the problems (P1)-(P4), would result in solving also the remaining problems (P5) and (P6).

2.2 Goals and Methodology

The goal of the work presented in this paper is to address the problems (P1)-(P6) of the current feedback-based QoS methods identified in section 2.1. We do that by proposing a reconfigurable QoS management method (Re-QoS). In order to provide a method that is application-, domain-, and system-independent, i.e., address (P1), we need to ensure that Re-QoS is applicable to a large class of real-time systems across different applications, and that the QoS architecture is flexible in that QoS policies can be exchanged and modified depending on the application requirements. We do this by: (i) defining a set of requirements that a real-time system needs to fulfill in order to apply the Re-QoS method, and (ii) utilizing aspect-oriented software development, together with component-based development in Re-QoS. Aspects allow us to overcome (P2) as we can exchange the controlling metric by simply weaving appropriate aspects into the controllers. Further, by using the notion of aspect as modules that encapsulate crosscutting features in the system [15], we efficiently handle the crosscutting QoS management feature and thereby attack the problem (P3). Re-QoS enables evolutionary design of existing QoS management architectures, thus, resolving problem (P4). We show that the solution to (P1)-(P4) by Re-QoS leads to a method that is both configurable and reusable, corresponding to (P5) and (P6).

We evaluate Re-QoS using a case study of a real-time database called COMET, where we demonstrate the way a real-time database can be used in different applications. The case study also serves as a guide for instantiating a QoS management architecture built based on Re-QoS for an arbitrary real-time system (given that it conforms to the requirements) across a number of applications.

3. ASPECTUAL COMPONENT-BASED REAL-TIME SYSTEMS DEVELOPMENT

Aspectual component-based real-time systems development, denoted ACCORD, is an approach that enables development of reconfigurable real-time systems [31, 32]. ACCORD prescribes that real-time systems should first be decomposed into a set of components followed by decomposition into a set of aspects.

Within ACCORD aspects in real-time systems are classified in different categories [31, 32]: (i) application aspects, (ii) run-time aspects, and (iii) composition aspects. The classification eases the reasoning about different embedded and real-time related requirements, as well as the composition of the system and its integration into a run-time environment. Application aspects can change the internal behavior of components to suit a particular application as they crosscut the code of components in the system, e.g., memory optimization aspect and real-time policy aspect. Run-time aspects give information needed by the run-time system to ensure that integrating a real-time system would not compromise timeliness or available memory consumption. Composition aspects describe with which components a component can be combined, the version of the component, and possibilities of extending the component with additional aspects.

ACCORD provides a real-time component model, denoted

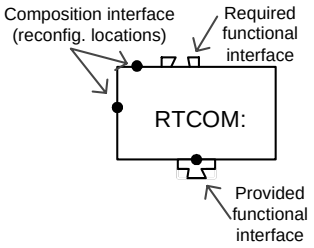

Figure 2: RTCOM in a nutshell

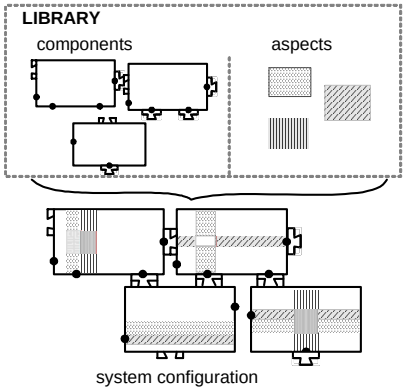

Figure 3: ACCORD-based design

RTCOM, to support reconfigurability [31, 32]. RTCOM components are "grey" as they are encapsulated in interfaces, but changes to their behavior can be performed in a predictable way using aspects. Hence, components provide certain initial functionality that can be modified or changed by weaving of aspects. Each RTCOM component has two types of functional interfaces: provided and required (see figure 2). Provided interfaces reflect a set of operations that a component provides to other components, while required interfaces reflect a set of operations that a component requires (uses) from other components. Composition interfaces define reconfiguration locations of the component code (represented as black circles in figure 2). Reconfiguration locations define the points in the component code where additional modification of components can be done by aspect weaving, i.e., they represent explicitly declared join points in the component code. These points can be used by the component user (or component developer) to reconfigure a component for a specific application or reuse context. Note also that the operations declared in the provided functional interface can be used for aspect weaving and, thus, they represent also implicit reconfiguration locations.

By choosing appropriate components and aspects from the library, different real-time system configurations can be made (see figure 3).

4. RECONFIGURABLE QOS MANAGEMENT

Requirements that a real-time system needs to fulfill in order to apply Re-QoS are elaborated in section 4.1. The Re-QoS method is then presented in section 4.2.

4.1 System Requirements

The Re-QoS method 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 following requirements.

1. Traditional real-time systems:
   - should be written in a language that has a corresponding aspect language;
   - should have the source code of the system available; and
   - should have well-structured code such that the code is structured in fine-grained pieces that perform well-defined functions, i.e., good coding practice is employed, and, thereby, the locations in which reconfiguration could be performed can be determined and extracted.
2. Configurable, component-based, real-time systems: the system should be built using "glass box" or "grey box" component models, where the component has well-defined interfaces, but also internals are accessible for manipulation by the software developer, e.g., Koala [34], RTCOM [32], AutoComp [25], PBO [29], and Rubus-based component models [13].

When a real-time systems is built from scratch, then the system can be optimized during design and development for Re-QoS. This is done by simply following the guidelines outlined in the following section adopted for developing different parts of the Re-QoS framework.

4.2 Re-QoS Method

The Re-QoS method prescribes a hierarchical QoS management architecture that, on the highest level, consists of two classes of entities: the QoS component type and the QoS aspect types (see figure 4). The top level of the QoS architecture can be used for any QoS management. On the lower level, the component type includes a feedback controller component (FCC), a QoS actuator component (QAC), and a sensor component (SC). The aspect types are constructed to embrace the following types of aspects (see figure 4):

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

The QoS component type is defined as a grey box component implementing a well-defined function and it conforms to the RTCOM model. As such, a component has an interface that contains the following information: (i) functionality, in terms of functions, procedures, or methods, that a component requires (uses) from the system, and functionality that a component provides to the system; (ii) the list of reconfiguration locations where the changes of the component policy can be done.

The QAC is of QoS component type and, in its simplest form, acts as a simple admission controller. It publishes a

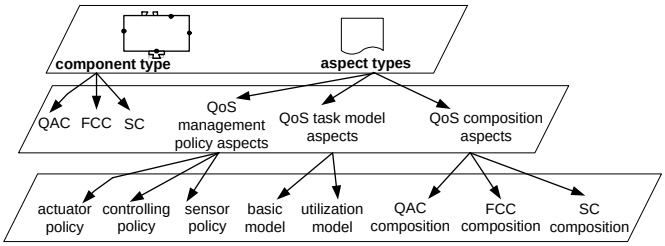

Figure 4: Re-QoS hierarchical model

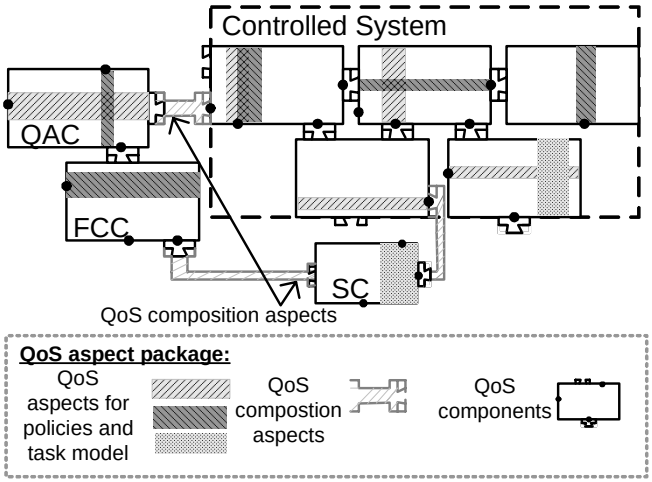

Figure 5: The Re-QoS architecture

list of reconfiguration locations in its interfaces where different actuator policies can be woven. Similarly, the FCC is by default designed with a simple control functionality and appropriate reconfiguration locations, such that the controller can be extended to support more sophisticated control algorithms, e.g., adaptive control [35]. The SC collects necessary data and possibly aggregates it to form the metric representing the controlled variable. In its simplest form the SC measures utilization, which is commonly used as a controlled variable [18]. The SC publishes a set of reconfiguration locations, where it is possible to change the measured metric.

The QoS policy aspects adapt the system to provide different QoS management policies. Based on the target application, the aspects modify the FCC to support an appropriate QoS controller, and also changes the QAC and the SC according to the choice of manipulated variable and controlled variable, respectively. For example, if deadline miss ratio is to be controlled by changing the computation quality of the tasks, then a QoS policy aspect is chosen such that the deadline miss ratio is measured. The QAC is modified by the aspect, exchanging the simple admission policy for a quality adaptation actuator. Hence, QoS policy aspects can further be refined into actuator policy, controller policy and sensor policy.

The QoS task model aspects adapt the task model of a real-time system to the model used by different QoS policies. There can be a number of aspects defined to ensure enrichment or modifications of the task model, so that the resulting task model is suitable for different QoS or applications needs. Here we only give simple examples of task models; see [8] for a detailed overview of available task models. For example, one may model a task to be periodic, aperiodic, or sporadic. Similarly, a task may be soft or firm. Concrete examples of the task model are given in section 5.3.

The QoS composition aspects facilitate the composition of a real-time system with the QoS-related components, FCC, QAC, and SC.

The Re-QoS architecture is shown in figure 5. This architecture allows QoS management policies to easily be exchanged by adding/changing aspects within the QoS management policy type. Hence, it ensures that QoS management policies are modifiable and configurable, depending on the application requirements; thereby addressing the problems (P1)-(P4). QoS composition aspects add the FCC and QAC to the parts/components of the system or a system itself, where needed; these aspects are represented with grey dashed lines between component connections in figure 5. Additionally, QoS composition aspects offer significant flexibility in the system design as the feedback loop can be placed just as easily "outside" the system as between any components in the system by simply adding QoS composition aspects.

Once the QoS management is developed according to Re-QoS, all developed aspects and components implementing a plethora of different QoS management policies and algorithms, are grouped into a so-called QoS aspects package. Hence, a QoS aspect package represents an implementation, i.e., an instantiation, of the Re-QoS method for a specific set of QoS management policies and a specific set of applications.

Now, a real-time system can be developed so that it fulfills certain functionality (without QoS mechanisms), and then QoS-mechanisms can be added to the system using Re-QoS. A specific implementation of a set of components and aspects based on Re-QoS, i.e., a QoS aspect package for a particular system, is developed. Re-QoS then enables efficient upgrades of already existing systems to support QoS performance assurance by simply adding aspects that implement different QoS policies from the developed QoS aspect package. When adopting a real-time system for new QoS needs, the developer takes appropriate aspects and components from the QoS aspect package and adds them to the system, depending on the application QoS requirements. This gives ability to produce a variety of different system configurations with distinct QoS management algorithms, which corresponds to solving problem (P5). Note also that aspects implemented within a QoS aspect package can easily be reused in different applications, hence, problem (P6) is addressed. Moreover, the Re-QoS method via its aspect package concept enables closed systems to be efficiently used in open environments.

It is indeed possible to design a real-time system without the QoS management and then add the QoS dimension to the system using Re-QoS (including an instantiation of the appropriate QoS aspect packages for a set of different appli-

cations). We prove these claims in the following section on the example of the COMET database.

5. A CASE STUDY

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 [20, 31]. Thus, the initial COMET implementation, which is discussed in section 5.1, does not contain QoS mechanisms. To adapt COMET to the real-time systems with performance assurance guarantees we applied the Re-QoS method to the existing COMET configuration, and developed a COMET QoS aspect package. In order to understand the choices made when developing different aspects for COMET QoS, we first present the QoS policies used for implementation of the QoS aspect package (section 5.2) and then we discuss the data and transaction models used in different COMET configurations (section 5.3). In section 5.4 we discuss in depth the details of different components and aspects within the COMET QoS. Possible COMET QoS configurations are given in section 5.5. In section 5.6 we prove experimentally that COMET with the QoS extensions indeed provides expected QoS guarantees. Finally, we report our experiences from the case study in section 5.7.

5.1 COMET Overview

Following the ACCORD design method described in section 3, the architecture of COMET consists of a number of components and a number of aspects. COMET components are (see figure 6(a)): user interface component (UIC), transaction management component (TMC), index management component (IMC), and memory management component (MMC). The UIC provides a database interface to the application, which enables a user (application) to query and manipulate data elements. Application requests are parsed by the UIC, and are then converted into an execution plan. The TMC is responsible for executing incoming execution plans, thereby performing the actual manipulation of data. The IMC 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, which is especially suitable for small embedded vehicular systems [20].

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 (LMC) and the scheduling manager component (SMC), are needed (see figure 6(b)). The SMC is responsible for registering new transactions to the system and scheduling them according to the chosen scheduling policy, e.g., earliest-deadline first (EDF) [17]. The LMC 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 [32]. For purposes of this paper it is important to note that we are implementing the COMET QoS aspect package for the concurrent COMET implementation.

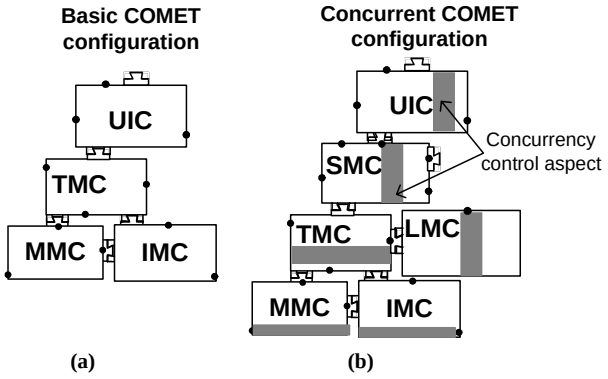

Figure 6: Basic and concurrent COMET configurations

Each component has interfaces as defined by RTCOM where it publishes both operations and reconfiguration locations. For example, the TMC executes transactions by executing an operation `TMC_getResult()` declared in its provided interface. The SMC declares an operation `SMC_CreateNew()` in its provided interface, which registers transactions. Moreover, transactions in the SMC are scheduled using internal function `scheduleRecord()`, and this function is declared as a reconfiguration location of the SMC.

5.2 QoS Policies

Given that we want to use the COMET database with applications that require performance guarantees, we need to adapt some existing QoS policies and, using Re-QoS, integrate them into the database. Hence, in this section we give a brief overview over two instances of feedback-based QoS management methods we use in our case study; we found that these are especially suitable for ensuring performance guarantees in real-time systems. First we describe one instance of the feedback-based QoS management method, referred to as FC-M [18], where deadline miss ratio is controlled by modifying the admitted utilization. This is followed by a description of the QoS sensitive approach for miss ratio and freshness guarantees (QMF) [14], used for managing QoS in real-time databases.

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 tasks that have missed their deadline and $admittedTransactions(k)$ be the number of terminated admitted tasks in the time interval $[(k-1)T, kT]$. The deadline miss ratio,

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

denotes the ratio of tasks 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 figure 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

[12], hence, $\delta u(k) = K_P e_m(k)$, 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 actuation approach used in QMF [14], 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 continuously updating them is unnecessary, i.e., they can be updated on-demand. On the other hand, data items that are frequently requested 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 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.

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 model for data is simple and does not support data freshness requirements. The basic transaction model is such that each transaction τ_i is characterized only with the period p_i, and the relative deadline d_i. However, QoS algorithms like FC-M and QMF require distinct and more complex data and the transaction models.

In the *differentiated data model*, data objects are classified into two classes, temporal and non-temporal [23]. 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 policies 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 data objects. User transactions arrive aperiodically and may read temporal and read/write non-temporal 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 $r_{E,i}$ (user transactions),
- the actual mean interarrival time $r_{A,i}$ (user transactions),

Attribute	Periodic Tasks	Aperiodic Tasks
d_i	$d_i = p_i$	$d_i = r_{A,i}$
$u_{E,i}$	$l_{E,i} = x_{E,i}/p_i$	$l_{E,i} = x_{E,i}/r_{E,i}$
$u_{A,i}$	$l_{A,i} = x_{A,i}/p_i$	$l_{A,i} = x_{A,i}/r_{A,i}$

Table 1: The utilization transaction model.

- the estimated execution time $x_{E,i}$,
- the actual execution time $x_{A,i}$,
- the relative deadline d_i,
- the estimated utilization[1], $u_{E,i}$, and
- the actual utilization, $u_{A,i}$.

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

5.4 COMET QoS Aspect Package

Applying the Re-QoS method on COMET resulted in the development of the COMET QoS aspect package that enables the database to be used in applications that have uncertain workloads and where requirements for data freshness are essential. The current QoS aspect package provides components and aspects that implement the FC-M and QMF QoS policies. The aspects within the package are implemented using AspectC++ [28]. The COMET QoS aspect package consists of the QAC and FCC components and the following aspects:

- QoS management policy aspects: QAC utilization policy, missed deadline monitor, missed deadline controller, scheduling strategy, data access monitor, and QoS through update scheduling aspect;
- QoS transaction and data model aspects: utilization transaction model aspect and data differentiation aspect; and
- QoS composition aspects: QAC composition and FCC composition aspect.

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

The *FCC* is a components that computes input to the admission policy of the QAC at regular intervals. By default, an input of zero is generated, but by using QoS controlling policy aspects different feedback policies can be used. The

[1]Utilization is also referred to as load.

```
1: aspect QAC_composition{
2:   / / Insert QAC between UIC and SMC.
3:   advice call("bool SMC_CreateNew(...)") : around() {
4:     if (QAC_Admit(*(scheduleRecord *)tjp->arg(0)))
5:       tjp->proceed();
6:     else
7:       *(bool *)tjp->result() = false;
8:   }
9: };
```

Figure 7: QAC composition aspect

```
1:  aspect QAC_utilization_policy{
2:    / / Add a utilization reference to the system
3:    advice "UIC_SystemParameters" : float utilizationRef;
4:    / / Changes the policy of the QAC to the utilization
5:   advice execution("% QAC_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:  }
```

Figure 8: QAC utilization policy aspect

FCC provides only one operation, `FCC_Init()`, that initializes the FCC component. FCC calls `QAC_Adjust()` 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 section 5.3. This is done using inter-type declaration that adds new parameters to the basic model, e.g., estimated utilization $u_{E,i}$ and estimated execution time $x_{E,i}$.

The *QAC composition aspect* enables QAC to intercept requests to create new transactions that are posed by the UIC to the SMC. This is done via an advice of type around which is executed when the SMC operation `SMC_CreateNew()` is called (lines 3-8 in figure 7). Since this operation of the SMC is in charge of registering a new transaction to the system, the advice ensures that, before the transaction is actually registered, an admission test is made by the QAC (line 4). If the transaction can be admitted the transaction registration is resumed; the `proceed()` in line 5 enables the normal continuation of the join point `SMC_CreateNew()`. 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 *QAC utilization policy aspect* shown in figure 8 replaces, via the around advice (lines 5-13), the default admission policy of QAC with an admission policy based on utilization (lines 9-12). The current transaction examined for admission in the system is denoted `ct` in figure 8.

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

```
1:  aspect missed_deadline_monitor {
2:    advice call("% SMC_CreateNew(...)") : after(){
3:        if (*(bool *)tjp->result()) { admittedTransactions++; }
4:    }
5:    advice call("% SMC_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("% SMC_Aborted(...)") : before(){…
14:      admittedTransactions--;}
15:   advice call("% SMC_RejectLeastValuableTransaction(...)") :  after(){
16:      if (*(bool *)tjp->result()) { admittedTransactions--;}
17:   }
18:   advice call("% getTimeToDeadline(...)") && within("%
19:     getNextToExecute(...)") : after() {… missedTransactions++;}
20: }
```

Figure 9: Missed deadline monitor aspect

The *missed deadline monitor aspect* modifies the SMC 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 that intercept SMC operations that handle completion and abortion of transactions (see figure 9). For example, the advice of type after that intercepts the call to `SMC_CreateNew()` increments the number of admitted transactions once transactions have been admitted to the system (lines 2-4). Similarly, the advice in lines 5-12 checks if the number of transactions with missed deadlines should be incremented before the transaction has completed, i.e., before invoking the TMC operation `SMC_Completed()`.

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

The *data differentiation aspect* enriches the data model of the basic COMET configuration to differentiate between base data and derived data. Differentiation is done by assigning avi_i and $timestamp_i$ attributes to data items manipulated by the transaction. Inserted data items containing fields for avi_i and $timestamp_i$ are assumed to be base data. Whenever these data values are inserted or modified, $timestamp_i$ is set to the current time.

The *scheduling strategy aspect* modifies the scheduling strategy and the data model of the COMET to support two distinct update strategies for base data: immediate and on-demand [23]. To accommodate these strategies the aspect adds an *update wait queue* in the SMC (advice of type after in lines 2-6 in figure 11). The name of the update strategy is stored in a field in the relation (see line 15 for an

```
1: aspect missed_deadline_control{
2:   / / Initialize the new variales need for control
3:   advice call("% UIC_init(...)") : after() {
4:     UIC_SystemParameters *sp =
5:         (UIC_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(RSMC_GetDeadlineMissRatio());
16:    *((float *)tjp->result()) = missRatioOutputHm;
17:   }
18: }
```

Figure 10: Missed deadline control aspect

```
1: aspect scheduling_policy{
2:    advice call("% SMC_constructor(...)") : after(){
3:      / / Initialize the update-wait queue
4:     UpdateWaitQueue_root = SMC_createNode(...);
5:    ...
6:    }
7:    advice call("% insert(...)") : before(){
8:      / / Set update type to immediate upon
9:       / /if the data is base data.
10:     if (isUpdateTypeData(buffer)){
11:     while (updateTypeNr > counter){
12:           counter++;
13:           treePtr = treePtr->right;
14:     }
15:    strcpy(treePtr->left->Data.operandID, "IMMEDIATE");
16:   }
17:  }
18:   advice call("% ReadData(...)") : after(){
19:     / / If it is a base data relation...
20:     / / If it is not an update or insert transaction...
21:     / / If it is invalid...
22:   }
23:  …..
24: }
```

Figure 11: Scheduling strategy aspect

example). Hence, an inserted data that contains a field for update strategy as well as fields for avi_i and $timestamp_i$ is handled by this aspect. Note that, when a transaction reads a base data item, the freshness of the item is examined in the advice that is executed after TMC `readData()` is called, i.e., after the data is read from the memory (lines 18-22 in figure 11). If the base data item is stale and the updating strategy is set to on-demand, the transaction is rolled back and moved to the update wait queue. If the updating strategy is set to immediate, the transaction is rolled back and restarted. Updates of base data items set to immediate are always allowed, while updates of base data items set to on-demand are rejected unless these data items have been requested by a transaction in the update wait queue. If so, the requesting transaction is moved to the ready queue and the update executes normally.

The *data access monitor aspect* modifies the TMC to keep track of how often base data items are accessed. Remember that in QMF data base items are updated on-demand or immediate based on how often they are accessed.

Table 2: Relationship between different parts of the QoS package and different COMET QoS configurations

QoS aspect package		COMET configurations: Admission control	COMET FC-M	COMET QMF
policy aspects	QAC utilization policy	X	X	X
	Missed deadline monitor		X	X
	Missed deadline controller		X	X
	Scheduling strategy			X
	Data access monitor			X
	QoS through update scheduling			X
transaction model aspects	Utilization transaction model	X	X	X
	Data differentiation			X
composition aspects	QAC composition aspect	X	X	X
	FCC composition aspect		X	X
components	QAC	X	X	X
	FCC		X	X

The *QoS through update scheduling aspect* uses the data differentiation aspect, scheduling strategy aspect, and the data access monitor aspect to modify the QAC such that the actuator policy in QMF is used. Hence, when applying the QoS through update scheduling aspect, changes to quality of data in combination with admission policy is used to enforce utilization changes based on the control signal from FCC.

5.5 QoS COMET Configurations

In this section we show that Re-QoS with its QoS aspect package concept ensures configurability and reusability of QoS management, and also in practice solves (P5) and (P6). Depending on the demands of the application with which COMET is used, three distinct COMET QoS configurations can be made by selecting appropriate aspects and components within the package. Table 2 illustrates which elements of the QoS aspect package are used in different configurations.

The admission control configuration requires the utilization transaction model aspect to extend the transaction model as well as the QAC, and its composition aspect (see table 2). This configuration is simple as it only provides facilities for admission control.

The miss ratio feedback configuration (COMET FC-M) provides the QoS guarantees based on FC-M policy. The configuration includes both the QAC and FCC components and their corresponding composition aspects, as well as the utilization transaction model aspect, and the missed deadline monitor and controller aspect (see table 2). These aspects modify the policy of the SMC and FCC to ensure that QoS based on deadline misses is enforced.

The update scheduling configuration (COMET QMF) provides the QoS guarantees based on QMF policy. Here the

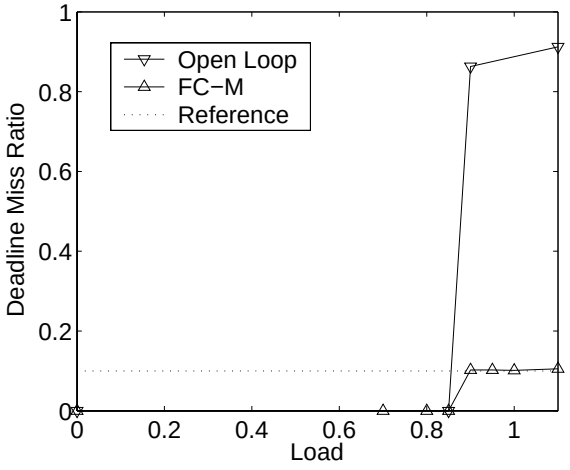

Figure 12: Deadline miss ratio as a function of load

data differentiation aspect and scheduling strategy aspect are used to enrich the transaction model even further than it was done in the previous configuration. 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 QAC to suit the QMF algorithm.

5.6 Experimental Evaluation

In this section we present one experiment made on the COMET platform with and without the QoS aspect package. The goal of the experiment is to show that the QoS management mechanisms in COMET perform as expected and thereby show that, when adding the QoS aspect package, we indeed achieve reconfigurability 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 (see [7]). Due to space limitations we present only one of the experiments.

For doing the experiment we have chosen the following experiment setup. The database consists of eight relations, each containing ten tuples. Constant streams of transaction requests are used in the experiments. To vary the load on the system, i.e., utilization, the interarrival time between the transactions is altered. Update transactions arrive periodically, whereas user transactions arrive aperiodically. The deadline miss ratio reference, i.e., the desired deadline miss ratio, is set to 10%.

We present an experiment 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 open-loop system, i.e., concurrent COMET configuration without the QoS aspect package, as a baseline. For all the experiment data, we have taken the average of 10 runs. Figure 12 shows the deadline miss ratio of concurrent COMET and COMET with the FC-M configuration. The dotted line indicates the reference deadline miss ratio, i.e., the desired QoS.

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. On the contrary, COMET with the 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 [3, 4, 5].

The consequence of this is that we achieve reliable QoS management when using the FC-M configuration, as the actual QoS is equal to the desired QoS. From this we conclude that the COMET FM-C configuration is able to provide QoS guarantees under varying load.

5.7 Experience Report

This section contains observations we made with respect to our usage of aspect-oriented software development in general and Re-QoS in particular for ensuring configurability and reusability of QoS mechanisms.

Lesson 1: *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 of the aspects and the system at the expense of limiting configurability and reusability of the aspects in the system. This conforms to the conclusions made in [33] where a tradeoff between requirements for reconfigurability and maintenance when using aspects in software systems is identified. In the case when there is a focus on maintainability, the missed deadline monitor aspect and the missed deadline control aspect, which are part of the QoS policy aspects category and implement FC-M policy, could be combined into one aspect which both monitors and controls the deadline misses. The same is true for the scheduling strategy aspect and the QoS through update scheduling aspect that both implement parts of the QMF algorithm. We have chosen, however, to have these in different aspects to enable them to be independently exchanged from the configuration and from each other. For example, the missed deadline monitor aspect is decoupled from missed deadline controller aspect to ensure that reuse of aspects is increased, since the missed deadline monitor aspect can generally be used in combination with another controller policy.

Lesson 2: *Aspects can be reused in all phases of the system development.* We found that aspects that are developed within the Re-QoS and implemented in an aspect package can efficiently be reused in different phases of the of the system development. For example, the missed deadline monitor aspect is used in the design and implementation phase of the system as a part of a QoS method, i.e., to implement a specific QoS policy. Additionally, this aspect is efficiently reused in the evaluation phase of the system development, where it was used for performance evaluation and gathering statistics. All performance evaluations presented in section 5.6 are done using aspects as means of monitoring system performance.

Lesson 3: *Reconfiguration locations lead to an efficient and analyzable product-line architecture.* We observed that,

for development of different system configurations using the same set of components in combination with different aspects, we need to explicitly define places in the architecture where extensions can be made, i.e., aspects woven. Therefore, we enforced in our component model that the places where possible extensions can take place (component can be reconfigured) are explicitly declared in the component interfaces. Although we restrict the join point model of the aspect language, we obtain clear extension points in the components and the system architecture. Moreover, the reconfiguration locations in the component code are desirable in the real-time domain as they provide pre-defined places where code modifications can be done, and therefore can be analyzed more efficiently during the design and pre-run time phases of the system development [30, 32]. Hence, using Re-QoS and notion of aspect package we can efficiently develop product line architecture of a real-time system that has the ability to satisfy specified QoS needs.

Lesson 4: *Aspect languages are means of dealing with legacy software.* Since we developed the COMET database system to be primarily suited for hard real-time systems in the vehicular industry [21], the programming language used for the development of the basic database functionality (described in section 5.1) needed to be suited for the software that already existed in a vehicular control system. Moreover, analysis techniques that have been used in the existing system should have been applicable to our basic database components. This leads to the development of the COMET basic configuration using C programming language. Aspects provided efficient means for introducing extensions to the system; we used the AspectC++ weaver since a weaver for C language [10] is not publicly available. In overall, we have concluded that, if extending existing real-time systems, which are typically developed in a non-object-oriented language such as C, aspects are of greater value than rebuilding the system using an object-oriented language and then making extensions to it using an object-oriented language such as C++.

6. SUMMARY

In this paper we have presented the feedback-based QoS method Re-QoS that empowers reconfigurability in QoS management of real-time systems. The Re-QoS method provides 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 QoS requirements. Furthermore, QoS policies within Re-QoS are encapsulated into aspects and can be exchanged and modified independently of the controlled real-time system. This improves reusability of QoS management and ensures applicability of Re-QoS across different applications. Re-QoS also enables existing real-time systems, without QoS guarantees, to be used in applications that require specific performance guarantees. This is done through the concept of a QoS aspect package, which is a Re-QoS-based implementation of different QoS policies (suitable for a number of applications) for a specific real-time system. By exchanging aspects within the QoS aspect package one can efficiently tailor the 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 adopted the COMET database platform. Initially, COMET was developed to be used in closed real-time systems, but by adding the QoS aspect package COMET can be used in open environments with unpredictable workloads.

Our on-going work focuses on adaptive control techniques [35], encapsulated into aspects, for reconfigurable QoS management. This approach enables a system to adapt to exchanges to or upgrades of system components on-line while preserving the specified level of QoS.

7. REFERENCES

[1] T. F. Abdelzaher, J. A. Stankovic, C. Lu, R. Zhang, and Y. Lu. Feedback performance control in software services. *IEEE Control Systems Magazine*, 23(3):74–90, June 2003.

[2] M. Amirijoo, J. Hansson, S. Gunnarsson, and S. H. Son. 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.

[3] M. Amirijoo, J. Hansson, and S. H. Son. Error-driven QoS management in imprecise real-time databases. In *Proceedings of the IEEE Euromicro Conference on Real-Time Systems (ECRTS)*, 2003.

[4] M. Amirijoo, J. Hansson, and S. H. Son. Specification and management of QoS in imprecise real-time databases. In *Proceedings of the IEEE International Database Engineering and Applications Symposium (IDEAS)*, 2003.

[5] M. Amirijoo, J. Hansson, S. H. Son, and S. Gunnarsson. Robust quality management for differentiated imprecise data services. In *Proceedings of the IEEE International Real-Time Systems Symposium (RTSS)*, 2004.

[6] C. Aurrecoechea, A. T. Campbell, and L. Hauw. A survey of QoS architectures. *Verlag Multimedia Systems Journal*, 6(3):138–151, May 1998.

[7] M. Björk. QoS management in configurable real-time databases. Master's thesis LITH-IDA-EX-04/071-SE, Department of Computer Science, Linköping University, Sweden, 2004.

[8] G. C. Buttazzo. *Hard Real-Time Computing Systems.* Kluwer Academic Publishers, 1997.

[9] A. Cervin, J. Eker, B. Bernhardsson, and K. Årzén. Feedback-feedforward scheduling of control tasks. *Journal of Real-time Systems*, 23(1/2), July/September 2002. Special Issue on Control-Theoretical Approaches to Real-Time Computing.

[10] Y. Coady and G. Kiczales. Back to the future: A retroactive study of aspect evolution in operating system code. In *Proceedings of the Second International Conference on Aspect-Oriented Software Development (AOSD 2003)*, pages 50–59, Boston, USA, 2003. ACM Press.

[11] D. Ecklund, V. Goebel, T. Plagemann, E. F. E. Jr., C. Griwodz, J. Aagedal, K. Lund, and A.-J. Berre. QoS management middleware: A separable, reusable solution. In *Proceedings of the 8th International Workshop on Interactive Distributed Multimedia Systems (IDMS'01)*, 2001.

[12] G. F. Franklin, J. D. Powell, and M. Workman. *Digital Control of Dynamic Systems.* Addison-Wesley, third edition, 1998.

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

[14] K.-D. Kang, S. H. Son, J. A. Stankovic, and T. F. Abdelzaher. A QoS-sensitive approach for timeliness and freshness guarantees in real-time databases. In *Proceedings of the Euromicro Conference on Real-time Systems (ECRTS)*, 2002.

[15] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In *Proceedings of the ECOOP*, volume 1241 of *Lecture Notes in Computer Science*, pages 220–242. Springer-Verlag, 1997.

[16] B. Li and K. Nahrstedt. A control theoretical model for quality of service adaptations. In *Proceedings of the International Workshop on Quality of Service*, 1998.

[17] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in hard real-time environments. *Journal of the Association for Computing Machinery*, 20(1):46–61, January 1973.

[18] C. Lu, J. A. Stankovic, G. Tao, and S. H. Son. Feedback control real-time scheduling: Framework, modeling and algorithms. *Journal of Real-time Systems*, 23(1/2), July/September 2002.

[19] Y. Lu, A. Saxena, and T. F. Abdelzaher. Differentiated caching services; a control-theoretical approach. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*, 2001.

[20] D. Nyström, A. Tešanović, M. Nolin, C. Norström, and J. Hansson. Comet: A component-based real-time database for automotive systems. In *Proceedings of the Workshop on Software Engineering for Automotive Systems at 26th International Conference on Software engineering (ICSE'04)*, Edinburgh, Scotland, May 2004. IEEE Computer Society Press.

[21] D. Nyström, A. Tešanović, C. Norström, J. Hansson, and N.-E. Bankestad. Data management issues in vehicle control systems: a case study. In *Proceedings of the 14th Euromicro International Conference on Real-Time Systems*, Vienna, Austria, June 2002.

[22] S. Parekh, N. Gandhi, J. Hellerstein, D. Tilbury, T. Jayram, and J. Bigus. Using control theory to achieve service level objectives in performance management. *Journal of Real-time Systems*, 23(1/2), July/September 2002.

[23] K. Ramamritham. Real-time databases. *International Journal of Distributed and Parallel Databases*, (1), 1993.

[24] K. J. Åström and B. Wittenmark. *Adaptive Control*. Addison-Wesley, second edition, 1995.

[25] K. Sandström, J. Fredriksson, and M. A. Kerholm. Introducing a component technology for safety critical embedded realtime systems. In *In Proceedings of the International Symposium on Component-based Software Engineering (CBSE7)*, Scotland, May 2004. Springer-Verlag.

[26] M. Sanfridson. 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, March 2000.

[27] H. Schmidt. Trustworthy components-compositionality and prediction. *The Journal of Systems and Software*, pages 215–225, 2003.

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

[29] D. B. Stewart, R. Volpe, and P. K. Khosla. Design of dynamically reconfigurable real-time software using port-based objects. *IEEE Transactions on Software Engineering*, 23(12), December 1997.

[30] A. Tešanović, S. Nadjm-Tehrani, and J. Hansson. Modular Verification of Reconfigurable Components chapter in *Embedded System Development with Components*. Springer-Verlag, 2005.

[31] A. Tešanović, D. Nyström, J. Hansson, and C. Norström. Towards aspectual component-based real-time systems development. In *Proceedings of the 9th International Conference on Real-Time and Embedded Computing Systems and Applications (RTCSA'03)*, volume 2968 of *Lecture Notes in Computer Science*. Springer-Verlag, 2003.

[32] A. Tešanović, D. Nyström, J. Hansson, and C. Norström. Aspects and components in real-time system development: Towards reconfigurable and reusable software. *Journal of Embedded Computing*, 1(1), October 2004.

[33] A. Tešanović, K. Sheng, and J. Hansson. Application-tailored database systems: a case of aspects in an embedded database. In *Proceedings of the 8th International Database Engineering and Applications Symposium (IDEAS'04)*, Coimbra, Portugal, July 2004. IEEE Computer Society.

[34] R. van Ommering. Building product populations with software components. In *Proceedings of the 24th international conference on Software engineering*, pages 255–265, Orlando, Florida, USA, May 2002. ACM Press.

[35] K. J. Åström and B. Wittenmark. *Adaptive Control*. Addion-Wesley, second edition, 1995.