Real-Time Interfaces for Interface-Based Design of Real-Time Systems with Fixed Priority Scheduling

Ernesto Wandeler Lothar Thiele
Computer Engineering and Networks Laboratory
Swiss Federal Institute of Technology (ETH)
8092 Zurich, Switzerland
{wandeler,thiele}@tik.ee.ethz.ch

ABSTRACT

The central idea behind interface-based design is to describe components by a component interface. In contrast to a component description that describes what a component does, a component interface describes how a component can be used. A well designed component interface provides enough information to decide whether two or more components can work together properly in a system. In this work, we expand the idea of interface-based design to the area of real-time system design. Here, the term of 'working together properly' refers to questions like: Does the composed system satisfy all requested real-time properties such as delay and throughput constraints? For this, we introduce Real-Time Interfaces, that connect the principles of Real-Time Calculus with Interface-based Design. In contrast to traditional real-time system design, in interface-based real-time system design the compliance to real-time constraints is checked at composition time. This leads to faster design processes and partly removes the need for the classical binary search approach to find an economically dimensioned system. Further, interface-based real-time system design also benefits from the properties of incremental design and independent implementability.

Categories and Subject Descriptors

C.3 [**Computer Systems Organization**]: Special-Purpose and Application-Based Systems—*Real-time and Embedded Systems*; C.4 [**Computer Systems Organization**]: Performance of Systems—*Modeling Techniques*

General Terms

Performance, Design, Theory.

Keywords

Performance Analysis, Real-Time Calculus, Real-Time Interfaces, Hierarchical Scheduling

EMSOFT'05, September 19–22, 2005, Jersey City, New Jersey, USA.

1. INTRODUCTION

One of the major challenges in the design process of complex real-time embedded systems is to analyze essential characteristics of a system architecture in an early design stage, to support the choice of important design decisions before much time is invested in detailed implementations. For real-time embedded systems, such essential system characteristics are for example whether maximum delay and throughput constraints are met, what the on-chip memory requirements will be, or how different architectural elements must be dimensioned.

The system-level analysis of embedded systems is currently mainly based on simulation, using for example SystemC, or trace based simulation as in [9]. While these techniques allow modeling and simulation of complex systems in any level of detail, simulation based approaches do not allow to obtain worst-case results, and moreover they often suffer from long run-times.

In contrast, formal analytical methods typically allow to obtain hard bounded results for embedded real-time system designs. An analytical framework for system level performance analysis was proposed in [10]. This framework uses a number of well know abstractions to capture the timing behavior of event streams and provides additional interfaces between them. Traditional results from the area of real-time schedulability analysis are then used to analyze a component-based real-time system design.

Another method for component-based analysis and design of real-time systems was proposed in [17]. This framework uses the notion and concept of traditional software component standards such as CORBA and proposes extensions for the support of real-time services. The method proposed in [17] is however tailored towards software engineering and does not incorporate any notion of hardware resources.

Another approach of composing real-time components is by making use of hierarchical scheduling, see [11]. The approach in [11] is however restricted to periodic tasks and a periodic resource model.

In the domain of communication networks, powerful abstractions have been developed to model flow of data through a network. In particular Network Calculus [8] provides means to deterministically reason about timing properties of data flows in queuing networks, Network Calculus can thereby be considered as a queuing theory that provides hard bounded worst-case/best-case results. Real-Time Calculus [13] extends the basic concepts of Network Calculus to the domain of real-time embedded systems and in [2] a unifying approach to Modular Performance Analysis with Real-Time

Calculus has been proposed. It is based on a general event model, allows for hierarchical scheduling and arbitration, and can take computation and communication resources into account. Modular Performance Analysis with Real-Time Calculus has been successfully used in several case studies for hard real-time system analysis, see e.g. [3] or [16].

The performance analysis methods mentioned above have in common that they are applied to analyze a component-based real-time system design a posteriori. Thus, with these methods a real-time system gets first designed and dimensioned before performance analysis is applied. The analysis results will then reveal whether a certain system architecture will meet all real-time requirements, or not.

In difference to this two-step approach is the idea of interface-based design [4, 7], where components are described by a component interface. In contrast to a component description that describes what a component does, a component interface describes how a component can be used. Through *input assumptions*, a component interface describes the expectations a component has about the other components in a system and through *output guarantees*, a component interface tells other components in a system what they can expect from this component. The major goal of a component interface is then to provide enough information to decide whether two or more components can work together properly.

When we expand this idea of interface-based design to the area of real-time system design, we need a component system with interfaces that provide enough information to decide whether two or more real-time components work together properly. Where in the case of such real-time interfaces, the term 'properly' refers to questions like: Does the composed system satisfy all requested real-time properties such as delay and throughput constraints?

Consequently, in contrast to traditional design, in interface-based real-time system design the compliance to real-time constraints is checked at composition time. This leads to faster design processes and partly removes the need for the classical binary search approach to find an economically dimensioned system. Further, interface-based real-time system design also benefits from the properties of incremental design and independent implementability that are elementary features of interface-based design.

Contributions of this work:

- We present and define the notion of Real-Time Interfaces. Real-Time Interfaces as proposed in this paper are a special instance of assume/guarantee interfaces and they connect the principles of Real-Time Calculus and Interface-based Design.
- We introduce a component model with Real-Time Interfaces, that defines the building blocks for interface-based design of real-time systems. For this, we define components that model hardware resources, components that model event-streams as well as components that model software-processes for preemptive fixed-priority scheduling.
- We show the applicability of Real-Time Interfaces in the area of real-time system design and we show how interface-based design of real-time systems benefits from the properties of incremental design and independent implementability that are elementary features of interface-based design.
- Finally, we show that the properties of Real-Time Interfaces also lead to some interesting on-line applications. Namely in the areas of on-line service and QoS-adaptation as well as in the area of on-line admission tests in real-time systems, where Real-Time Interfaces seem to be a natural and expressive way to capture at run-time the information required by such on-line applications.

2. INTERFACE-BASED DESIGN

The definition of Real-Time Interfaces follows the principles of interface-based design as described by de Alfaro and Henzinger in [4] and more recently [7]. Whereas most previous results relate to stateful interface languages such as interface automata [6], or extensions towards the use of time [5] or resources [1], the Real-Time Interfaces proposed in this paper are based on a stateless assume/guarantee language, see [4]. We shortly describe the underlying principles of interface-based design as one of the two major prerequisites of the new approach to the design of composable real-time systems, for details see [7].

The major goal of the interface of a hardware-software component is to provide enough information to decide whether two or more components work together properly. In case of Real-Time Interfaces, the term 'properly' refers to questions like: Does the composed system satisfy all requested real-time properties such as delay and throughput constraints? Does the composed system satisfy imposed (buffer) memory constraints?

In case of assume/guarantee interfaces, a component interface has a set of disjoint input and output variables X^I and X^O, respectively. As expected, two interfaces F and G can be composed by connecting input variables of one to output variables of the other. The composed interface is denoted as $F\|G$ and has as its input variables all not connected input variables of F and G and as output variables all not connected outputs of F and G.

In order to decide, whether two components can work together properly, the corresponding interfaces are checked for compatibility. To this end, an assume/guarantee interface F contains predicates ϕ_F^I and ϕ_F^O on the values of its input and output variables, respectively. The input assumption ϕ_F^I and output guarantee ϕ_F^O represent a precondition and postcondition of the interface, respectively. In other words, the interface F has the property $\phi_F^I \Rightarrow \phi_F^O$, i.e. if its input variables satisfy ϕ_F^I then the output variables satisfy ϕ_F^O. An isolated component has open inputs and the associated interface has free input variables.

After composing all the components of a system, there are no open inputs any more and therefore, all input variables of the corresponding interfaces are connected to output variables of other components. If the final system has some inputs to the environment, it must be closed by modeling the environment by an appropriate interface. Interface-based design supports incremental design (among other properties like independent implementability). To this end, we would like to compose two interfaces and check for compatibility without closing the system. As a result, we come to an existential interpretation of interface compatibility, see [4]: Two interfaces F and G are compatible, if there exist admissible values of free input variables of $F\|G$ (i.e. satisfying the input assumptions) such that all other input assumptions are also satisfied: $(\forall X_F^O \cup X_G^O)(\phi_F^O \wedge \phi_G^O \Rightarrow \phi_F^I \wedge \phi_G^I)$.

In other words, the new input interface $\phi^I_{F\|G}$ is a predicate (or assumption) on the free input variables of the composed interfaces $F\|G$ such that all internal predicates are satisfied. This way, interfaces can be added one-by-one and the assumptions on the free input variables are getting increasingly tight. Note that the formal definitions of interfaces, compatibility, incremental design, and refinement are described in [4] and [7].

The Real-Time Interfaces as proposed in this paper can be considered as a special instance of assume/guarantee interfaces tailored towards guarantees on the delays of events. On the other hand, there are essential differences to previous work in the interpretation of input/ouput variables [4, 6] of interfaces and in the handling of resources and constraints [1, 5]:

- The variables of interfaces are not directly related to the sequence or timing of events. They contain already abstractions of timed event streams in the form of Variability Characterization Curves, see e.g. [13, 12, 2]. This way, we connect the successful theory of Real-Time Calculus with interface-based design.
- Requirements such as delay and buffer space are part of the input variables and therefore, are part of a component interface. As a result, they can be modified by a component and are subject to input assumptions and output guarantees.
- Resources are considered to be part of a component interface. Therefore, their availability can be modified by a component and input/output predicates can be applied.

This way, composability is achieved in terms of (a) event streams, (b) resources usage and (c) constraints as they can be propagated through a set of hardware/software components.

3. REAL-TIME INTERFACES

As described in the previous section, Real-Time Interfaces as proposed in this paper combine principles of interface-based design with principles of real-time system design and Real-Time Calculus, see e.g. [13, 12, 2]. This section describes the meaning of input and output variables of a Real-Time Interface and the associated assume/guarantee predicates.

3.1 Component, Abstract Component and Interface

In this work, we interpret a single hardware/software component as a task that is executed on a hardware resource. Each component processes an incoming event stream. The component is thereby triggered by the events of the event stream, and a fully preemptable, independent task is instantiated at every event arrival, to process the incoming event. Active tasks are processed in a greedy fashion while being restricted by the availability of resources. Resources that are not consumed by the component are available for other components.

The first step in deriving Real-Time Interfaces involves the concept of an abstract component. In comparison to the above described concrete hardware/software component, an abstract component processes an abstract event stream. The properties of the concrete event stream that enters the hardware/software component are characterized by a Variability Characterization Curve (VCC) which is called arrival curve $\alpha(\Delta)$ following [8]. It describes the properties of event streams that are essential for real-time analysis, see [13, 12]. In addition, the hardware resource which enables the execution of the task is modeled by means of a service curve $\beta(\Delta)$, see also [8]. This Variability Characterization Curve describes the essential properties of the resources available for the task execution. In our simple scenario, the abstract component has only one output $\beta'(\Delta)$ that describes the remaining resource after executing the task for each input event.

Finally, the Real-Time Interface exports the properties of a component that are essential in order to prove compatibility, to enable incremental design and to allow independent implementability. There are two classes of input and output variables of a Real-Time Interface: The value of a service variable is a service curve $\beta(\Delta)$ whereas the value of an arrival variable consists of an arrival curve $\alpha(\Delta)$ and an associated maximal event delay d. The delay d denotes the maximal delay between the arrival of an event and the required finishing time of the associated task execution. To each input variable x and output variable y there are associated assume and guarantee predicates ϕ^I_x and ϕ^O_y, respectively.

In this paper, we consider a very simple scenario only. Components reflect software tasks, the scheduling policy is restricted to preemptive fixed priority, and the tasks associated to the different components are independent of each other. But opposite to the usual assumption of periodic event streams (or at most periodic with jitter), we allow for an arbitrary burstiness of the incoming event streams. It should be noted that the very same principles described in this paper can not only be applied to other constraints such as buffer and memory but also to more general scenarios involving communication resources, distributed systems, other scheduling policies such as EDF and TDMA and much more detailed characterizations of workload properties, see e.g. [14, 15].

3.2 Variability Characterization Curves

As described above, the timing characterization of event and resource streams in abstract components and Real-Time Interfaces are based on Variability Characterization Curves (VCC) which substantially generalize the classical representations such as sporadic, periodic or periodic with jitter.

An event streams is described using an arrival curve $\alpha(\Delta) \in \mathbb{R}^{\geq 0}$, $\Delta \in \mathbb{R}^{\geq 0}$ which provides an upper bound on the number of events in *any* time interval of length Δ. In particular, there are at most $\alpha(\Delta)$ events within the time interval $[t, t+\Delta)$ for all $t \geq 0$.

In a similar way, the resource availability is characterized using a service curve $\beta(\Delta) \in \mathbb{R}^{\geq 0}$, $\Delta \in \mathbb{R}^{\geq 0}$. This VCC provides a lower bound on the available service in any time interval of length Δ, i.e. in any time interval of length Δ at least $\beta(\Delta)$ events can be processed.

Following the results from network calculus and Real-Time Calculus see [8] and [13, 12], the following relations can be derived:

- **Remaining Service Curve**: If a task with arrival curve $\alpha(\Delta)$ is executed on a resource with availability $\beta(\Delta)$, then the remaining resource available to other lower-priority tasks can be bounded by the service

curve:

$$\beta'(\Delta) = \sup_{0 \le \lambda \le \Delta} \{\beta(\lambda) - \alpha(\lambda)\} \stackrel{def}{=} RT(\beta, \alpha) \quad (1)$$

- **Delay**: The maximum delay d_{max} of an event processing for arrival curve $\alpha(\Delta)$ and service curve $\beta(\Delta)$ is bounded by the maximal horizontal distance between $\alpha(\Delta)$ and $\beta(\Delta)$:

$$d_{max} \le \sup_{\Delta \ge 0} \{\inf\{\tau \ge 0 : \alpha(\Delta) \le \beta(\Delta + \tau)\}\} \stackrel{def}{=} Del(\beta, \alpha) \quad (2)$$

Note that the arrival and service curves can be determined using traces (by using a sliding window approach), analytic characterization of the corresponding stream (periodic, periodic with jitter, sporadic), by modular performance analysis or by using data sheets of the used hardware resources, see e.g. [2, 14].

Figure 1 shows examples of the arrival curves $\alpha(\Delta)$ for event-streams with some typical analytic timing characterizations. Note, that following [13], event-streams are typically described by a set of lower and upper arrival curves. Thus, timing of the event-streams in Fig. 1 can be modeled losslessly. For Real-Time Interfaces we however only consider the upper arrival curve $\alpha(\Delta)$ at the moment.

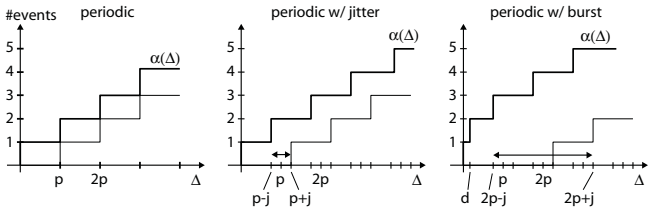

Figure 1: Upper (thick) and lower (thin) arrival curves for a periodic, a periodic with jitter and a periodic with bursts event stream.

3.3 Interface Variables and Predicates

A Real-Time Interface may have input and output variables related to event streams (arrival) and resource availability (service). The value of an arrival variable consists of the arrival curve $\alpha(\Delta)$ and the requested maximal event delay d. The associated output guarantee ϕ^O contains the bounds $\hat{\alpha}^G(\Delta)$ and $\hat{d}^G$ which are part of the component interface. The predicate guarantees that the provided arrival curve satisfies $\alpha(\Delta) \le \hat{\alpha}^G(\Delta)$ for all Δ and that the requested maximal delay d is larger than $\hat{d}^G$, see Table 1.

Table 1: Assume/guarantee predicates of the variables in a Real-Time Interface.

var. type	value	input assumption ϕ^I	output guarantee ϕ^O
arrival	α, d	$(\alpha \le \hat{\alpha}^A) \wedge (d \ge \hat{d}^A)$	$(\alpha \le \hat{\alpha}^G) \wedge (d \ge \hat{d}^G)$
service	β	$\beta \ge \beta^A$	$\beta \ge \beta^G$

In a similar way, the input predicate ϕ^I reflects the assumption that the arrival curve of the incoming event stream satisfies $\alpha(\Delta) \le \hat{\alpha}^A(\Delta)$ for all Δ, and that the requested maximum delay satisfies $d \ge \hat{d}^A$. The values of the resource variables (service) are constraint by corresponding assume/guarantee predicates, see Table 1.

In order to determine whether two Real-Time Interfaces are compatible, we need to check the relation $\phi^O \Rightarrow \phi^I$ for each connection, see Section 2. An arrival connection is compatible if

$$(\hat{d}^A \le \hat{d}^G) \quad \wedge \quad (\hat{\alpha}^A(\Delta) \ge \hat{\alpha}^G(\Delta)) \quad \forall \Delta \ge 0$$

and a service connection is compatible if

$$\hat{\beta}^A(\Delta) \le \hat{\beta}^G(\Delta) \quad \forall \Delta \ge 0$$

Two components are compatible if all internal connections are compatible and if all open input predicates and all output predicates are still satisfiable.

The next section describes the relations between the component interface functions $\hat{\alpha}^A(\Delta)$, $\hat{\alpha}^G(\Delta)$, $\hat{d}^A$, $\hat{d}^G$, $\hat{\beta}^A(\Delta)$ and $\hat{\beta}^G(\Delta)$ in a real-time component system.

4. A COMPONENT SYSTEM WITH REAL-TIME INTERFACES

Let us recall, that for the design of real-time systems, we basically distinguish three different types of system elements that are used as building blocks to build together a real-time system. Firstly, a real-time system contains a set of hardware resources, such as CPU's, DSP's or buses, that provide computing and communication services within a real-time system. Secondly, a real-time system contains a set of real-time load specifications, that describe how the system is being used by the environment, i.e. how much load arriving events on an incoming event-stream generate, and what the delay constraints of an event-stream are. And finally, a real-time system contains a set of processes, that use the available system services to process the incoming real-time event-streams. Different processes may share the available system services, and the method of sharing is defined by a scheduling policy.

4.1 Service Components

Definition 1 (Serivce Component). *A* service component *models a computing or communication resource. The Real-Time Interface of a service component has a single service output variable with the output guarantee*

$$\phi^O = (\beta \ge \hat{\beta}^G) \quad (3)$$

Through the output guarantee of its Real-Time Interface, a service components expresses that the service $\beta(\Delta)$, that is provided by the component on its service output, is always larger or equal $\hat{\beta}^G(\Delta)$ for any time interval Δ. The left-most component in Figure 2 depicts a service component.

4.2 Load Components

Definition 2 (Load Component). *A* load component *models a real-time load specification. The Real-Time Interface of a load component has a single arrival output variable with the output guarantee*

$$\phi^O = (\alpha \le \hat{\alpha}^G) \wedge (d \ge \hat{d}^G) \quad (4)$$

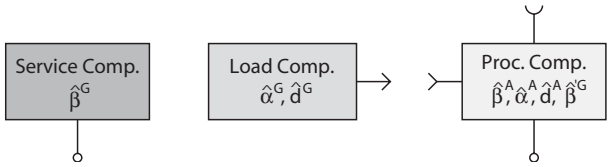

Figure 2: A service component (left), a load component (middle) and a process component (right).

Through the output guarantee of its Real-Time Interface, a load component expresses that the load $\alpha(\Delta)$ that will be emitted through its arrival output is always less or equal $\hat{\alpha}^G(\Delta)$ for any time interval Δ, and that the delay requirement d for this load is greater or equal $\hat{d}^G$. The component in the middle of Figure 2 depicts a load component.

4.3 Process Components for FP

Definition 3 (Process Component for FP). *A process component for preemptive fixed priority scheduling models a process in a real-time system, that shares system services with a fixed priority scheduling strategy, and that uses the available system services to process real-time loads. The Real-Time Interface of a process component for FP scheduling has an arrival input variable, a service input variable, and a service output variable. The interface has the input assumption*

$$\phi^I = (d \geq \hat{d}^A) \wedge (\alpha \leq \hat{\alpha}^A) \wedge (\beta \geq \hat{\beta}^A) \tag{5}$$

and the output guarantee

$$\phi^O = (\beta' \geq \hat{\beta}'^G) \tag{6}$$

With the input assumptions and the output guarantees of its Real-Time Interface, a process component for fixed priority scheduling expresses that whenever (a) the service $\beta(\Delta)$ that is provided to the component on its service input is larger or equal $\hat{\beta}^A(\Delta)$ for any Δ and (b) the load $\alpha(\Delta)$ that arrives at the component on its arrival input is less or equal $\hat{\alpha}^A(\Delta)$ for any Δ and has a maximum required delay that is greater or equal $\hat{d}^A$, then (i) the arriving load can be processed in real-time, i.e. with a guaranteed delay $d_{max} \leq \hat{d}^A \leq \hat{d}^G$, and (ii) the service $\beta'(\Delta)$ that is provided by the component on its service output is always larger or equal $\hat{\beta}'^G(\Delta)$ for any time interval Δ, and can be used by lower-priority processes.

From the different component interfaces in Fig. 2, we see that we can build interface models of real-time systems by connecting the arrival output of load components to the arrival input of process components, and by connecting the service output of service and process components to the service input of process components. As already alluded above, the order in which process components are connected determine their priority for preemptive fixed priority scheduling on a resource. The process component that is directly connected to a service component is processed with priority $P = 1$, while the next lower process component only gets the service that is left over by the first process component, and is therefore processed with priority $P = 2$. The next lower process component is then processed with priority $P = 3$, and so on.

During composition of a process component interface with other service, arrival and process component interfaces, the different interfaces share information on their input assumptions and output guarantees. A part of an interface composition with the involved information sharing is depicted in Fig. 3.

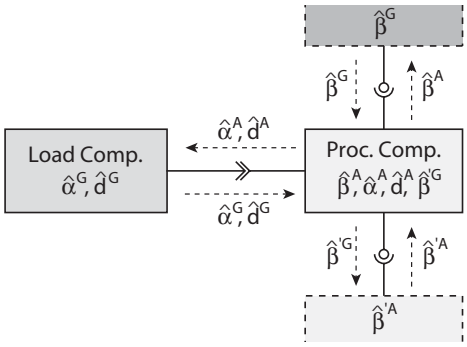

Figure 3: Information-sharing between interface components during composition.

During composition with other interfaces, the equations (7)-(10) define the relations between input assumptions and output guarantees in every process component for preemptive fixed priority scheduling:

$$\hat{\beta}'^G = RT(\hat{\beta}^G, \hat{\alpha}^G) \tag{7}$$

$$\hat{\beta}^A = \max\left\{\hat{\alpha}^G(\Delta - \hat{d}^G), RT^{-\beta}(\hat{\beta}'^A, \hat{\alpha}^G)\right\} \tag{8}$$

$$\hat{\alpha}^A = \min\left\{\hat{\beta}^G(\Delta + \hat{d}^G), RT^{-\alpha}(\hat{\beta}'^A, \hat{\beta}^G)\right\} \tag{9}$$

$$\hat{d}^A = Del(\hat{\beta}^G, \hat{\alpha}^G) \tag{10}$$

with

$$RT^{-\alpha}(\beta', \beta)(\Delta) = \beta(\Delta + \lambda) - \beta'(\Delta + \lambda) \quad \text{for} \quad \lambda = \sup\left\{\tau \ : \ \beta'(\Delta + \tau) = \beta'(\Delta)\right\} \tag{11}$$

and

$$RT^{-\beta}(\beta', \alpha)(\Delta) = \beta'(\Delta - \lambda) + \alpha(\Delta - \lambda) \quad \text{for} \quad \lambda = \sup\left\{\tau \ : \ \beta'(\Delta - \tau) = \beta'(\Delta)\right\} \tag{12}$$

While (7) and (10) can be derived directly from (1) and (2) that are well-known results from the area of network calculus [8], equations (8) and (9) for $\hat{\beta}^A$ and $\hat{\alpha}^A$ are new results.

Equation (8) is derived from the fact that $\hat{\beta}^A$ needs to satisfy the delay constraint

$$\hat{\beta}^A(\Delta) \geq \hat{\alpha}^G(\Delta - \hat{d}^G)$$

and the resource constraint

$$\hat{\beta}^A(\Delta) \geq \inf\left\{\beta \ : \ \hat{\beta}'^A(\Delta) = \sup_{0 \geq \lambda \geq \Delta}\{\beta(\lambda) - \hat{\alpha}^G(\lambda)\}\right\}$$

For taking into account the resource constraint, we construct the inverse of the resource transformation $RT(\beta, \alpha)$ with respect to β, as the smallest $\beta(\Delta)$ such that (1) holds. This inverse is denoted as $RT^{-\beta}(\beta', \alpha)(\Delta)$, and $\hat{\beta}^A$ is then the maximum of both expressions above.

Analogously, equation (9) is derived from the fact that $\hat{\alpha}^A$ needs to satisfy the delay constraint

$$\hat{\alpha}^A(\Delta) \leq \hat{\beta}^G(\Delta + \hat{d}^G)$$

and the resource constraint

$$\hat{\alpha}^A(\Delta) \leq \sup\left\{\alpha \ : \ \hat{\beta}'^A(\Delta) = \sup_{0 \geq \lambda \geq \Delta}\{\hat{\beta}^G(\lambda) - \alpha(\lambda)\}\right\}$$

For taking into account the resource constraint, we construct again the inverse of the resource transformation $RT(\beta, \alpha)$, this time with respect to α, as the largest $\alpha(\Delta)$ such that (1) holds. This inverse is denoted as $RT^{-\alpha}(\beta', \beta)(\Delta)$, and $\hat{\alpha}^A$ is then the minimum of both expressions above.

4.4 Local Compatibility Checking

The presented component system for real-time systems with fixed priority scheduling possesses a property that we will refer to as ability for *local compatibility checking*.

While syntactical composability checking is usually a computationally simple and straight-forward process for most interface theories, this is in general not the case for semantical compatibility checking of two interfaces.

For the presented component system however, semantical compatibility checking of two interfaces is computationally fast, whenever the two interfaces under consideration only share one single connection, i.e. whenever the outputs of one interface provide only one single input for the other interface, or vice versa. If this property holds, semantical compatibility checking can be done locally, i.e. only the local interface predicates of the two components that get connected must be considered during compatibility checking.

This is true, because in any composed interface of the presented component system, through evaluation of (7)-(10), the local interface predicates contain enough information of the complete system, to check the compatibility of the complete interface with any other interface that is only locally connected.

In practice, this means that when we are given two interfaces that share only one single interface connection, then the act of semantical compatibility checking of these two interfaces can be done in $O(n + n')$ or in $O(m + p)$ when the two interfaces share only a service interface connection or only an arrival interface connection, respectively. Thereby n and n' denote the number of segments that are used to describe $\hat{\beta}^G$ and $\hat{\beta}'^A$, respectively, and m and p denote the number of segments that are used to describe $\hat{\alpha}^A$ and $\hat{\alpha}^G$, respectively. In both cases, $O(n + n')$ and $O(m + p)$ are needed to check that $\hat{\beta}^G \geq \hat{\beta}'^A$ or $\hat{\alpha}^A \geq \hat{\alpha}^G$, respectively, and for arrival interfaces $O(m)$ are additionally needed to adapt (9) to $\hat{d}^G$.

We will see that this ability for efficient local compatibility checking allows for some interesting on-line applications of Real-Time Interfaces.

5. APPLICATIONS

For the sake of simplicity, the upper linear approximation of arrival curves $\alpha(\Delta)$ and the lower linear approximation of service curves $\beta(\Delta)$ are used in the examples of this Section. All methods are however also directly applicable to step-wise defined arrival and service curves.

5.1 Interface-Based Real-Time System Design

The most natural application for Real-Time Interfaces is to enable interface-based design of real-time systems. For this, suppose we are given the a set of service components, a set of load components and a set of processing components for fixed priority scheduling, together with their corresponding Real-Time Interface descriptions.

The goal of interface based-design is then to build the interface of a complete system, by composing all interfaces of the different components. If this composition is successful, a concrete instance of this system can be built from concrete components that conform to their corresponding Real-Time Interface descriptions. We are then assured that this concrete system fulfills all real-time requirements.

EXAMPLE 1. *Suppose, we want to build a real-time system, that must process three different event-streams in real-time. Event-stream* A *has a maximum burst demand of* $100'000$ *cycles, a maximum burst-rate demand of* $100'000$ *cycles per second (c/s) for* $1ms$ *and a maximum long-term demand of* $25'000$ c/s. *Event-stream* B *has a maximum burst demand of* $10'000$ *cycles, a maximum burst-rate demand of* $400'000$ c/s *for* $0.5ms$ *and a maximum long-term demand of* $75'000$ c/s. *And event-stream* C *has a maximum burst demand of* $200'000$ *cycles and a maximum long-term demand of* $50'000$ c/s. *Further, event-stream* A *must not experience a delay of more than* $0.5ms$ *when processed by the system, while the maximum acceptable delay of event-stream* B *and* C *are* $2.5ms$ *and* $4ms$, *respectively. To build this system, a CPU with a minimum resource availability of* $300'000$ c/s *(300MHz) is available.*

For an interface-based design of a system that corresponds to the above specification, we take three load components with arrival interfaces that conform to the three event-streams A, B *and* C, *a service component with a service interface that conforms to the given CPU, and three processing components with corresponding interfaces.*

Through successful interface-composition we can then show that a system that corresponds to the above specification can indeed be built on the given CPU. To implement this system, preemptive fixed priority scheduling can be used on the CPU, where processing of event-stream A *has the highest priority and processing of event-stream* C *has the lowest priority.*

The interface model of the above system is shown on the right side of Fig. 4. On the left side of the same figure, a concrete instance of the real-time system that corresponds to the interface model on the right is shown.

In Fig. 5 input assumptions of the service interfaces of the three processing components of the above system are shown, together with the output guarantee of the service interface of the CPU. In this figure, we see that $\hat{\beta}_I^A \leq \hat{\beta}_{CPU}^G$ *and that therefore the service component of the CPU in the interface model of Fig. 4 is compatible with the composed interface of the other components. Note, that* $\hat{\beta}_I^G$, $\hat{\beta}_{II}^G$ *and* $\hat{\beta}_{III}^G$ *are not shown here.*

Finally, in Fig. 6 input assumptions of the arrival interfaces of the three processing components of the system in Fig. 4 are shown together with the output guarantees of the arrival interfaces of the three event-streams. In this figure, we see that all input assumptions are strictly greater or equal than the corresponding output guarantees and allow therefore the successful composition of all load components with the composed interface of the other components.

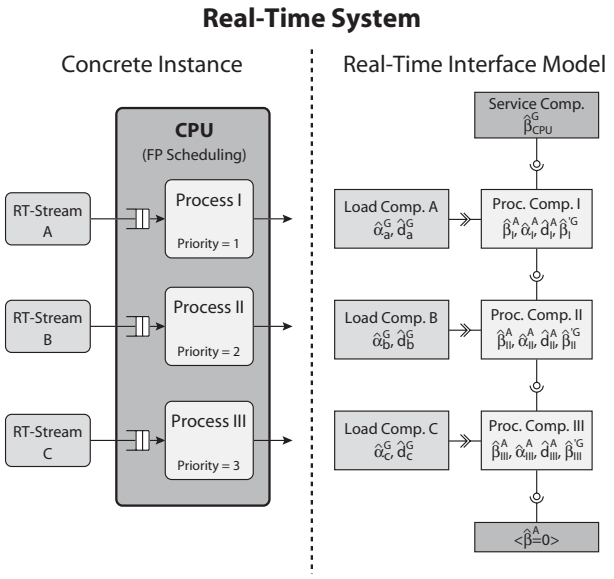

Figure 4: A concrete stream-processing real-time system and its Real-Time Interface Model.

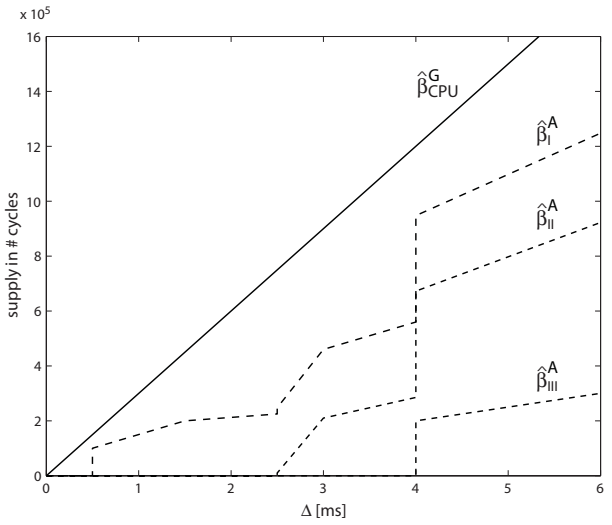

Figure 5: Output guarantee (solid) of the service component in Fig. 4, together with the input assumptions (dashed) of the three process components in the composed interface model.

Interface-based design of real-time systems benefits from all the advantages of interface-based design as described in [7]. Most notable of all, interface-based design of real-time systems supports *incremental design* and *independent implementability* of real-time systems.

The property of incremental design ensures that the compatible components of a system can be put together in any order [7]. In practice, this allows for example to design an economic real-time system by first composing all load components and process components. This leads to an interface model with only open service inputs. By looking at

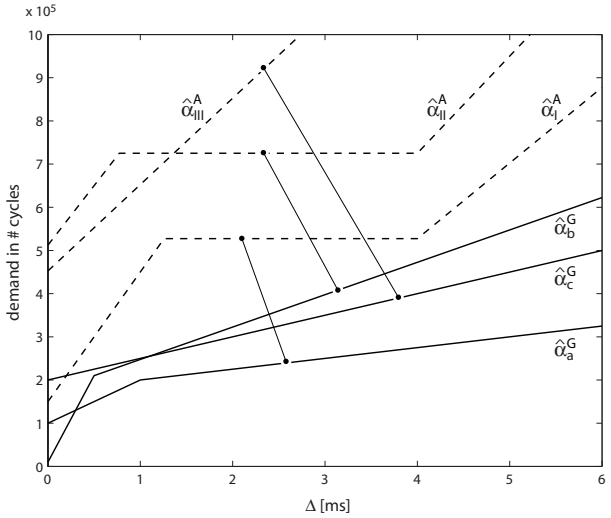

Figure 6: Output guarantees (solid) of the three load components in Fig. 4, together with the corresponding input assumptions (dashed) of the three process components in the composed interface model.

the input assumptions on these open service inputs, we can then directly find the tightest service interfaces with output guarantees that just still allow composition. By choosing the most economic service components, i.e. processors, that still conform to these tight service interfaces, we directly obtain an economic real-time system that guarantees to fulfil all real-time requirements, without being over-dimensioned.

This design procedure is a straight-forward process and stands in contrast to traditional real-time system design, where service-components are chosen a-priori, and performance analysis methods like [10] or [13] are then used to decide whether a system design fulfills all real-time requirements or not. In this traditional approach, economic designs must be found by binary search, i.e. by parameter sweeps (see e.g. [16]).

The property of independent implementability on the other hand ensures that compatible interfaces can always be refined independently [7]. In practice, this allows to outsource the implementation of different system components, or to replace existing implementations of system components with different or new implementations. As long as the implementations of all components conform to their respective interfaces, the components will fit together and the complete integrated system is guaranteed to fulfill all real-time requirements.

5.2 On-Line Service- & Load-Adaptation

Real-Time Interfaces can not only be used for interface-based design of real-time systems, but due to the ability of local compatibility checking (see Section 4.4), there exist also a wide range of interesting on-line applications for Real-Time Interfaces.

For on-line applications of Real-Time Interfaces, an interface model of the complete real-time system under consideration is deployed together with the concrete system, and is kept up-to-date at all times. The information contained in this interface model can then be used to permit on-line

adaptation of different parameters of the running real-time system. This application of Real-Time Interfaces may be particularly interesting for power-aware real-time systems.

In the case of on-line service adaptation, the information contained in service interfaces is exploited. The service input assumption of any process component that is directly connected to a service component specifies the detailed service that is required from the service component over time, to guarantee real-time processing. With the help of Real-Time Interfaces, a service component can therefore lower its output guarantee $\hat{\beta}^G$ at run-time, as long as it remains greater or equal as the input assumption $\hat{\beta}^A$ of the directly connected process component. In practice, this allows for example a CPU to optimally adapt its frequency at run-time, with the guarantee to not violate any real-time requirements.

This adaptation of the output guarantee $\hat{\beta}^G$ does of course not refine the original interface of the service component, but does instead coarsen it and leads to a new interface for the service component. Because of this, the composition of the remaining system interface with the new interface of the adapted service component must be computed to obtain again an interface model of the complete real-time system. But due to the ability of local compatibility checking, the compatibility of the new service component interface with the remaining system interface can be guaranteed a priori, and the service component can already adapt its service guarantee, before computation of the interface composition of the complete system is finished.

EXAMPLE 2. *Figure 7 shows the input assumption $\hat{\beta}_I^A$ of process component* I *of the real-time system in Fig. 4. Also shown is the output guarantee $\hat{\beta}_{CPU}^G$ of the service component that models the $300MHz$ CPU of the system. We know that any service component with an output guarantee $\hat{\beta}^G \geq \hat{\beta}_I^A$, i.e. that lies completely inside the grey shaded area, will guarantee real-time processing. We can therefore directly lower the speed of the CPU down to $240MHz$, and we can be sure that this adaptation will not lead to any violations of real-time requirements.*

The same strategy as for service adaptation can be used for on-line load adaptation (QoS-adaptation). In this case, the information contained in arrival interfaces is exploited. The load input assumption of any process component specifies the detailed load that can be processed over time with the given real-time guarantee. With the help of Real-Time Interfaces, a load component can therefore increase its output guarantee $\hat{\alpha}^G$, at run-time, as long as it remains lower or equal as the input assumption $\hat{\alpha}^A$ of the connected process component. And analogously, a load component can lower its output guarantee $\hat{d}^G$ at run-time, as long as the it remains greater or equal as the input assumption $\hat{d}^A$ or the process component. In practice, this allows an event-stream to increase its data-rate, or to get a guaranteed shorter processing delay.

Like the on-line service adaptation, this on-line load adaptation requires to compute the composition of the remaining system interface with the interface of the new load component to obtain again an interface model of the complete real-time system. This composition can again be computed after the load adaptation took place.

EXAMPLE 3. *Figure 8 shows the input assumption $\hat{\alpha}_{II}^A$ of process component* II *of the real-time system in Fig. 4, and*

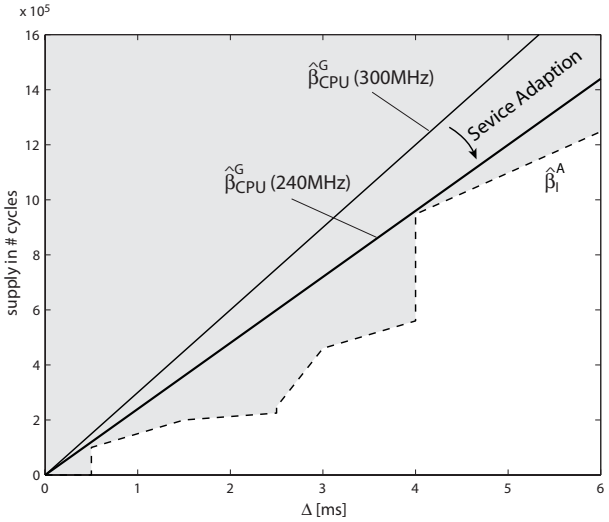

Figure 7: On-line service adaptation of the CPU in Fig. 4.

the output guarantee $\hat{\alpha}_b^G$ of the load component that models the event-stream B. *We know that any load component with an output guarantee $\hat{\alpha}^G \geq \hat{\alpha}_{II}^A$, i.e. that lies completely inside the grey shaded area, is guaranteed to be processed in real-time. We can therefore directly increase the maximum long-term demand of event-stream* B *from* $75'000$ c/s *to* $140'000$ c/s. *Further, since $\hat{d}_{II}^A = xxms$, event-stream* B *can also get the guarantee of a maximum processing delay of e.g.* $1ms$, *compared to the initially guaranteed* $2.5ms$. *For this, the event stream must set $\hat{d}_b^G = 1ms$, which results in an adaptation of $\hat{\alpha}_{II}^A$ that is however still guaranteed to be strictly greater or equal than $\hat{\alpha}_B^G$.*

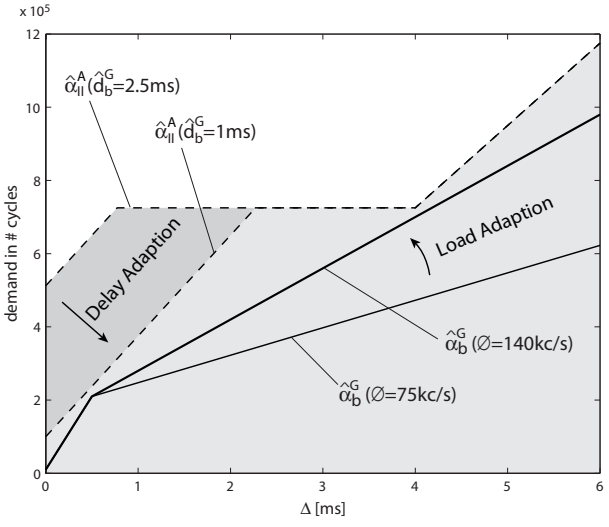

Figure 8: On-line load adaptation of stream B in Fig. 4.

For power-aware real-time systems a combination of Real-Time Interface enabled on-line service and on-line load adap-

tation could also be interesting. Using on-line service adaptation, the speed of the processor in the system would be kept at a minimum at all times. When one of the event-streams would need to increase its load, it would adapt (coarse) the output guarantee of its arrival interface. In the system interface, we would then disconnect the service component and would compose the adapted arrival interface with the remaining system interface. After this composition, the input assumption on the now open service input of the system interface would specify the detailed service that is required from the service component over time, to guarantee real-time processing of the system with the increased load. The output guarantee of the service component could then be increased on-line, to meet this service requirement. After this on-line service adaptation, the event-stream could be permitted to increase its load.

5.3 On-Line Admission Tests

Another on-line application of Real-Time Interfaces is enabled by the property of incremental design, that allows Real-Time Interfaces to be used for on-line admission tests on running real-time systems. For this, again an interface model of the complete real-time system under consideration is deployed together with the concrete system. To enable on-line admission tests with this system interface, we add a dummy process component at every existing service interface connection in this interface. The load inputs of these dummy processes are left open. Part of such an extended system interface is depicted in Fig. 9.

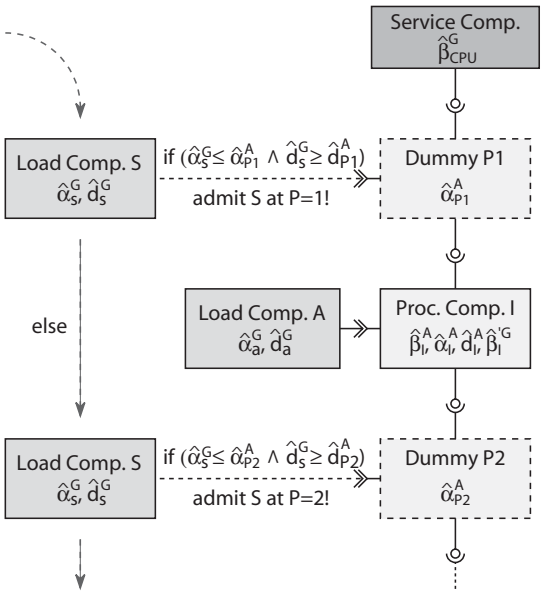

Figure 9: On-line admission tests with Real-Time Interfaces.

Because of the property of incremental design, these additional process components, that add open load inputs to the system interface, have no influence to the compatibility of the initial system interface. And still, the load input assumption $\hat{\alpha}^A$ on the open load inputs of the different dummy components carry valuable information. These load input assumptions specify the detailed load over time that could be processed on the system with real-time guarantee, if the respective dummy component would be replaced by a process component.

In practice, the open load input $\hat{\alpha}^A_{P1}$ of the dummy component *P1* in Fig. 9, specifies for example the detailed load over time that could be processed by a new process component if scheduled with priority $P = 1$. Analogous, $\hat{\alpha}^A_{P2}$ specifies the load that could be processed by a process with priority $P = 2$, and so on.

For on-online admission tests with the extended system interface, the output guarantee $\hat{\alpha}^G_s$ of the interface of a new load component S must be checked against the input assumptions $\hat{\alpha}^A_{Pj}$ of all dummy components. If $\hat{\alpha}^G_s \leq \hat{\alpha}^A_{Pj}$, then a new real-time load that conforms to the arrival interface of S can be admitted, and the process that processes this load can be run with priority $P = j$.

After admission of the new load S, the composition of the interface of S with the remaining system interface must be computed, and new dummy components must be added before and after the dummy process that became the new process component.

EXAMPLE 4. *Figure 10 shows the output guarantee $\hat{\alpha}^G_s$ of an event stream* S *that wants to be admitted for processing on the real-time system in Fig. 4 and that has a delay requirement of $\hat{d}^G_s = 1ms$. Also shown are the input assumptions $\hat{\alpha}^A_{P1}$-$\hat{\alpha}^A_{P4}$ of dummy components that were added at the different service interface connections in the interface in Fig. 4. By checking the output guarantee against the different input assumptions in Fig. 10, we see that the event stream* S *can be admitted. But for the system to still fulfill all real-time requirements, the process that processes* S *can only be scheduled with priority $P = 2$.*

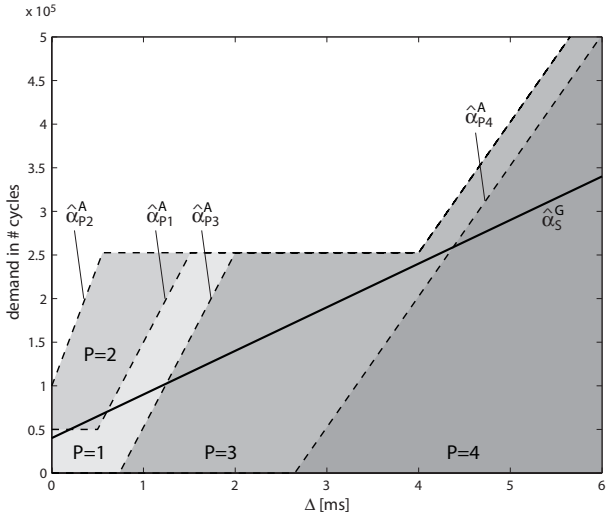

Figure 10: Admission test of an event-stream S in the real-time system in Fig. 4.

5.4 Computational Complexity

To analyze the different examples in this section, we used a prototype implementation of Real-Time Calculus and Real-Time Interfaces. This prototype is implemented in Java and uses Matlab as user frontend. With this prototype tool, it took less than 1*s* for each example to compose and analyze it, using Matlab 7 on a Pentium Mobile 1.6 GHz.

6. CONCLUSIONS AND FUTURE WORK

We presented and defined the notion of Real-Time Interfaces, that are a special instance of assume/guarantee interfaces. Real-Time Interfaces as proposed in this paper connect the principles of Real-Time Calculus and interface-based design and can therefore fall back to the wide range of already established research results in these two areas. We also introduced a component model with Real-Time Interfaces that defines the building blocks for interface-based design of real-time systems, and in this component model, we defined different components that model hardware resources, event-streams and software-processes for preemptive fixed-priority scheduling, respectively. We have shown the applicability of Real-Time Interfaces in the area of interface-based design and we further showed that a range of on-line applications in real-time systems could profit from the use of Real-Time Interfaces.

In future, we would like to extend Real-Time Interfaces to include additional information, as for example required buffer and memory. Further, we want to introduce new components for EDF and TDMA scheduling policies as well as for communication hardware resources. Finally, a Java implementation of Real-Time Calculus including Real-Time Interfaces is currently under development.

Acknowledgements

This research has been funded by the Swiss National Science Foundation (SNF) under the Analytic Performance Estimation of Embedded Computer Systems project 200021-103580/1, and by ARTIST2.

7. REFERENCES

[1] A. Chakrabarti, L. de Alfaro, T. Henzinger, and M. Stoelinga. Resource interfaces. In *EMSOFT 03: Embedded Software*, Lecture Notes in Computer Science 2855, pages 117–133. Springer-Verlag, 2003.

[2] S. Chakraborty, S. Künzli, and L. Thiele. A general framework for analysing system properties in platform-based embedded system designs. In *Proc. 6th Design, Automation and Test in Europe (DATE)*, pages 190–195, March 2003.

[3] S. Chakraborty, S. Künzli, L. Thiele, A. Herkersdorf, and P. Sagmeister. Performance evaluation of network processor architectures: Combining simulation with analytical estimation. *Computer Networks*, 41(5):641–665, April 2003.

[4] L. de Alfaro and T. Henzinger. Interface theories for component-based design. In *EMSOFT 01: Embedded Software*, Lecture Notes in Computer Science 2211, pages 148–165. Springer-Verlag, 2001.

[5] L. de Alfaro, T. Henzinger, and M. Stoelinga. Timed interfaces. In *EMSOFT 02: Embedded Software*, Lecture Notes in Computer Science 2491, pages 108–122. Springer-Verlag, 2002.

[6] L. de Alfaro and T. A. Henzinger. Interface automata. In *Proc. Foundations of Software Engineering*, pages 109–120. ACM Press, 2001.

[7] L. de Alfaro and T. A. Henzinger. Interface-based design. In *To appear in the Proceedings of the 2004 Marktoberdorf Summer School*. Kluwer, 2005.

[8] J. Le Boudec and P. Thiran. *Network Calculus - A Theory of Deterministic Queuing Systems for the Internet*. LNCS 2050, Springer Verlag, 2001.

[9] C. Lee, M. Potkonjak, and W. H. Mangione-Smith. Mediabench: A tool for evaluating and synthesizing multimedia and communicatons systems. In *International Symposium on Microarchitecture*, pages 330–335, 1997.

[10] K. Richter, M. Jersak, and R. Ernst. A formal approach to mpsoc performance verification. *IEEE Computer*, 36(4):60–67, April 2003.

[11] I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In *Proceedings of the Real-Time Systems Symposium (RTSS)*, pages 2–13. IEEE Press, 2003.

[12] L. Thiele, S. Chakraborty, M. Gries, A. Maxiaguine, and J. Greutert. Embedded software in network processors – models and algorithms. In *Proc. 1st Workshop on Embedded Software (EMSOFT)*, Lecture Notes in Computer Science 2211, pages 416–434, Lake Tahoe, CA, USA, 2001. Springer Verlag.

[13] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 4, pages 101–104, 2000.

[14] E. Wandeler, A. Maxiaguine, and L. Thiele. Quantitative characterization of event streams in analysis of hard real-time applications. *Real-Time Systems*, 29(2-3):205–225, March 2005.

[15] E. Wandeler and L. Thiele. Abstracting functionality for modular performance analysis of hard real-time systems. In *Asia South Pacific Design Automation Conference (ASP-DAC)*, 2005.

[16] E. Wandeler, L. Thiele, M. Verhoef, and P. Lieverse. System architecture evaluation using modular performance analysis - a case study. In *1st International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, 2004.

[17] S. Wang, S. Rho, Z. Mai, R. Bettati, and W. Zhao. Real-time component-based systems. In *Proceedings of the 11th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 428–437. IEEE Press, 2005.