Interface-Based Rate Analysis of Embedded Systems

Samarjit Chakraborty and Yanhong Liu
Department of Computer Science
National University of Singapore
Email: {samarjit,liuyanho}@comp.nus.edu.sg

Nikolay Stoimenov, Lothar Thiele and Ernesto Wandeler
Computer Engineering and Networks Laboratory
ETH Zurich, Switzerland
Email: {nikolays,thiele,wandeler}@tik.ee.ethz.ch

***Abstract*— Interface-based design is now considered to be one of the keys to tackling the increasing complexity of modern embedded systems. The central idea is that different components comprising such systems can be developed independently and a system designer can connect them together only if their interfaces match, without knowing the details of their internals. We use the concept of real-time interfaces to establish *rate interfaces* which can be used for compositional (correct-by-construction) design of embedded systems whose components communicate through data streams. Using the associated rate interface algebra, two components can be connected together if the *output rate* of one component is "compatible" with the *input rate* of the other component. We formalize this notion of compatibility and show that such an algebra is non-trivial because it has to accurately model the burstiness in the arrival rates of such data streams and the variability in their processing requirements. We discuss how rate interfaces simplify compositional design and at the same time help in functional and performance verification which would be difficult to address otherwise. Finally, we illustrate these advantages through a realistic case study involving a component-based design of a multiprocessor architecture running a picture-in-picture application.**

I. Introduction

Most embedded systems today consist of a heterogeneous collection of processing elements, application-specific hardware accelerators and memory modules, which are connected together using some communication subsystem. Typically these components are designed by different vendors, and hence integrating them together and ensuring that the system works correctly often turns out to be a challenging problem. To address this problem, recently there has been a considerable emphasis on *interface-based design* [7]. Here, the basic premise is that a system designer using a component only needs to understand the component's *interface* and not the details of how the functionality offered by the component is implemented. In other words, it should be possible to integrate a set of components if their interfaces "match". This gives rise to two related questions: i) What kind of information about a component should be exposed by its interface? ii) How is the notion of two interfaces "matching" technically formulated and realized? Answers to these questions clearly depend on the system being designed. Hence, lately there has been a number of proposals on different kinds of interface specifications and on what constitutes a good notion of "match" for different system design scenarios, see for example [8], [4], [16], [17].

Following this line of work, in this paper we extend the concept of real-time interfaces [16], [17] to propose what we refer to as *rate interfaces*. To check if two such interfaces match, we resort to an analysis technique called *rate analysis*. The key feature of *rate interfaces* is a specification of the allowed input *rate* at which data may arrive at a component and the *rate* at which such data gets processed by the component. The processed data may then serve as input to other components. Two components can be composed together if the output rate of the first component is *compatible* with the input rate of the second component. Later in this paper we will precisely define what "compatibility" of input and output rates mean in the context of our work. But here we would like to point out that "compatible" does not necessarily mean "equal", and very often compatible rates are *not* equal.

The abstractions offered by *rate interfaces* are particularly suited for component-based design of systems that process continuous data streams. Examples of these might be sensors sensing at a pre-specified rate and sending the data to an actuator for processing (or triggering certain tasks). Other examples might be media processors, network processors, etc., which consist of multiple processing elements (PEs) with each of them running one or more tasks. An input data stream gets processed at the first PE, the partially processed stream then enters the second PE for further processing, and this continues until the stream is fully processed and leaves the system. The timing properties (or the output *rate*) of a partially processed stream coming out of a PE might be very different from the properties of the input stream. Such a transformation depends on the tasks running on the PE, the scheduling policy used and the architecture of the PE [13]. In this setup, two PEs can be composed (i.e. work together) if the timing properties of the data stream coming out of the first PE can be guaranteed to be compatible with the timing properties of the input data stream expected by the second PE. Here, compatibility may be defined, for example, as a constraint that the buffer between the two PEs should not overflow.

The algebra that we present in this paper can be used to effectively verify such compatibility conditions by only analyzing the interfaces of the components. Further, it can also be verified if one of the components can be replaced by a different component, which is often referred to as component *refinement*. This might involve, for example, changing the scheduler in the component or changing its hardware architecture. In contrast to global (or monolithic) system verification, such compatibility and refinement checks significantly ease component integration, design-space exploration and resource dimensioning.

In this paper, *rates* not only capture the average arrival rates of data streams, but also accurately specify the burstiness

in the arrival of a stream over different time scales. Such a detailed specification is necessary for our framework to be useful because on-chip traffic and processing requirements of applications tend to be highly bursty in modern component-based embedded systems (see, for example, [14], [13], [15]). However, this also necessitates an involved algebra for reasoning about such rate specifications.

Relation to previous work: There have been a few previous attempts to analyze real-time systems in terms of their input/output data rates. More specifically, the *rate analysis* proposed in [10] considered a collection of concurrently executing components that interact through synchronization messages. The problem is then to compute bounds on the execution rates of these components under certain resource constraints. Alternatively, given a set of rate constraints, the problem is to efficiently check if these constraints are consistent. Similarly, [2] addressed the problem of identifying task activation rates for fixed-priority scheduled systems with deadline constraints.

This paper is largely motivated by our previous efforts to study such rate analysis for buffer-constrained architectures in the context of multimedia processing [9], [11]. In contrast to [10] and [2], which deal with resource and deadline constraints, the problem addressed in [9], [11] was concerned with determining upper and lower bounds on the arrival rates of multimedia streams such that certain buffer overflow and underflow constraints are satisfied. The model specifying the arrival rate (or timing properties) of a stream in [11] was refined in [9], thereby leading to tighter results.

The interface algebra presented here is an interface-theoretic formulation of the rate analysis problem studied in [9], [11]. In particular, we use the concept of *real-time interfaces* that was recently proposed in [16], [17] to address the rate analysis problem in the context of component-based system design. *Real-time interfaces* in turn were built on the theory of assume/guarantee (or A/G) interfaces introduced in [6], to specify real-time properties of components. A/G interfaces consist of a set of input and output variables, and a predicate ϕ^I on the input variables which states the constraints on the input that a component *assumes*. Another predicate ϕ^O on the output variables represents the *guarantee* that the component will only provide outputs which satisfy ϕ^O. The work in [16], [17] is extended as we allow distributed systems where tasks are distributed to several processing elements.

Contributions: In summary, the main contributions of this paper are as follows:

- We formulate the rate analysis problem in an interface-theoretic setting. It allows for compositional design of embedded systems whose components communicate through event streams. This translates into two components being composable if their input and output data rates are *compatible*. We show that such compatibility of two rate interfaces can be effectively checked and compatible interfaces guarantee buffer overflow and underflow constraints.

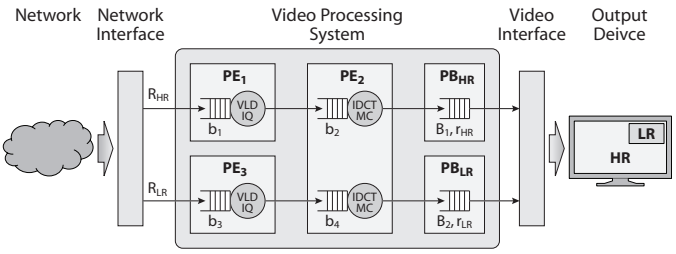

Fig. 1. A picture-in-picture (PiP) application decoding two MPEG-2 video streams on a multiprocessor architecture.

- We extend the theory of real-time interfaces [16], [17] to a distributed system implementation. Therefore, we not only consider tasks that are executed on a single processing element but allow for data streams that are processed on several computing resources. This way, independent composition in terms of data streams and resources is possible.
- Lastly, we show that the interface-theoretic formulation of the rate analysis problem has several advantages. These include the possibility of component-level analysis, which is computationally more efficient than the global, monolithic analysis proposed in [9], [11]. This implies easier component-level design space exploration and resource dimensioning. In other words, questions like "if the scheduler in one particular component is changed, then will the buffer constraints still be satisfied?" can now be answered more efficiently. We also show that this formulation enables the modeling of different component-level scheduling policies.

Organization of the paper: The rest of this paper is organized as follows. In the next section we present a motivating example that is used throughout the paper to illustrate the proposed theory. In Section III we outline the basics of A/G interfaces and in Section IV show how they are extended to specify timing properties of components in *real-time interfaces*. This is followed by our theory of rate interfaces in Section V. In Section VI we present several extensions of basic rate interfaces to support multiple scheduling policies within a component. Finally, in Section VII we present a case study using the motivating example introduced earlier. This case study illustrates how our proposed interfaces may help in resource dimensioning, incremental design and component refinement. Section VIII summarizes the paper and outlines some directions for future work.

II. Illustrative Example

Here we present an example to illustrate typical hardware/software architectures that are amenable to component-based design using rate interfaces. After describing our rate interface algebra in the subsequent sections, we return to this example to present its component-based realization, and show how components can be composed using their rate interfaces.

Figure 1 shows an architecture consisting of three processing elements PE_1, PE_2 and PE_3 onto which an MPEG-2 decoder has been partitioned and mapped. This architecture

implements a picture-in-picture (PiP) application where two concurrent video streams are being decoded and displayed on the same output device. The variable length decoding (VLD) and inverse quantization (IQ) tasks of the decoder have been mapped onto PE_1 and also replicated on PE_3. Each of these two PEs process a different video stream. PE_2, on the other hand, implements the inverse discrete cosine transform (IDCT) and the motion compensation (MC) tasks and processes both the streams. A scheduler implemented on PE_2 schedules these streams, typically using different QoS parameters for each stream. The stream corresponding to the main window in the output display device is typically associated with a higher frame resolution (indicated as "HR" in Figure 1) and generates a higher workload on PE_2, compared to the lower resolution video ("LR"in the figure) associated with the secondary window.

As shown in the figure, the video streams arrive over a network and enter the system after some initial packet processing at the network interface. The inputs to PE_1 and PE_3 are compressed bitstreams and their outputs are partially decoded macroblocks, which serve as input to PE_2. The fully decoded video streams are written into two playout buffers PB_{HR} and PB_{LR} associated with the high and low resolution streams respectively. These buffers are read by the display device at pre-specified constant frame rates r_{HR} and r_{LR}.

Note that each task running on a PE is allocated some buffer which resides inside the PE. Such buffers are used to store the incoming stream to be processed by the PE. Hence, a typical design constraint that needs to be satisfied while connecting these PEs is that none of the buffers should overflow. In addition, the playout buffers should not underflow because that would result in the output device having to stall.

Component-based design: In this paper we show that following a component-based design approach, each PE and playout buffer may be considered to be an independent component with well-defined interfaces. To connect these components together in order to realize an architecture like the one shown in Figure 1, a designer only needs to check if their interfaces are compatible. If these interfaces are well-designed, then such compatibility would guarantee that the buffers inside the components would not overflow and the playout buffers would not underflow. Our goal in this paper is to design such interfaces. By designing appropriate interfaces, it is also possible to satisfy other non-functional or performance constraints such as the utilization of the components should be above a certain threshold or that the data streams being processed should meet their deadlines.

We also show that the proposed interfaces can be used to efficiently answer questions such as: Given the input arrival rates of the streams and the consumption rates by the output device, what is the optimal buffer size for a component that is being added to a partially-designed architecture? Can the scheduler in the component PE_2 be replaced by a different scheduler without violating the buffer constraints of all the existing components? Note that answering these questions at the component level—especially for complex architectures—is much more efficient than designing the complete architecture, and then doing a global performance analysis followed by re-designing the architecture if it does not satisfy the specified performance constraints.

III. Assume/Guarantee Interfaces

In this section we outline the basic principles of assume/guarantee (or A/G) interfaces (as proposed in [6]) and briefly discuss how we adapt them in the context of our setup, where components communicate via continuous data streams and have limited buffers inside them.

Each A/G interface consists of two disjoint sets of input and output variables denoted by X^I and X^O respectively. The interface makes certain *assumptions* on X^I, which are specified using a predicate ϕ^I. Provided this predicate is satisfied, the interface *guarantees* that the output variables will satisfy a predicate ϕ^O. Hence, ϕ^O is the guarantee that the component provides to the environment assuming the precondition ϕ^I. In other words, $\phi^I \Rightarrow \phi^O$ is true. Clearly, the predicate ϕ^I need not be *valid*, i.e. there might exist environments where this component cannot be used or cannot provide the output guarantee. Environments or other components which satisfy ϕ^I are environments where this component can be used. In what follows, we do not distinguish between components and their interfaces. Hence, an interface implicitly refers to the component it belongs to.

Two interfaces F and G are composed by connecting the output variables of one to the input variables of the other and the composed interface is typically denoted as $F\|G$. The input variables of this composed interface are all the unconnected (or free) input variables of F and G, and its output variables are the unconnected outputs of F and G. We will use X_F^I, X_F^O, ϕ_F^I and ϕ_F^O to denote the variables and predicates of F and similarly for G.

Two interfaces can be syntactically composed if their output variables are disjoint. Interfaces F and G are semantically composable if whenever F provides inputs to G, $\phi_F^O \Rightarrow \phi_G^I$. A similar implication should also hold if G provides inputs to F. If F and G form a closed system, i.e. all outputs of F and connected to the inputs of G and vice-versa, then for F and G to be composable, the following closed formula should be true:

$$(\forall X_F^O \cup X_G^O)(\phi_F^O \wedge \phi_G^O \;\Rightarrow\; \phi_F^I \wedge \phi_G^I) \tag{1}$$

If $F\|G$ is open, i.e. it has free input variables then formula (1) should be *satisfiable*. In other words, there must exist *some* environment in which F and G can be composed. Formula (1) is hence the weakest precondition on the environment of $F\|G$ for the two interfaces to be composable. Therefore, the assumption $\phi_{F\|G}^I$ on the input variables of the composed interface is given by formula (1). A component H can now provide inputs to $F\|G$ if $\phi_H^O \Rightarrow \phi_{F\|G}^I$ is satisfiable. This way, a system can be incrementally designed by adding one component at a time and verifying if the newly added component is compatible with the existing partially designed system.

A/G interfaces in our setup: In our setup, each component receives as input one or more data streams, processes them, and the processed data streams enter other components for further processing (see Figure 1). Each component has a certain processing capability and allocates this capability among the incoming streams according to a predefined scheduling policy. Components also have a specified amount of buffer to hold the incoming streams. As mentioned in Section I, the interface variables X^I and X^O represent rates or timing properties of the incoming and outgoing streams. Hence, the component composability conditions translate into predicates involving these data rates. The predicate ϕ^I specifies constraints on the rates of the incoming streams and ϕ^O specifies guarantees that a component provides on the rates at which the processed streams can be consumed by other components. The composability of two components implicitly guarantees that the buffers inside them do not overflow and for certain components they also do not underflow.

IV. Real-Time Interfaces

Real-time interfaces were first introduced in [16] and combine principles of interface-based design with principles of real-time system analysis. They can be considered as a special instance of assume/guarantee interfaces, tailored towards assumptions and guarantees on the throughput and delay of events, and the availability of resources. In [17] a three-step approach is presented that yields to a component system for interface-based design of real-time systems:

1) First we need to define an abstract component that describes the real-time properties of a concrete HW/SW system component. This entails defining proper abstractions for component inputs and outputs as well as internal component relations that meaningfully relate abstract inputs to abstract outputs.
2) To derive the interface of an abstract component we then need to define interface variables as well as input and output predicates on these interface variables.
3) Finally, we need to establish the internal interface relations that relate incoming guarantees and assumptions to outgoing guarantees and assumptions of the interface.

The abstract component model and the associated interface model for the picture-in-picture application of Figure 1 are depicted in Figure 2.

A. Abstract Real-Time Components

In real-time systems, as we consider them in this work, event streams are processed on a sequence of HW/SW components that we will interpret as tasks executing on possibly different hardware resources. Figure 3(a) depicts such a component. An event or data stream described by the cumulative function $R(t)$ enters the input buffer of the component and is eventually processed by the component that is executed on a hardware resource whose availability is described by the cumulative function $C(t)$. Here, $R(t)$ denotes the number of received events (or data items) in time interval $[0, t)$ and $C(t)$ the number of events that could be processed in $[0, t)$.

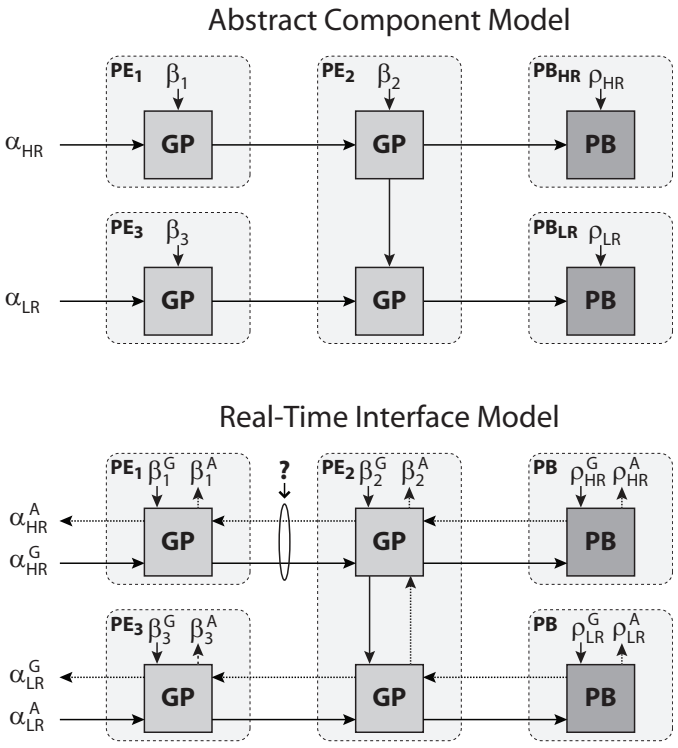

Fig. 2. A component model (top) and the corresponding interface model (bottom) of the architecture in Figure 1.

After being processed, events are emitted on the component's output, resulting in an outgoing event stream $R'(t)$, and the remaining resources that were not consumed are available to be used by other components and are described by an outgoing resource availability trace $C'(t)$. The relations between $R(t)$, $C(t)$, $R'(t)$ and $C'(t)$ depend on the component's processing semantics, and typically the outgoing event stream $R'(t)$ will not equal the incoming event stream $R(t)$, as it may for example exhibit more or less jitter.

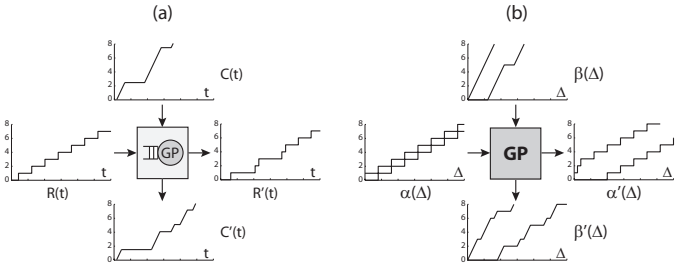

Fig. 3. (a) A concrete component, processing an event stream on a resource. (b) An abstract component, processing an abstract event stream on an abstract resource.

1) Variability Characterization Curves: For real-time system analysis, we model such a HW/SW component as an abstract component as depicted in Figure 3(b). While cumulative functions such as $R(t)$ or $C(t)$ describe one concrete trace of an event stream or a resource availability, variability characterization curves (VCC) provide means to capture all possible traces within an event stream or of a resource

availability, see [12]. In the context of this paper, we use the notion of arrival curves and service curves, see [3], which are special cases of VCCs. They are considerably more expressive than, for example, traditional event stream models. While these typically only specify a period and a jitter to model an event stream, arrival curves and service curves can accurately characterize the detailed burstiness and variability of event streams.

Arrival curves are used to characterize the burstiness in the arrival pattern of events or data items. If $R(t)$ denotes the total number of events that arrived till at time t, then $\alpha^l(\Delta)$ and $\alpha^u(\Delta)$ denote the minimum and maximum number of events that arrived over *any* time interval of length Δ, i.e. $\alpha^l(t-s) \leq R(t) - R(s) \leq \alpha^u(t-s)$ for all $t > s$. We also denote $\alpha = (\alpha^l, \alpha^u)$.

Service curves are used to characterize the variability in the service provided by a hardware resource due to the variability of the processing requirements of eventa or data items. If $C(t)$ denotes the total number of events that could have been processed until time t, then $\beta^l(\Delta)$ and $\beta^u(\Delta)$ denote the minimum and maximum number of events that can be processed on a hardware resource within *any* time interval of length Δ, i.e. $\beta^l(t-s) \leq C(t) - C(s) \leq \beta^u(t-s)$ for all $t > s$. We also denote $\beta = (\beta^l, \beta^u)$.

In Figure 3(b), an abstract event stream $\alpha(\Delta)$ enters the abstract component and is processed using an abstract hardware resource $\beta^l(\Delta)$. The output is again an abstract event stream $\alpha'(\Delta)$, and the remaining resources are expressed again as an abstract hardware resource $\beta^{l'}(\Delta)$. Later on, we will also use the concept of VCCs in order to specify properties of reading out a playout buffer (readout VCC).

2) Abstract Component Relations: Consider a HW/SW component as depicted in Figure 3(a) that is triggered by the events of an incoming event stream. A fully preemptable task is instantiated at every event arrival to process the incoming event, and active tasks are processed in a greedy fashion in FIFO order, while being restricted by the availability of resources. At completion of a task processing, the corresponding event is removed from the input buffer and an event is emitted on the outgoing event stream. Following the results from Network Calculus [3] and Real-Time Calculus [5], such a component can be modeled by an abstract component *GP* as depicted in Figure 3(b) and in Figure 2 with the following internal component relations[1]:

$$\alpha^{u'} = \alpha^u \oslash \beta^l \tag{2}$$

$$\alpha^{l'} = \alpha^l \otimes \beta^l \tag{3}$$

$$\beta^{l'}(\Delta) = \sup_{0 \leq \lambda \leq \Delta} \left\{ \beta^l(\lambda) - \alpha^u(\lambda) \right\} \tag{4}$$

And moreover, the backlog of the input FIFO buffer can be bounded as $b(t) \leq \sup_{0 \leq \lambda \leq \Delta} \left\{ \alpha^u(\lambda) - \beta^l(\lambda) \right\}$ which leads to the following constraint

$$\alpha^u(\Delta) \leq \beta^l(\Delta) + b_{max} \quad \forall \Delta \geq 0 \tag{5}$$

[1]See the Appendix for a definition of $\otimes$ and $\oslash$

where b_{max} denotes the maximal available buffer space.

Now, let us derive the abstract component model *PB* of the playout buffer as used in the picture-in-picture application (see Figure 1) and shown in Figure 2. It receives data with an arrival curve $\alpha(\Delta)$ which are stored in a buffer with size B_{max}. We make the assumption that at time $t = 0$, there is already an B_0 number of data items in the playout buffer (due to the playback delay).

Data items in the playout buffer are removed by the video interface (see Figure 1). In particular, $D(t)$ data items are being removed in the time interval $[0, t)$. This behavior can be described by the readout VCC $\rho(\Delta) = (\rho^l(\Delta), \rho^u(\Delta))$, i.e. $\rho^l(t-s) \leq D(t) - D(s) \leq \rho^u(t-s)$ for all $t > s$. What needs to be guaranteed is that the playout buffer *PB* never overflows or underflows.

Assume that the total number of data items written to the playout buffer *PB* at time t is given by $R(t)$ and the total number of data items being read from *PB* at time t is given by $D(t)$ (readout stream). Then the number of data items $B(t)$ in the buffer at time t can be expressed as:

$$B(t) = R(t) - R(0) - [D(t) - D(0)] + B_0$$

Using the definitions of the associated VCCs we get the inequalities $B(t) \geq \alpha^l(\Delta) - \rho^u(\Delta) + B_0$ for all $\Delta \geq 0$ and $B(t) \leq \alpha^u(\Delta) - \rho^l(\Delta) + B_0$ for all $\Delta \geq 0$. In other words, in order for the *PB* not to overflow or underflow, its input arrival curve α and readout VCC ρ need to obey the following constraints

$$\alpha^l(\Delta) \geq \rho^u(\Delta) - B_0, \quad \forall \Delta \geq 0 \tag{6}$$

$$\alpha^u(\Delta) \leq \rho^l(\Delta) + B_{max} - B_0, \quad \forall \Delta \geq 0 \tag{7}$$

where B_{max} denotes the maximal available playout buffer space.

B. Interface Variables and Predicates

A real-time interface of an abstract component as introduced above has input and output variables related to event streams (rate variables α), resource availability (service variables β) and readout streams ρ. The output guarantee on a rate variable contains the bounds α^{lG} and α^{uG} which are part of the interface, and the predicate ϕ^O guarantees $\alpha^l(\Delta) \geq \alpha^{lG}$ and $\alpha^u(\Delta) \leq \alpha^{uG}$. The input assumption on the other hand contains the bounds α^{lA} and α^{uA}, and the predicate ϕ^I reflects the assumption $\alpha^l(\Delta) \geq \alpha^{lA}$ and $\alpha^u(\Delta) \leq \alpha^{uA}$. Similar relations hold for the service variables and readout variables.

In order to determine whether two abstract components are compatible we can check whether their interfaces are compatible. For this we need to check the relation $\phi^O \Rightarrow \phi^I$ for all connections. Arrival and readout connections are therefore compatible if

$$(\alpha^{lA}(\Delta) \leq \alpha^{lG}(\Delta)) \wedge (\alpha^{uG}(\Delta) \leq \alpha^{uA}(\Delta)) \; \forall \Delta \geq 0 \tag{8}$$

$$(\rho^{lA}(\Delta) \leq \rho^{lG}(\Delta)) \wedge (\rho^{uG}(\Delta) \leq \rho^{uA}(\Delta)) \; \forall \Delta \geq 0 \tag{9}$$

while a service connection is compatible if

$$\beta^{lA}(\Delta) \leq \beta^{lG}(\Delta) \qquad \forall \; \Delta \geq 0 \tag{10}$$

We can then generalize that two interfaces are compatible if (8),(9) and (10) are true for all internal rate, readout and service connections respectively, and if the input predicates of all open input variables are still satisfiable. Relation (8) is depicted in Figure 4.

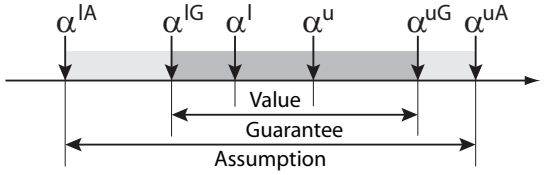

Fig. 4. Relation between interface assumptions, interface guarantees, and variable values.

In the next section we develop the internal interface relations between the interface elements in a real-time component system for stream processing.

V. Rate Interfaces

In this section, we are going to illustrate the concept of rate interfaces using a simple system consisting of one processing element (PE) containing one abstract component and one playout buffer (PB) as shown in Figure 5. Because of the modularity of our method, results derived for it can be reused for building more complex systems.

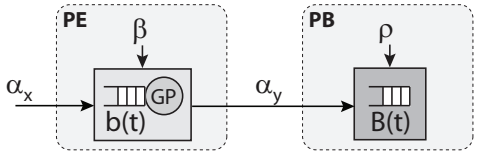

Fig. 5. Abstract components of a system with one processing element and one playout buffer.

For this simple system, there is one input stream going into the processing element described by the arrival curve $\alpha_x = (\alpha_x^l, \alpha_x^u)$. The processing element provides a service described by the lower service curve β^l. The processed output stream is described by the arrival curve $\alpha_y = (\alpha_y^l, \alpha_y^u)$. Note that the input stream α_x could be a stream coming from the environment or it could have been processed by other processing elements. The processed stream α_y is the input for the playout buffer. Data items from the *PB* are read at a rate described by the readout VCC $\rho = (\rho^l, \rho^u)$. It specifies the minimum and maximum number of data items being read from the *PB* over time intervals of any specified length.

For the system shown in Figure 5, we are going to show the conditions under which the buffer b associated with the component in the PE never overflows, and the buffer B does not overflow or underflow.

A. Rate Interface Algebra

Figure 6 represents the abstract components of Figure 5 in terms of their interfaces. As discussed in Section IV-B, for each rate and service variable in the system, the output guarantees and the input assumptions are shown.

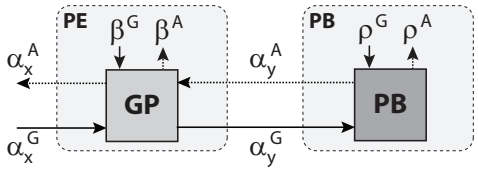

Fig. 6. Interface description of the abstract components shown in Figure 5.

We now need to develop the relations between guarantees and assumptions in order to be able to verify that predicates (8), (9) and (10) are satisfied when composing two or more interfaces.

1) Relations in a Processing Element: Using the relation between interface values, assumptions and guarantees (see Figure 4), one can deduce that the equations describing the output guarantees are equivalent to those for the abstract component, i.e. (2) and (3), just using interface guarantees instead of values. Therefore, we have

$$\alpha_y^{lG} = \alpha_x^{lG} \otimes \beta^{lG} \tag{11}$$
$$\alpha_y^{uG} = \alpha_x^{uG} \oslash \beta^{lG} \tag{12}$$

In order to calculate the input assumptions of the abstract component in the processing element, we need to determine pseudo-inverse relations corresponding to (2), (3) and the constraint (5). In particular, we have to find the largest upper curve and smallest lower curve such that the respective relations still hold. This approach leads to the weakest possible input assumption. In order to guarantee that all relations hold if the input and output predicates are satisfied, we have then to take the minimum (in case of the upper curves) or maximum (in case of the lower curves) of all these pseudo-inverses.

From the pseudo-inverse of equation (3), we get the inequalities $\alpha_x^{lA} \geq \alpha_y^{lA} \oslash \beta^{lG}$ and $\beta^{lA} \geq \alpha_y^{lA} \oslash \alpha_x^{lG}$. Here we use the duality relation between the $\oslash$ and $\otimes$ operators, see the Appendix. From the pseudo-inverse of equation (2), we get the inequalities $\beta^{lA} \geq \alpha_x^{uG} \oslash \alpha_y^{uA}$ and $\alpha_x^{uA} \leq \beta^{lG} \otimes \alpha_y^{uA}$. Inverting the buffer overflow constraint (5) is trivial and we get the inequalities $\alpha_x^{uA} \leq \beta^{lG} + b$ and $\beta^{lA} \geq \alpha_x^{uG} - b_{max}$.

In summary after combining the previous inequalities, the assumptions of a processing element can be determined as follows:

$$\alpha_x^{lA} = \alpha_y^{lA} \oslash \beta^{lG} \tag{13}$$
$$\alpha_x^{uA} = \min\{\beta^{lG} \otimes \alpha_y^{uA}, \beta^{lG} + b_{\max}\} \tag{14}$$
$$\beta^{lA} = \max\{\alpha_y^{lA} \oslash \alpha_x^{lG}, \alpha_x^{uG} \oslash \alpha_y^{uA}, \alpha_x^{uG} - b_{\max}\} \tag{15}$$

2) Relations in a Playout Buffer: For a playout buffer *PB*, the relations are somewhat simpler. We just need to determine the inverse relations for the buffer constraints (6), (7) which yields the following relations:

$$\alpha_y^{uA} = \rho^{lG} + B_{\max} - B_0 \tag{16}$$
$$\alpha_y^{lA} = \rho^{uG} - B_0 \tag{17}$$
$$\rho^{uA} = \alpha_y^{lG} + B_0 \tag{18}$$
$$\rho^{lA} = \alpha_y^{uG} - (B_{\max} - B_0) \tag{19}$$

We could now also combine the above interface relations if we want to construct a *single* interface that describes the composed system $PE \| PB$ in Figure 6. The new interface states the input assumptions of the system only in terms of the output guarantees that the system receives from its environment:

$$\alpha_x^{lA} = (\rho^{uG} - B_0) \oslash \beta^{lG} \quad (20)$$

$$\alpha_x^{uA} = \min\{\beta^{lG} \otimes (\rho^{lG} + B_{max} - B_0), \beta^{lG} + b_{max}\} \quad (21)$$

$$\beta^{lA} = \max\{(\rho^{uG} - B_0) \oslash \alpha_x^{lG}, \alpha_x^{uG} \oslash (\rho^{lG} + B_{max} - B_0), \alpha_x^{uG} - b_{max}\} \quad (22)$$

$$\rho^{lA} = (\alpha_x^{uG} \oslash \beta^{lG}) - (B_{max} - B_0) \quad (23)$$

$$\rho^{uA} = (\alpha_x^{lG} \otimes \beta^{lG}) + B_0 \quad (24)$$

3) Composing Systems with Fixed Priority Scheduling: The system shown in Figure 6 can be composed with other processing elements, for example having more than one task that needs to be executed for a single data item, see the scenario depicted in Figure 1 and the associated component model in Figure 2.

It is also possible to have systems with processing elements that process more than a single data stream. This is actually the case for the application shown in Figure 1 where in PE_2 there are two components which share the service provided by PE_2 using fixed priority scheduling. The corresponding abstract components and interface connections are depicted in Figure 2.

If a component shares the service that it receives from a processing element with another lower-priority component, this remaining service is bounded by (4). In terms of output guaranteed values, this can be expressed as:

$$\beta'^{lG}(\Delta) = \sup_{0 \le \lambda \le \Delta} \{\beta^{lG}(\lambda) - \alpha_x^{uG}(\lambda)\} \stackrel{def}{=} RT(\beta^{lG}, \alpha_x^{uG})$$

In order to find the input assumptions of a component using fixed priority scheduling, we will need to use the inverse of the RT operator. It is defined in the Appendix, for more details please refer to [16]. Most of the relations for the input assumptions are the same as the ones for a PE with one component. However, equations (14) and (15) change because of taking into account the pseudo-inverse of (4):

$$\alpha_x^{uA} = \min\{\beta^{lG} \otimes \alpha_y^{uA}, \beta^{lG} + b, RT^{-\alpha}(\beta'^{lA}, \beta^{lG})\}$$

$$\beta^{lA} = \max\{\alpha_y^{lA} \oslash \alpha_x^{lG}, \alpha_x^{uG} \oslash \alpha_y^{uA}, \alpha_x^{uG} - b, RT^{-\beta}(\beta'^{lA}, \alpha_x^{uG})\}$$

VI. Generalizing the Interface Model

The theory of real-time interfaces is not confined to the interface models presented in the previous sections, but can be generalized into various directions. Following we only show some of the possible generalizations.

A. Large Interface Models

In Section V we derived the interface relations for a PE with one abstract component in (11)–(15), as well as for a PB in (16)–(19). In (20)–(24) we combine these relations to the interface of a system consisting of a single PE and a single PB, as depicted in Figure 6. Analogously one could combine the interfaces of several single PE's and PB's into larger systems that are composed of many PE's and PB's, as for example depicted in the bottom of Figure 2. By concatenating the relations of the single component interfaces we then directly obtain the interface of the larger system.

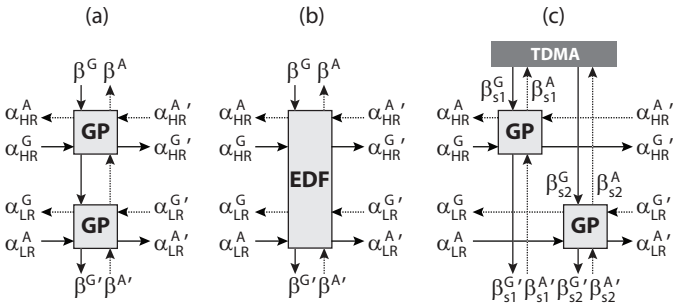

Fig. 7. Interface models for (a) fixed priority, (b) EDF and (c) TDMA scheduling.

B. Scheduling Policies

As mentioned at then end of Section V, the theory of real-time interfaces also allows to express the interface of a PE that processes more than one task, i.e. that contains more than one abstract component. In Section V we presented the internal interface relations for abstract components that share the available service using a fixed priority scheduling policy. However, it is also possible to derive the internal interface relations for various other scheduling policies, allowing to obtain interface models of PE's that implement any of these scheduling policies. For example, [17] derives relations for EDF scheduling and hierarchical scheduling with polling servers, while [18] on the other hand focuses on the interface relations for TDMA scheduling. Some interface models for PE's with different scheduling policies are depicted in Figure 7.

C. Rich Interfaces

In this work, the interface of an abstract component exposes the rate and service guarantees and the requirements it has, such as its internal buffer does not overflow. The interface of a playout buffer component on the other hand exposes the rate and consumption guarantees and assumptions to ensure that its internal buffer neither overflows nor underflows. Hence, the buffers of both components are treated as fixed internal parts and are not exposed at the interfaces.

However, we could also expose the internal buffers of abstract components at their interfaces. Analogous to (11)–(15) and (16)–(19), we could then derive the interface relations for the buffer guarantees and assumptions, using the various buffer constraints described in Section V. With such richer interfaces, one could solve additional design problems. For example, given the service guarantees and assumptions as well as the rate guarantees and assumptions of an abstract

component, what is the minimum assumption on its internal buffer such that the buffer does not overflow? Moreover, if we would also expose the initial buffer fill level in the interface of a playout buffer, one could even derive the assumptions on the minimum and maximum initial buffer fill level such that the playout buffer does not overflow or underflow, given its rate and consumption guarantees.

Taking this concept one step further, we might even expose additional features on a component's interface. For real-time event streams that are processed on a PE with EDF scheduling it is, for example, interesting to determine the deadlines that can be associated with the event streams such that all streams remain schedulable. By exposing the deadlines at the component interface, we could easily derive the assumptions on the minimum deadline that must be associated to an event stream such that the system remains schedulable.

VII. Case Study

In this section we show how our proposed theory can be applied to the application scenario described in Section II. Towards this, each PE and playout buffer in Figure 1 is considered to be an independent component and our objective is to connect them together to realize the architecture shown in the figure. In order to decide whether two components can be connected together, we would only inspect their interfaces. Two compatible interfaces implicitly guarantee that the buffers inside their respective components will never overflow, and in addition, the playout buffers will never underflow.

The main message in this section is an illustration of how the internal details of a component (e.g. its buffer size, scheduling policy, processor frequency) are reflected (or summarized) through its rate interface. We show that if these internal details are changed then the component's rate interface also gets changed and two previously compatible components may now become incompatible (or vice versa). As mentioned earlier, this approach is in contrast to first designing/assembling the complete system and postponing the performance analysis step to the very end.

Figure 2 shows the component model of the architecture in Figure 1 and also its interface model. Whereas the arrows in the component model (top) represent the flow of the video streams, those in the interface model (bottom) represent the assumptions and the guarantees associated with the individual components.

Experimental setup: We used two different sets of video clips; the first set being made up of regular clips with moderate to high motion content and the second set being made up of clips displaying still images. The former set characterizes video streams to be viewed through the main window of the PiP application. The latter set represents content like stock quotes and upcoming program schedules, which will be viewed in the smaller secondary window (see Figure 1). These two sets characterize the two streams HR and LR shown in Figure 1. In our experiments, both the incoming streams have the same frame resolution of 704×576 pixels and we assume that the down-scaling for the secondary window is being done at the output device. However, for simplicity we slightly abuse the notation and refer to the stream for the main window as the high-resolution (or HR) stream and that for the secondary window as the low-resolution (or LR) stream. Both the streams arrive at the system at a constant bitrate of 8 Mbps and the playout buffers are read at a constant rate of 25 frames/second. Further, in the architecture shown in Figure 1, PE_1 is set to run at a processor frequency of 1.3 GHz, PE_2 at 3 GHz and PE_3 at 1.25 GHz. The buffer sizes b_2 and b_4 inside PE_2 are set to 1 and 2 video frames respectively (each frame being made up of 1584 macroblocks).

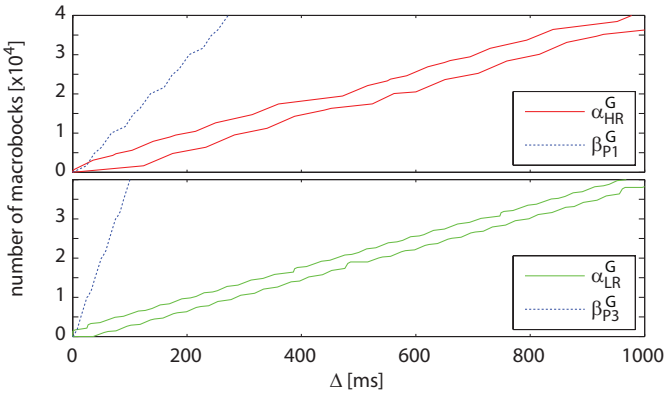

Fig. 8. Upper and lower bounds on the arrival process of the video streams at PE_1 and PE_3, and the guaranteed service offered by these PEs.

We simulated the execution of the VLD, IQ, IDCT and MC tasks for these two sets of video clips using a SimpleScalar model [1] of the PEs (with the *sim-profile* configuration and the PISA instruction set). Both the video streams were modeled at the macroblock granularity. Note that the number of bits constituting each compressed macroblock in the input stream is variable. Hence, at the macroblock granularity, a constant bitrate stream translates into a bursty arrival pattern. Similarly, the number of processor cycles required to process each macroblock is also highly variable. Hence, the service offered by a PE running at a constant frequency translates into a variable service when expressed in terms of the number of macroblocks guaranteed to be processed within any time interval of a specified length. Figure 8 shows the bounds on these bursty arrival patterns at the input to the system for the HR and LR streams. It also shows the guaranteed service β^G offered by PE_1 and PE_3 to these two streams.

Results: We report three different cases, each of them involving different instances of the five components shown in Figure 2. These instances were obtained by changing the sizes and initial fill-levels of the playout buffer components, with all the other components remaining unchanged. We show that in the first case, the components turn out to be compatible, whereas in the following two cases they are incompatible. Our main result is that this compatibility is checked by only inspecting the components' interfaces.

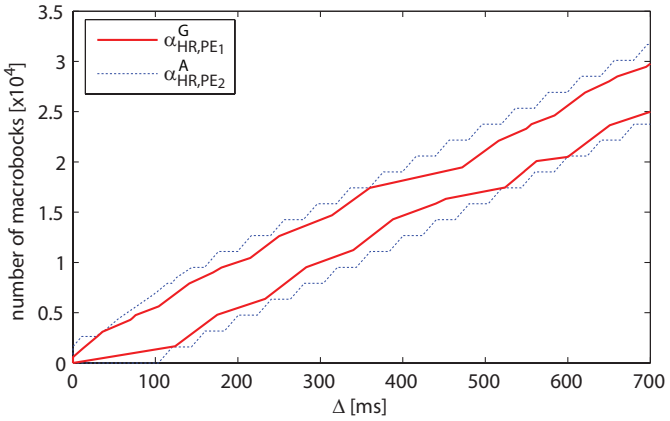

Fig. 9. Assumption on the input rate expressed by PE_2's interface matches the guarantee on the output rate provided by PE_1's interface. Hence, PE_1 is compatible with the partially designed system.

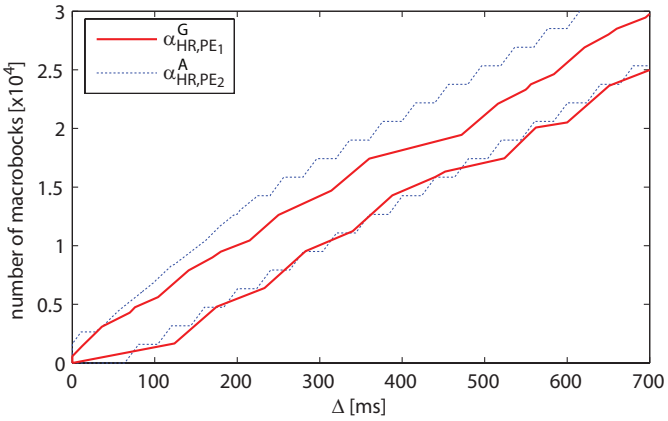

Fig. 10. Assumption on the input rate expressed by PE_2's interface does not match the guarantee on the output rate provided by PE_1's interface. Hence, PE_1 is *not* compatible with the partially designed system.

Before continuing further, we would like to point out that the buffer sizes b_2 and b_4 in PE_2 (see Figure 1) can be chosen independently of the other system parameters. However, they influence the rates of the incoming streams *assumed* by the component PE_2. On the other hand, for given arrival rates of the input streams and given processor frequencies, the buffer sizes b_1 and b_3 inside PE_1 and PE_3 should be of a certain minimum size for them not to overflow. Given the parameters in our experimental setup, these buffer sizes turn out to be $b_1 = 576$ macroblocks and $b_2 = 1315$ macroblocks.

In what follows, we will only be concerned with checking if the component PE_1 (in Figure 2) is compatible with a partially designed system with all the other components already connected together. This is done by checking if the output rate guaranteed by PE_1 is compatible with the input rate assumed by PE_2. This compatibility check is indicated in Figure 2 with a "?" mark.

Case I: Here we consider a partially designed system with all the components except PE_1 already connected. The buffer size in the component PB_{HR} is equal to 6 frames and its initial fill-level is set to 3 frames. In other words, the output device starts consuming frames from this component only after an initial playout delay, during which the buffer fill-level reaches 3 frames. The buffer size and initial fill-level associated with the component PB_{LR} are also 6 and 3 frames respectively. All the other components are as specified in our experimental setup.

The problem is now to check whether the component PE_1 is compatible with this partially designed system. Figure 9 shows the *assumption* on the input rate α^A_{HR,PE_2} expressed by PE_2's component interface. The same figure also shows the *guarantee* on the output stream rate α^G_{HR,PE_1} expressed by PE_1's interface. Here, the output guarantee is fully "enclosed" by the input assumption (i.e. they match), thereby suggesting that the two components are compatible.

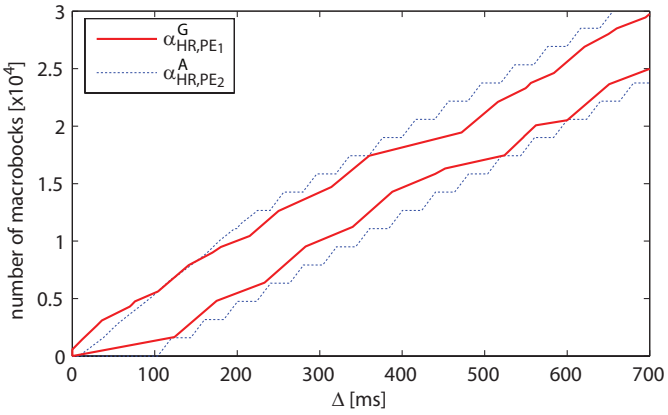

Fig. 11. The assumptions and guarantees do not match; hence, PE_1 is *not* compatible with the partially designed system. Here, the incompatibility is due to the component PB_{LR} being changed.

Case II: Now we consider the same partially designed system but with the component PB_{HR}'s initial buffer fill-level reduced to 2 frames and everything else remaining unchanged. The assumptions and the guarantees on the input and output rates in this case are shown in Figure 10. Note that since the component PE_1 remains unchanged, its output guarantee α^G_{HR,PE_1} is also unchanged. However, the input assumption from the partially designed system has changed and it does not satisfy the guarantee provided by PE_1 anymore. Hence, the two components are no longer compatible. If they are connected together, then the playout buffer in PB_{HR} might underflow.

Case III: Here, the partially designed system is the same as in Case I, with the only exception being the component PB_{LR}, whose buffer size and initial fill-level are changed to 5 and 3 frames respectively. Note that (surprisingly) changing the size of the playout buffer associated with the low resolution stream has an influence on the input assumption expressed at the interface of PE_2 that is associated with the high resolution stream! This nicely illustrates how changes made

in one component are reflected in a different component's interface once they have been composed together. Figure 11 shows the assumptions and the guarantees on the input and output rates in this case. Again, here they do not match. If PE_1 is connected to the system then this might potentially lead to an overflow of the buffer inside PB_{LR}.

In summary, we have shown through a concrete example how incremental compatibility checking can be done using rate interfaces. Clearly, such interfaces can also be used in a straightforward manner for resource dimensioning and component-level design space exploration. Here, the typical questions that one would ask are: What is the minimum buffer size of a component such that its interface is compatible with a partially existing design? Or what is the minimum processing frequency such that its interface is still compatible? Finally, although here we were only concerned with buffer overflow and underflow constraints, as mentioned before, our framework can be easily extended to handle other performance constraints as well.

VIII. Concluding Remarks

In this paper we proposed a theory of rate interfaces for compositional design of embedded systems whose components communicate via data streams. We showed how rate interfaces can be effectively checked to guarantee buffer overflow and underflow constraints. As mentioned earlier, the framework can be extended to handle other types of constraints as well.

This paper extends our previous work in two different ways. First, we cast the rate analysis problem studied in [9], [11] in an interface-theoretic setting and show how this leads to efficient component-based design of embedded systems. Second, it also extends our previous work on real-time interfaces [16], [17] by applying them to a distributed setup. Real-time interfaces were proposed by us for the design and analyze real-time systems in an interface-theoretic setting. However, the results in [16], [17] were restricted to uniprocessor systems. In this paper we show that by using the rate interface algebra, real-time interfaces can be extended to analyze multiprocessor architectures.

The algebra we presented in this paper is purely stateless. It would be interesting to extend our work to setups where the processing in a component is state-dependent. Towards this, stateful interface languages like *interface automata* [7] will be worth investigating.

Appendix: Min-Max Algebra

The operators $\otimes$ and $\oslash$ are known as min-plus convolution and min-plus deconvolution respectively. Details of their mathematical properties may be found in [3]. For the purpose of this paper, we only need the following definitions and results.

The min-plus convolution and deconvolution operators are defined as:

$$\begin{aligned}(f \otimes g)(\Delta) &= \inf_{0 \le \lambda \le \Delta} \{f(\Delta - \lambda) + g(\lambda)\} \\ (f \oslash g)(\Delta) &= \sup_{\lambda \ge 0} \{f(\Delta + \lambda) - g(\lambda)\}\end{aligned}$$

The duality between $\otimes$ and $\oslash$ states that:
$f \oslash g \le h \iff f \le g \otimes h$ (see [3] for proof).

The operator $RT(\beta, \alpha)$ is defined as:

$$\beta'(\Delta) = \sup_{0 \le \lambda \le \Delta} \{\beta(\lambda) - \alpha(\lambda)\}$$

Its two inverses are defined as:

$$\begin{aligned}RT^{-\alpha}(\beta', \beta)(\Delta) &= \beta(\Delta + \lambda) - \beta'(\Delta + \lambda) \\ &\quad \text{for} \quad \lambda = \sup\{\tau \ : \ \beta'(\Delta + \tau) = \beta'(\Delta)\} \\ RT^{-\beta}(\beta', \alpha)(\Delta) &= \beta'(\Delta - \lambda) + \alpha(\Delta - \lambda) \\ &\quad \text{for} \quad \lambda = \sup\{\tau \ : \ \beta'(\Delta - \tau) = \beta'(\Delta)\}\end{aligned}$$

References

[1] T. Austin, E. Larson, and D. Ernst. SimpleScalar: An infrastructure for computer system modeling. *IEEE Computer*, 35(2):59–67, 2002.

[2] E. Bini and M. Di Natale. Optimal task rate selection in fixed priority systems. In *IEEE Real-time Systems Symposium (RTSS)*, 2005.

[3] J.-Y. Le Boudec and P. Thiran. *Network Calculus*. LNCS 2050, Springer Verlag, 2001.

[4] A. Chakrabarti, L. de Alfaro, T. A. Henzinger, and M. Stoelinga. Resource interfaces. In *International Conference on Embedded Software (EMSOFT)*, LNCS 2855, 2003.

[5] S. Chakraborty, S. Künzli, and L. Thiele. A general framework for analysing system properties in platform-based embedded system designs. In *Design, Automation and Test in Europe (DATE)*, 2003.

[6] L. de Alfaro and T. A. Henzinger. Interface theories for component-based design. In *International Conference on Embedded Software (EMSOFT)*, 2001.

[7] L. de Alfaro and T. A. Henzinger. Interface-based design. In *Engineering Theories of Software Intensive Systems, Proceedings of the Marktoberdorf Summer School*. Kluwer, 2004.

[8] L. de Alfaro, T. A. Henzinger, and M. Stoelinga. Timed interfaces. In *International Conference on Embedded Software (EMSOFT)*, LNCS 2431, 2002.

[9] Y. Liu, S. Chakraborty, and R. Marculescu. Generalized rate analysis for media-processing platforms. In *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA) (to appear)*, 2006.

[10] A. Mathur, A. Dasdan, and R. K. Gupta. Rate analysis for embedded systems. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 3(3):408–436, 1998.

[11] A. Maxiaguine, S. Künzli, S. Chakraborty, and L. Thiele. Rate analysis for streaming applications with on-chip buffer constraints. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2004.

[12] A. Maxiaguine, Y. Zhu, S. Chakraborty, and W.-F. Wong. Tuning SoC platforms for multimedia processing: Identifying limits and tradeoffs. In *International conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2004.

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

[14] M.J. Rutten, J.T.J. van Eijndhoven, and E.-J.D. Pol. Robust media processing in a flexible and cost-effective network of multi-tasking coprocessors. In *Euromicro Conference on Real-Time Systems (ECRTS)*, 2002.

[15] G. Varatkar and R. Marculescu. On-chip traffic modeling and synthesis for MPEG-2 video applications. *IEEE Transactions on VLSI*, 12(1), January 2004.

[16] E. Wandeler and L. Thiele. Real-time interfaces for interface-based design of real-time systems with fixed priority scheduling. In *International Conference on Embedded Software (EMSOFT)*, 2005.

[17] E. Wandeler and L. Thiele. Interface-based design of real-time systems with hierarchical scheduling. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2006.

[18] E. Wandeler and L. Thiele. Optimal tdma time slot and cycle length allocation for hard real-time systems. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2006.