

# An Interface Algebra for Estimating Worst-Case Traversal Times in Component Networks<sup>\*</sup>

Nikolay Stoimenov<sup>1</sup>, Samarjit Chakraborty<sup>2</sup>, and Lothar Thiele<sup>1</sup>

<sup>1</sup> Computer Engineering and Networks Laboratory, ETH Zurich, Switzerland  
{stoimenov, thiele}@tik.ee.ethz.ch

<sup>2</sup> Institute for Real-Time Computer Systems, TU Munich, Germany  
samarjit@tum.de

**Abstract.** Interface-based design relies on the idea that different components of a system may be developed independently and a system designer can connect them together only if their interfaces match, without knowing the details of their internals. In this paper we propose an interface algebra for analyzing networks of embedded systems components. The goal is to be able to compute worst-case traversal times and verify their compliance to provided deadlines in such component networks in an incremental manner, i.e., as and when new components are added or removed from the network. We lay the basic groundwork for this algebra and show its utility through an illustrative example.

## 1 Introduction

Today most embedded systems consist of a collection of computation and communication components that are supplied by different vendors and assembled by a system manufacturer. Such a component-based design methodology is followed in several domains such as automotive, avionics, and consumer electronics. The system manufacturer responsible for component assembly has to take design decisions (related to the choice of components and how they are to be connected, e.g., using a bus or a network-on-chip) and perform system analysis. One such important analysis is related to the computation of end-to-end delays or worst-case traversal times (WCTTs) of data through the component network.

To carry out such an analysis, it is important to model the internals of each component, which is often difficult because of the complexity of the components and their proprietary nature. To get around this problem, an *interface*-based design methodology may be appropriate. Here, components may be connected if and only if their interfaces *match*. Further, the timing behavior of the network may be computed in an incremental manner – i.e., as and when new components are added or existing ones are modified – without always having to analyze the entire system from scratch. In this paper, we develop an *interface algebra* for estimating the worst-case traversal times and verify their compliance to provided upper bounds, where the different components exchange data through first-in first-out (FIFO) buffers. Such architectures are common for streaming applications, e.g., audio/video processing and distributed controllers where data flows

---

<sup>\*</sup> The work is partially supported by NCCR-MICS, a center supported by the Swiss National Science Foundation under grant number 5005-67322, and by the DFG through the SFB/TR28 “Cognitive Automobiles”.

from sensors to actuators while getting processed on multiple processors. Our proposed interface algebra is an extension of [4, 8] and is motivated by the concept of *assume/guarantee* interfaces from [1]. In particular, we cast the *Real-Time Calculus* framework from [3] within the *assume/guarantee* interface setting. At a high level, two interfaces *match* when the guarantees associated with one of them comply with the assumptions associated with the other. Our main technical result is Theorem 1 which describes how to compute the *assumptions* and *guarantees* associated with components based on a *monotonicity* property. This result is then used to compute the WCTT in a component network incrementally, and validate that it complies to a given deadline. Our approach may be summarized in the following three steps:

1. Define an abstract component that describes the real-time properties of a concrete system component. This involves defining proper abstractions for component inputs and outputs, and internal component relations that meaningfully relate abstract inputs to abstract outputs.
2. To derive the interface of an abstract component, we need to define interface variables as well as input and output predicates on these interface variables.
3. Derive the internal interface relations that relate incoming guarantees and assumptions to outgoing guarantees and assumptions of a component's interfaces.

In the setting we study here, event streams are processed on a sequence of components. An event or data stream described by the cumulative function  $R(t)$  enters the input buffer of the component and is processed by the component whose availability is described by the cumulative function  $C(t)$ . Formally, the cumulative functions  $R(t) \in \mathbb{R}^{\geq 0}$  and  $C(t) \in \mathbb{R}^{\geq 0}$  for  $t \geq 0$  denote the number of events/data items that have been received or could be processed within the time interval  $[0, t)$ , respectively. After being processed, events are emitted on the component's output, resulting in an outgoing event stream  $R'(t)$ . The remaining resources that were not consumed are available for use 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. For example, Greedy Processing (GP) denotes that events are always processed when there are resources available. 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.

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* (VCCs) capture all possible traces using upper and lower bounds on their timing properties. The *arrival* and *service curves* from Network Calculus [5] are specific instances of VCCs and are more expressive than traditional event stream models such the periodic, periodic with jitter, sporadic, etc. Arrival curves  $\alpha^l(\Delta)$  and  $\alpha^u(\Delta)$  denote the minimum and the maximum number of events that can arrive in *any* time interval of length  $\Delta$ , i.e.,  $\alpha^l(t-s) \leq R(t) - R(s) \leq \alpha^u(t-s)$  for all  $t > s \geq 0$ . In addition,  $\alpha^l(0) = \alpha^u(0) = 0$ . We also denote the tuple  $(\alpha^l, \alpha^u)$  with  $\alpha$ . Service curves characterize the variability in the service provided by a

resource. The curves  $\beta^l(\Delta)$  and  $\beta^u(\Delta)$  denote the minimum and the maximum number of events that can be processed within *any* time interval of length  $\Delta$ , i.e.,  $\beta^l(t-s) \leq C(t) - C(s) \leq \beta^u(t-s)$  for all  $t > s \geq 0$ . In addition,  $\beta^l(0) = \beta^u(0) = 0$ . We also denote the tuple  $(\beta^l, \beta^u)$  with  $\beta$ . An event stream modeled by  $\alpha(\Delta)$  enters a component and is processed using the resource modeled as  $\beta^l(\Delta)$ . The output is again an event stream  $\alpha'(\Delta)$ , and the *remaining* resource is expressed as  $\beta'^l(\Delta)$ . Note that the domain of the arrival and service curves are events, i.e., they describe the number of arriving events and the capability to process a certain number of events, respectively. The generalization towards physical quantities such as processing cycles or communication bits can be done by means of *workload curves* which is another instance of a VCC (see [6]).

## 2 Timing Analysis of Component Networks

In this section, we describe a timing analysis framework (in particular, for computing worst-case traversal times) for component networks that is based on Real-Time Calculus [3, 7]. This calculus is an adaptation of Network Calculus [5] that was designed to analyze communication networks. Here, we consider three basic types of abstract components: Processing Element (PE), Playout Buffer (PB), and Earliest Deadline First (EDF) component. For the component models of Greedy Shapers, Time Division Multiple Access, servers, and hierarchical scheduling, please refer to [9–11]. In Sect.3 we lift this framework to an interface-theoretic setting and present our main technical result.

### 2.1 Processing Element

The PE component can be used to model a single processing element which processes one input stream. However, it can also be composed with other components of the same type, and model components processing more than one input stream using a fixed priority (FP) scheduling. Consider a concrete GP component that is triggered by the events of an incoming event stream. A fully preemptive task is instantiated at every event arrival to process the incoming event, and active tasks are processed in a FIFO order, while being restricted by the availability of resources. The completion of each task execution results in the corresponding event being removed from the input buffer and an event being emitted on the outgoing event stream.

Following results from Real-Time and Network Calculus [3, 5], such a component can be modeled by an abstract component  $PE$  with the following internal component relations<sup>3</sup>:

$$\alpha'^u = \alpha^u \otimes \beta^l, \quad \alpha'^l = \alpha^l \otimes \beta^l, \quad (1)$$

$$\beta'^l(\Delta) = \sup_{0 \leq \lambda \leq \Delta} \{ \beta^l(\lambda) - \alpha^u(\lambda) \} := RT(\beta^l, \alpha^u), \quad (2)$$

---

<sup>3</sup> See the Appendix for definitions of  $\otimes$  and  $\circledast$ .

and the backlog of the input buffer is bounded by  $\sup_{0 \leq \lambda \leq \Delta} \{\alpha^u(\lambda) - \beta^l(\lambda)\}$ . If the available buffer space in the input buffer is constrained by  $b_{\max}$ , the backlog should never become bigger than the buffer size, i.e., we have a *buffer overflow constraint*. In this case, we can obtain the following component-based constraint on the admissible arrival and service curves:

$$\alpha^u(\Delta) \leq \beta^l(\Delta) + b_{\max}, \quad \forall \Delta \in \mathbb{R}^{\geq 0} . \quad (3)$$

If the input arrival and service curves satisfy the above constraint, the backlog will never be bigger than  $b_{\max}$ . Before continuing with the computation of the WCTT for a PE component, we need to define the following shift function:

$$r(\alpha, c, \Delta) = \begin{cases} \alpha(\Delta - c) & \text{if } (\Delta > c) \wedge (\Delta \neq 0) \\ 0 & \text{if } (\Delta \leq c) \vee (\Delta = 0) \end{cases} \quad (4)$$

which simply shifts a given curve  $\alpha(\Delta)$  by the amount  $c$  to 'the right'. The WCTT experienced by an event in the component, defined as its finishing time minus its arrival time, can be computed as:

$$\text{Del}(\alpha^u, \beta^l) := \sup_{\lambda \geq 0} \left\{ \inf \{ \tau \geq 0 : \alpha^u(\lambda) \leq \beta^l(\lambda + \tau) \} \right\} .$$

It is also possible to have systems with processing elements that process more than one data stream. For this purpose, the remaining service output of a higher priority PE component, computed with (2), can be connected to the service input of a lower priority PE component. This way we can model an FP resource sharing between PE components.

## 2.2 Playout Buffer

The PB component models a playout buffer. It receives data and stores it in a buffer which is read at a constant (usually periodic) rate. The buffer has a maximum size  $B_{\max}$ . We make the assumption that at the start of the system, there are already  $B_0$  initial data items in the playout buffer, e.g., due to a playback delay. Data items in the playout buffer are removed at a constant rate. In particular,  $P(t)$  data items are removed within the time interval  $[0, t)$ . This behavior can be described by the readout VCC  $\rho(\Delta) = (\rho^l(\Delta), \rho^u(\Delta))$ , i.e.,  $\rho^l(t - s) \leq P(t) - P(s) \leq \rho^u(t - s)$  for all  $t > s \geq 0$ . What needs to be guaranteed is that the playout buffer never *overflows* or *underflows*. Following results on Real-Time and Rate Interfaces [4], for a PB component with input and readout event streams characterized by the VCCs  $\alpha$  and  $\rho$ , respectively, and  $B_0$  initial events, the playout buffer size  $B(t)$  is constrained by  $0 \leq B(t) \leq B_{\max}$  at all times if

$$\alpha^l(\Delta) \geq \rho^u(\Delta) - B_0, \text{ and } \alpha^u(\Delta) \leq \rho^l(\Delta) + B_{\max} - B_0, \quad \forall \Delta \in \mathbb{R}^{\geq 0} . \quad (5)$$

The WCTT experienced by an event in the component can be computed as  $\text{Del}(\alpha^u, \rho_\tau^l)$  where

$$\rho_\tau^l(\Delta) = r(\rho^l, \tau, \Delta) \quad (6)$$

is the lower readout curve 'shifted to the right' by the initial playback delay  $\tau \geq 0$  necessary to accumulate  $B_0$  events.

### 2.3 Earliest Deadline First Component

The EDF component is similar to PE but it models processing of several data streams with a resource shared using the earliest deadline first scheduling policy. This requires a new abstract component with different internal relations [10]. Such a component processes  $N$  input event streams and emits  $N$  output event streams. Each input event stream  $i$ ,  $1 \leq i \leq N$ , is associated with a fully preemptive task which is activated repeatedly by incoming events. Each input event stream  $i$  has an associated FIFO buffer with maximum size  $b_{i \max}$  where events are backlogged. Tasks process the head events in these buffers and are scheduled in an EDF order. Each task has a best-case execution time of  $\text{BCET}_i$ , a worst-case execution time  $\text{WCET}_i$ , and a relative deadline  $D_i$  where  $0 \leq \text{BCET}_i \leq \text{WCET}_i \leq D_i$ . The completion of a task execution results in the corresponding input event being removed from the associated buffer and an output event being emitted on the associated output event stream.

For an EDF component with a service curve  $\beta$  and event streams characterized by arrival curves  $\alpha_i$ , all tasks are processed within their deadlines if and only if

$$\sum_{i=1}^N r(\alpha_i^u, D_i, \Delta) \leq \beta^l(\Delta), \quad \forall \Delta \in \mathbb{R}^{\geq 0}, \quad (7)$$

using the shift function  $r$  from (4). The output streams can be characterized by arrival curves computed for all streams  $i$  as:

$$\alpha_i'^u(\Delta) = r(\alpha_i^u, -(D_i - \text{BCET}_i), \Delta), \quad \alpha_i'^l(\Delta) = r(\alpha_i^l, (D_i - \text{BCET}_i), \Delta), \quad (8)$$

and the number of events in input buffers do not exceed their capacity  $b_{i \max}$  if

$$\alpha_i^u(D_i) \leq b_{i \max}, \quad \forall i. \quad (9)$$

The EDF component schedulability condition (7) can be related to the demand bound functions described in [2]. Given that the condition is satisfied, the service curve provided to each stream can be modeled with a burst-delay function [5] defined for each stream  $i$  as:

$$\beta_{D_i}^l(\Delta) = \begin{cases} +\infty & \text{if } \Delta > D_i \\ 0 & \text{otherwise} \end{cases} \quad (10)$$

The WCTT experienced by an event from a stream can be computed as  $\text{Del}(\alpha_i^u, \beta_{D_i}^l)$  which is bounded by  $D_i$  for each stream.

### 2.4 Worst-Case Traversal Times of Component Networks

The worst-case traversal time of an event from an input stream which is processed by a sequence of components can be computed as the sum of the worst-case

traversal times of the individual components. However, this would lead to a very pessimistic and unrealistic result as it would assume that the worst-case traversal times occur in all components for the same event. A better bound on the worst-case traversal time can be achieved by considering a concatenation of the components. This is a phenomenon known as “pay bursts only once” [5]. Following results from Network Calculus, this leads to the following computation for the WCTT.

For an input event stream  $\alpha$  traversing a sequence of components which consists of a set of PEs, a set of PBs, and a set of EDF components denoted as  $\mathcal{PE}$ ,  $\mathcal{PB}$  and  $\mathcal{EDF}$ , respectively, the worst-case traversal time that an event can experience can be computed as  $\text{Del}(\alpha^u, \beta_{\mathcal{PE}} \otimes \rho_{\mathcal{PB}} \otimes \beta_{\mathcal{EDF}})$  with  $\beta_{\mathcal{PE}} = \bigotimes_{c \in \mathcal{PE}} \beta_c^l$ ,  $\rho_{\mathcal{PB}} = \bigotimes_{c \in \mathcal{PB}} \rho_{\tau_c}^l$ ,  $\beta_{\mathcal{EDF}} = \bigotimes_{c \in \mathcal{EDF}} \beta_{D_i, c}^l$ , and  $\beta_c^l$  is the service availability of PE component  $c$ ,  $\rho_{\tau_c}^l$  is the lower readout curve for PB component  $c$  as defined with (6), and  $\beta_{D_i, c}^l$  is the service availability provided to the stream served with relative deadline  $D_i$  by EDF component  $c$  as defined with (10). A WCTT constraint on the sequence of components  $\text{Del}(\alpha^u, \beta_{\mathcal{PE}} \otimes \rho_{\mathcal{PB}} \otimes \beta_{\mathcal{EDF}} \otimes \sigma_{\mathcal{GS}}) \leq D$  can be written as follows:

$$\beta_{\mathcal{PE}} \otimes \rho_{\mathcal{PB}} \otimes \beta_{\mathcal{EDF}} \geq r(\alpha^u, D, \Delta), \quad \forall \Delta \in \mathbb{R}^{\geq 0}, \quad (11)$$

using the shift function  $r$  from (4).

### 3 Interface Algebra

In this section, we develop an interface-based design approach which will allow us by only inspecting the interfaces of two components to check whether WCTT and buffer underflow/overflow constraints would be satisfied if the components are composed together. The proposed interface algebra includes concepts from Real-Time Calculus, Assume/Guarantee Interfaces [1], and constraints propagation.

In our setup each component has two disjoint sets of input and output ports  $I$  and  $O$ . The actual input and output values of an abstract component are VCC curves. A connection from output  $j$  of one component to the input  $i$  of some other component will be denoted by  $(j, i)$ . The interface of a component makes certain *assumptions* on  $I$ , which are specified using the predicate  $\phi^I(I)$ . Provided this predicate is satisfied, the interface *guarantees* that the component works correctly and its outputs will satisfy a predicate  $\phi^O(O)$ .

In order to simplify the presentation, we introduce the *complies to* relation  $\vdash$  between two VCC curves  $a(\Delta)$  and  $b(\Delta)$  as follows:

$$a \vdash b = (\forall \Delta : (a^l(\Delta) \geq b^l(\Delta)) \wedge (a^u(\Delta) \leq b^u(\Delta))) .$$

In other words,  $a$  complies to  $b$  ( $a \vdash b$ ) if for all values of  $\Delta$  the interval  $[a^l(\Delta), a^u(\Delta)]$  is enclosed by  $[b^l(\Delta), b^u(\Delta)]$ .

Following the introduced notation, for any VCC  $\alpha$ , we can define the input and output predicates for some component input  $i$  and output  $j$  as  $\phi_i^I(\alpha_i) =$

$(\alpha_i \vdash \alpha_i^A)$  and  $\phi_j^O(\alpha_j) = (\alpha_j \vdash \alpha_j^G)$ , respectively, where  $\alpha^A$  and  $\alpha^G$  are assume and guarantee curves provided by the component interface.

We would like to have that if the input predicates of a component are all satisfied, then it works correctly and all output predicates are satisfied. In other words the condition  $\bigwedge_{\forall i \in I} \phi_i^I(\alpha_i) \Rightarrow \bigwedge_{\forall j \in J} \phi_j^O(\alpha_j)$  must be satisfied by the interfaces of all components.

If we now connect several components, we want to be able to check if the whole system can work correctly by just checking whether their interfaces are compatible. This can be done by testing whether the relation  $\bigwedge_{\forall (j,i)} \phi_j^O(\alpha_j) \Rightarrow \bigwedge_{\forall (j,i)} \phi_i^I(\alpha_i)$  is satisfiable. In other words, we must check if there exists *some* environment in which the components can be composed. The relation is hence the weakest precondition on the environment of the system.

We also need to propagate information about the predicates between the interfaces, see also [8]. This way, we combine interface theory with constraints propagation, which enables parameterized design of component-based systems. We propagate the assume and guarantee curves of the input and output predicates through the interfaces. Each interface connection would have both assume and guarantee curves propagated in opposite directions. We connect the interfaces, i.e., the corresponding guarantee and assume curves, as  $\forall (j, i) : (\alpha^G(i) = \alpha^G(j)) \wedge (\alpha^A(j) = \alpha^A(i))$ .

Now, we can determine whether two abstract components are compatible by checking the compatibility of their interfaces. Let us suppose that the assume and guarantee variables of an interface of any component and their relation to the input and output values of the corresponding abstract component satisfy

$$(\forall i \in I : \alpha_i \vdash \alpha_i^G \vdash \alpha_i^A) \Rightarrow (\forall j \in J : \alpha_j \vdash \alpha_j^G \vdash \alpha_j^A) , \quad (12)$$

where the component has inputs  $I$  and outputs  $J$ . Then if for a network of components, the relation  $\alpha_i^G \vdash \alpha_i^A$  is satisfied for all inputs  $i$ , we can conclude that the system works correctly.

Now we need to develop the relations between guarantees and assumptions in order to satisfy (12) for every component. We will first describe a general method how these relations can be determined and then apply it to the abstract components described so far.

To this end, as we are dealing with stateless interfaces,  $I$  and  $O$  can be related by a *transfer function*, e.g.,  $O = F(I)$ . The actual function depends on the processing semantics of the modeled component.

We need to define the concept of a monotone abstract component. Note that the 'complies to' relation  $\vdash$  has been generalized to tuples, i.e.,  $(a_i : i \in I) \vdash (b_i : i \in I)$  equals  $\forall i \in I : a_i \vdash b_i$ .

**Definition 1.** *An abstract component with a set of input and output ports,  $I$  and  $J$ , respectively, and a transfer function  $F$  that maps input curves to output curves, is monotone if  $((\tilde{\alpha}_i : i \in I) \vdash (\alpha_i : i \in I)) \Rightarrow ((\tilde{\alpha}_j : j \in J) \vdash (\alpha_j : j \in J))$  where  $(\alpha_j : j \in J) = F(\alpha_i : i \in I)$  and  $(\tilde{\alpha}_j : j \in J) = F(\tilde{\alpha}_i : i \in I)$ .*

In other words, if we replace the input curves of an abstract component with curves that are compliant, then the new output curves are also compliant to the

previous ones. Note that all components we look at in this paper satisfy this monotonicity condition, see for example (1), (2), and (8).

The following theorem leads to a constructive way to compute the input assumes and output guarantees from the given input guarantees and output assumes. We make use of the individual components of the transfer function  $F$ , i.e.,  $\alpha_j = F_j(\alpha_i : i \in I)$  for all  $j \in J$  where  $I$  and  $J$  denote the input and output ports of the corresponding abstract component, respectively. The theorem establishes that we can simply determine the output guarantees using the components of a given transfer function of an abstract component. For the input assumes we need to determine inverses of the transfer function  $F_j$  with respect to at least one of its arguments. All arguments of some  $F_j$  are determined by the input guarantees but one, say for example  $\alpha_{i^*}^G$ . This one we replace by  $\alpha_{i^*}^A$  and try to determine this curve such that the result of the transfer function still complies to the given output assumes. If we choose the same  $i^*$  for several components of the output function, then the resulting  $\alpha_{i^*}^A$  needs to comply to all partial 'inverses'.

**Theorem 1.** *Given a monotone component with input ports  $I$ , output ports  $J$ , and a transfer function  $F$  that maps input curves to output curves, i.e.,  $(\alpha_j : j \in J) = F(\alpha_i : i \in I)$ . Let us suppose that we determine the output guarantees using:*

$$\alpha_j^G = F_j(\alpha_i^G : i \in I) \quad \forall j \in J, \quad (13)$$

and the input assumes are computed such that

$$\forall j \in J \exists i^* \in I : \left( F_j(\alpha_i^G : i \in I) \Big|_{\alpha_{i^*}^G \leftarrow \alpha_{i^*}^A} \vdash \alpha_j^A \right), \quad (14)$$

where  $\alpha_{i^*}^G \leftarrow \alpha_{i^*}^A$  denotes that in the preceding term  $\alpha_{i^*}^G$  is replaced by  $\alpha_{i^*}^A$ .

Then (12) holds.

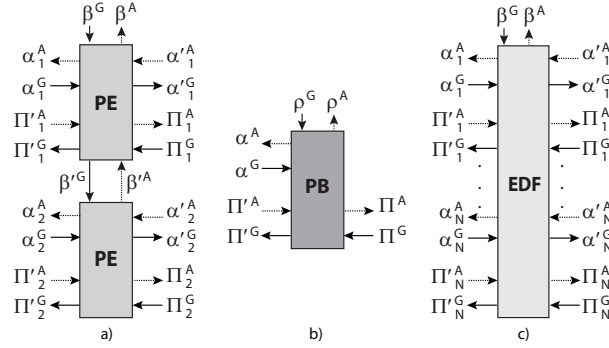
*Proof.* Let us assume that for all input ports  $i \in I$  we have  $\alpha_i \vdash \alpha_i^G$ , see (12). Using the monotonicity of  $F$ , we can now see that  $(\forall i \in I : \alpha_i \vdash \alpha_i^G) \Rightarrow F(\alpha_i : i \in I) \vdash F(\alpha_i^G : i \in I) \Rightarrow (\forall j \in J : \alpha_j \vdash \alpha_j^G)$ .

We still need to show that  $(\forall i \in I : \alpha_i^G \vdash \alpha_i^A) \Rightarrow (\forall j \in J : \alpha_j^G \vdash \alpha_j^A)$  using the construction in (13). At first note that this expression is equivalent to  $\forall j \in J \exists i^* \in I : ((\alpha_{i^*}^G \vdash \alpha_{i^*}^A) \Rightarrow (\alpha_j^G \vdash \alpha_j^A))$ . We also know that for any  $i^* \in I$  we have  $(\alpha_{i^*}^G \vdash \alpha_{i^*}^A) \Rightarrow ((\alpha_i^G : i \in I) \vdash (\alpha_i^G : i \in I) \Big|_{\alpha_{i^*}^G \leftarrow \alpha_{i^*}^A})$ .

Because of the monotonicity of  $F$  we can derive that for any  $i^* \in I$  we have  $(\alpha_{i^*}^G \vdash \alpha_{i^*}^A) \Rightarrow (F(\alpha_i^G : i \in I) \vdash F(\alpha_i^G : i \in I) \Big|_{\alpha_{i^*}^G \leftarrow \alpha_{i^*}^A})$ , and using (13) we find  $\forall j \in J \exists i^* \in I$  such that  $((\alpha_{i^*}^G \vdash \alpha_{i^*}^A) \Rightarrow (F_j(\alpha_i^G : i \in I) \vdash F_j(\alpha_i^G : i \in I) \Big|_{\alpha_{i^*}^G \leftarrow \alpha_{i^*}^A})) \Rightarrow (\alpha_j^G \vdash \alpha_j^A)$ .  $\square$

Next, we show how to compute the largest upper curve and smallest lower curve for which the respective relations still hold. This leads to the weakest possible input assumptions. We do this for the three types of components introduced so far.





**Fig. 1.** Interface models for: a) two PE components processing two streams with FP scheduling b) PB component c) EDF component processing  $N$  streams.

### 3.1 Processing Element

Now, using the relation between interface values, assumptions and guarantees in (12), and following the results from Theorem 1, we can deduce that the equations describing the output guarantees are equivalent to those for the abstract component, i.e., (1), just using interface guarantees instead of values. Therefore, we have:

$$\alpha'^{uG} = \alpha^{uG} \circledast \beta^{lG}, \quad \alpha'^{lG} = \alpha^{lG} \otimes \beta^{lG} .$$

In order to calculate the input assumptions of the PE abstract component, we need to determine inverse relations corresponding to (1) and (3). Following results from Network Calculus [5], we can do this by determining the pseudo-inverse functions which have the following definition  $f^{-1}(x) = \inf\{t : f(t) \geq x\}$ .

In order to guarantee that all relations hold if the input and output predicates are satisfied, we then need to use the minimum (in case of the upper curves) or the maximum (in case of the lower curves) of all the determined pseudo-inverses.

From the pseudo-inverses of (1), we get the inequalities  $\alpha^{lA} \geq \alpha'^{lA} \circledast \beta^{lG}$  and  $\beta^{lA} \geq \alpha'^{lA} \circledast \alpha^{lG}$ . Here we use the duality relation between the  $\circledast$  and  $\otimes$  operators (see the Appendix). Similarly, we get the inequalities  $\beta^{lA} \geq \alpha^{uG} \circledast \alpha'^{uA}$  and  $\alpha^{uA} \leq \beta^{lG} \otimes \alpha'^{uA}$ . Inverting the buffer overflow constraint (3) is trivial and we get the inequalities  $\alpha^{uA} \leq \beta^{lG} + b$  and  $\beta^{lA} \geq \alpha^{uG} - b_{max}$ .

If a PE component shares the service it receives with other lower priority PE components, the remaining service is bounded by (2). In terms of output guaranteed values, this can be expressed as  $\beta'^{lG}(\Delta) = RT(\beta^{lG}, \alpha^{uG})$  where the  $RT$  operator is defined in (2). In order to obtain the input assumptions of a component using FP scheduling, we need to use the inverses of the  $RT$  operator (see the Appendix).

After combining all inverses, the assumptions related to component PE can be determined as follows:

$$\begin{aligned}\alpha^{uA} &= \min\{\beta^{lG} \otimes \alpha'^{uA}, \beta^{lG} + b_{\max}, RT^{-\alpha}(\beta^{lA}, \beta^{lG})\}, & \alpha^{lA} &= \alpha^{lA} \circlearrowleft \beta^{lG}, \\ \beta^{lA} &= \max\{\alpha^{lA} \circlearrowleft \alpha^{lG}, \alpha^{uG} \circlearrowleft \alpha'^{uA}, \alpha^{uG} - b_{\max}, RT^{-\beta}(\beta^{lA}, \alpha^{uG})\} .\end{aligned}\quad (15)$$

The interface connections for two PE components are illustrated in Fig.1a.

### 3.2 Playout Buffer

For a PB component, the relations are simpler. We only need to determine the inverse relations for the buffer constraints (5), which directly yield the following relations:

$$\begin{aligned}\alpha^{uA} &= \rho^{lG} + B_{\max} - B_0, & \alpha^{lA} &= \rho^{uG} - B_0, \\ \rho^{uA} &= \alpha^{lG} + B_0, & \rho^{lA} &= \alpha^{uG} - (B_{\max} - B_0) .\end{aligned}\quad (16)$$

The interface connections for a single PB component are illustrated in Fig.1b.

### 3.3 Earliest Deadline First Component

Similarly to the PE component, equations describing the output guarantees are again equivalent to those for the abstract component, i.e., (8). They only need to be expressed in terms of interface variables instead of values for all streams  $i$ :

$$\alpha_i'^{uG}(\Delta) = r(\alpha_i^{uG}, -(D_i - \text{BCET}_i), \Delta), \quad \alpha_i^{lG}(\Delta) = r(\alpha_i^{lG}, (D_i - \text{BCET}_i), \Delta) ,$$

using the definition of the shift function  $r$  in (4).

Similarly, for the resource and buffer constraints, (7) and (9), we obtain:

$$\begin{aligned}\sum_{i=1}^N r(\alpha_i^{uG}, D_i, \Delta) &\leq \beta^{lG}(\Delta), \quad \forall \Delta \in \mathbb{R}^{\geq 0}, \\ \alpha_i^{uG}(D_i) &\leq b_{i \max}, \quad \forall i .\end{aligned}$$

Determining the input assumptions of the EDF component also involves finding the pseudo-inverse functions of the relations. Finding the input assumes for the upper arrival curves involves inverting (7) and (9). Again, we need to compute the largest upper curves for which the relations still hold. Finding the inverses and combing them, we find for all streams  $i$ :

$$\begin{aligned}\alpha_i^{uA}(\Delta) &= \min \left\{ \beta^{lG}(\Delta + D_i) - \sum_{\substack{j=1 \\ j \neq i}}^N r(\alpha_j^{uG}, (D_j - D_i), \Delta), \right. \\ &\quad \left. s(\alpha_i'^{uA}, (D_i - \text{BCET}_i), \Delta), t(D_i, b_{i \max}, \Delta) \right\} ,\end{aligned}$$

using functions  $s(\alpha, c, \Delta)$  and  $t(d, b, \Delta)$  defined as:

$$s(\alpha, c, \Delta) = \begin{cases} \alpha(\Delta - c) & \text{if } \Delta > c \\ \lim_{\epsilon \rightarrow 0} \{\alpha(\epsilon)\} & \text{if } 0 < \Delta \leq c \\ 0 & \text{if } \Delta = 0 \end{cases} \quad t(d, b, \Delta) = \begin{cases} \infty & \text{if } \Delta > d \\ b & \text{if } 0 < \Delta \leq d \\ 0 & \text{if } \Delta = 0 \end{cases}$$

Calculating the input assumption for the lower curve is much simpler as it involves finding the smallest lower curve solution to the pseudo-inverse of (8) or  $\alpha_i^{lA}(\Delta) \geq \alpha_i^{lA}(\Delta + (D_i - \text{BCET}_i))$  for all  $i$ . Therefore, we can determine the following assume interface function for the lower curve of each input data stream:

$$\alpha_i^{lA}(\Delta) = r(\alpha_i^{lA}, -(D_i - \text{BCET}_i), \Delta), \quad \forall i,$$

using the shift function  $r$  as defined in (4).

Similarly, for the assume of the lower service curve we invert (7) which yields the inequality  $\beta^{lA}(\Delta) \geq \sum_{i=1}^N r(\alpha_i^{uG}, D_i, \Delta)$ . Therefore, the input assume for the lower service curve of an EDF component can be determined as:

$$\beta^{lA}(\Delta) = \sum_{i=1}^N r(\alpha_i^{uG}, D_i, \Delta). \quad (17)$$

The interface model for the EDF component is illustrated in Fig.1c.

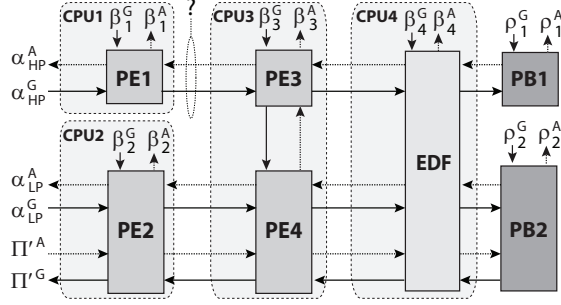
### 3.4 Worst-Case Traversal Time Interface

We develop an additional type of interface to alleviate design of systems with WCTT constraints. It is an interface-based interpretation of the analytical computation of WCTT with (11).

The 'complies to' relation  $\vdash$  for this interface connection is defined as  $\Pi^G(\Delta) \vdash \Pi^A(\Delta) = (\forall \Delta : \Pi^G(\Delta) \geq \Pi^A(\Delta))$ , where  $\Pi^A$  expresses the minimum service requested from all subsequent components such that the WCTT constraint is satisfied, and  $\Pi^G$  expresses the minimum service guaranteed by all subsequent components.

Computing the guarantee for a sequence of components follows directly from (11) and can be done with  $\Pi^G = \beta_{\mathcal{PE}}^G \otimes \rho_{\mathcal{PB}}^G \otimes \beta_{\mathcal{EDF}}^G$ . Connecting a PE component to the sequence would change the combined service to  $\Pi'^G = \beta^{lG} \otimes \Pi^G$  where  $\beta^{lG}$  is the lower service guaranteed by the PE. Similarly, connecting a PB component we would have  $\Pi'^G = \rho_{\tau}^{lG} \otimes \Pi^G$ , where  $\rho_{\tau}^l(\Delta)$  is the lower guaranteed shifted readout curve as defined with (6). For an EDF component, we have  $\Pi'^G = \beta_{D_i}^{lG} \otimes \Pi^G$  where  $\beta_{D_i}^{lG}$  is the service curve for the stream with relative deadline  $D_i$  as defined in (10).

Inverting (11), we can compute the assume on the combined service of a sequence of components as  $\Pi^A = r(\alpha^{uG}, D, \Delta)$  which expresses the minimum necessary service in order to meet a WCTT constraint of  $D$  for the input  $\alpha^{uG}$ .



**Fig. 2.** Interface model of an example stream processing system.

Propagating the assume value through a sequence of components can be done for the three types of components by inverting (11) as follows:

$$\text{PE: } \Pi^A = \Pi'^A \circ \beta^{lG}, \quad \text{PB: } \Pi^A = \Pi'^A \circ \rho_{\tau}^{lG}, \quad \text{EDF: } \Pi^A = \Pi'^A \circ \beta_{D_i}^{lG}.$$

We can also compute component-wise constraints on the resources provided by each component given the resource assumption from preceding components  $\Pi'^A$ , and the resource guarantee from subsequent components  $\Pi^G$ :

$$\text{PE: } \beta^{lA} \geq \Pi'^A \circ \Pi^G, \quad \text{PB: } \rho_{\tau}^{lA} \geq \Pi'^A \circ \Pi^G, \quad \text{EDF: } \beta_{D_i}^{lA} \geq \Pi'^A \circ \Pi^G.$$

The above constraints can be combined with the previously computed input assumes for the resources of the three components with (15), (16), and (17). By doing this, satisfying all interface relations of components composed in a sequence will guarantee that the WCTT constraint on the sequence of components is satisfied. The WCTT interfaces for the PE, PB, and EDF components are shown in Fig.1.

## 4 Illustrative Example

In this section we show how our proposed theory can be applied to an example system shown in Fig.2. Towards this, each PE, PB, and EDF component 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 or underflow, and in addition, the WCTT constraints are satisfied.

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, deadline) are reflected (or summarized) through its interfaces. We show that if these internal details are changed then the component's interfaces also change and two previously compatible components may become incompatible (or vice versa).

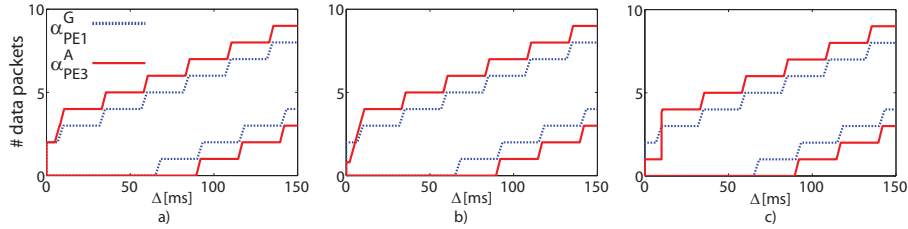
**Experimental Setup.** We consider the system illustrated in Fig.2. It consists of a multiprocessor platform with four CPUs. A distributed application is mapped to the platform. It processes two data streams, a high priority one denoted as *HP*, and a low priority one denoted as *LP*. The application consists of six tasks. Streams are preprocessed by the tasks modeled with components *PE1* and *PE2* which are mapped separately to *CPU1* and *CPU2*, respectively. Afterwards, they are processed by components *PE3* and *PE4* which are mapped to *CPU3*. The tasks share the resource using FP scheduling where stream *HP* is given higher priority. Additionally, streams are processed by two tasks mapped to *CPU4* which they share with the EDF policy. This is modeled with the EDF component. The fully processed streams are written to playout buffers which are read by an I/O interface at a constant rate. The buffers are modeled with components *PB1* and *PB2*. For simplicity, the communication is not modeled here. If necessary, it can be taken into account by additional components in the model.

Data packets from the streams have bursty arrivals described with period  $p$ , jitter  $j$ , and minimum inter-arrival distance  $d$ . For the *HP* stream the parameters are  $p = 25\text{ms}$ ,  $j = 40\text{ms}$ ,  $d = 0.1\text{ms}$ , and for *LP* stream they are  $p = 25\text{ms}$ ,  $j = 30\text{ms}$ ,  $d = 0.1\text{ms}$ . Each data packet from the two streams has a constant processing demand of 1 cycle. *CPU1* is fully available with service availability of 0.3 cycles/ms. For *CPU2*, *CPU3*, and *CPU4*, the respective service availabilities are 0.3, 0.4, and 0.4 cycles/ms. Components *PE3* and *PE4* have internal buffer sizes of 2 and 3 packets, respectively. The buffers should never overflow. The EDF component schedules tasks processing streams *HP* and *LP* with relative deadlines of 8ms and 10ms, respectively, with both buffers being limited to 3 packets. These buffers should also never overflow. Components *PB1* and *PB2* are read at a constant rate of 25 packets/ms. Both components have maximum buffer sizes of 8 data packets, and initially they contain 4 data packets. Both buffers should not underflow and overflow. Additionally, we have a WCTT constraint on the *LP* stream of 200ms.

**Results.** We consider three different scenarios of the system's parameters. In each of them, we check the compatibility of component *PE1* with the partially designed system when all other components are already connected. Compatibility is checked by only inspecting the interface connection between *PE1* and the system which is marked with '?' in Fig.2. Compatibility meaning that the output guarantee is fully "enclosed" by the input assumption.

*Case I:* The system is considered with the specified parameters. The components turn out to be compatible. The interface connection is illustrated in Fig.3a. It shows that the guarantee on the output stream rate  $\alpha_{PE1}^G$  expressed by *PE1*'s interface is compatible with the assumption on the input rate  $\alpha_{PE3}^A$  expressed by *PE3*'s interface.

*Case II:* The WCTT constraint on the *LP* stream is decreased to 192ms. This leads to incompatibility between components *PE1* and *PE3* which reveals in the interface connection as shown in Fig.3b.



**Fig. 3.** Interface connection between the output guarantee of component *PE1* and the input assumption of component *PE3* shows: a) compatibility b) incompatibility when WCTT for stream *LP* is reduced to 192ms c) incompatibility when buffer of component *PB2* is decreased to 5 packets.

*Case III:* The maximum buffer size of component *PB2* is decreased to 5 packets which leads to incompatibility as shown in Fig.3c.

In summary, we have shown through a concrete example how incremental compatibility checking can be done using the proposed interfaces. Clearly, such interfaces can also be used in a straightforward manner for resource dimensioning and component-level design space exploration. 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? What is the minimum processing frequency such that the interface is still compatible? Or what are the feasible relative deadlines in an EDF component? In this paper, we are concerned with buffer and WCTT constraints however, one can imagine developing similar interfaces for power, energy, and temperature constraints.

## 5 Concluding Remarks

In this paper we proposed an interface algebra for checking whether multiple components of an embedded system may be composed together while satisfying their worst-case traversal time (WCTT) constraints. The main advantage of such an interface-based formulation is that component composition only requires a compatibility checking of the interfaces of the components involved, without having to compute the WCTT of the entire component network from scratch, each time a new component is added or an existing component is modified. This has a number of advantages. It significantly reduces design complexity, it does not require components to expose the details of their internals, and it allows a correct-by-construction design flow.

The interfaces studied here were purely functional in nature, i.e., they do not contain any state information. This might be restrictive in a number of settings, e.g., when the components implement complex protocols. As an example, the processing rate of a component might depend on the “state” or the fill level of an internal buffer. As a part of future work, we plan to extend our interface algebra to accommodate such “stateful” components. This may be done by describing an automaton to represent an interface, with language inclusion or equivalence to denote the notion of *compatibility* between components.

## Appendix: Min-Max Algebra

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

$$(f \otimes g)(\Delta) = \inf_{0 \leq \lambda \leq \Delta} \{f(\Delta - \lambda) + g(\lambda)\}, \quad (f \oslash g)(\Delta) = \sup_{\lambda \geq 0} \{f(\Delta + \lambda) - g(\lambda)\}.$$

The duality between  $\otimes$  and  $\oslash$  states that:  $f \oslash g \leq h \iff f \leq g \otimes h$  .

The inverses of the  $RT(\beta, \alpha)$  are defined as:

$$\alpha = RT^{-\alpha}(\beta', \beta) \Rightarrow \beta' \leq RT(\beta, \alpha), \quad \beta = RT^{-\beta}(\beta', \alpha) \Rightarrow \beta' \leq RT(\beta, \alpha),$$

with solutions:

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

## References

1. Alfaro, L.d., Henzinger, T.A.: Interface theories for component-based design. In: 1st International Workshop on Embedded Software (EMSOFT). pp. 148–165 (2001)
2. Baruah, S., Chen, D., Gorinsky, S., Mok, A.: Generalized multiframe tasks. Real-Time Systems 17(1), 5–22 (1999)
3. Chakraborty, S., Kunzli, S., Thiele, L.: A general framework for analysing system properties in platform-based embedded system designs. In: Design, Automation and Test in Europe (DATE). p. 10190 (2003)
4. Chakraborty, S., Liu, Y., Stoimenov, N., Thiele, L., Wandeler, E.: Interface-based rate analysis of embedded systems. In: 27th IEEE International Real-Time Systems Symposium (RTSS). pp. 25–34 (2006)
5. Le Boudec, J.Y., Thiran, P.: Network Calculus: A Theory of Deterministic Queuing Systems for the Internet. Springer Berlin / Heidelberg (2001)
6. Maxiaguine, A., Künzli, S., Thiele, L.: Workload characterization model for tasks with variable execution demand. In: Design, Automation and Test in Europe (DATE). p. 21040 (2004)
7. Thiele, L., Chakraborty, S., Naedele, M.: Real-time calculus for scheduling hard real-time systems. In: IEEE International Symposium on Circuits and Systems (ISCAS). vol. 4, pp. 101–104 (2000)
8. Thiele, L., Wandeler, E., Stoimenov, N.: Real-time interfaces for composing real-time systems. In: 6th ACM & IEEE International Conference on Embedded Software (EMSOFT). pp. 34–43 (2006)
9. Wandeler, E., Maxiaguine, A., Thiele, L.: Performance analysis of greedy shapers in real-time systems. In: Design, Automation and Test in Europe (DATE). pp. 444–449 (2006)
10. Wandeler, E., Thiele, L.: Interface-based design of real-time systems with hierarchical scheduling. In: 12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS). pp. 243–252 (2006)
11. Wandeler, E., Thiele, L.: Optimal TDMA time slot and cycle length allocation for hard real-time systems. In: Asia and South Pacific Design Automation Conference (ASP-DAC). pp. 479–484 (2006)