Chapter 4
Interface-Based Design of Real-Time Systems

Nikolay Stoimenov, Samarjit Chakraborty, and Lothar Thiele

4.1 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. Such important analysis is usually related to verifying that buffers in components never overflow or underflow, end-to-end delays or worst-case traversal times (WCTTs) of data through a component network fall within given deadlines, and others.

Typically performance analysis methods are used for the analysis of a component-based real-time system design a posteriori. This means that a real-time system is designed and dimensioned in a first step, and only after completion of this first step, the performance analysis is applied to the system design in a second step. The analysis result will then give an answer to the binary question whether the system design that was developed in the first step meets all real-time requirements, or not. A designer must then go back to the first step, change the design, and iterate on the two steps until an appropriate system design is found.

N. Stoimenov
Computer Engineering and Networks Laboratory, ETH Zurich, 8092 Zurich, Switzerland
e-mail: stoimenov@tik.ee.ethz.ch

S. Chakraborty
Institute for Real-Time Computer Systems, TU Munich, 80290 Munich, Germany
e-mail: samarjit@tum.de

L. Thiele (✉)
Computer Engineering and Networks Laboratory, ETH Zurich, 8092 Zurich, Switzerland
e-mail: thiele@tik.ee.ethz.ch

S. Chakraborty and J. Eberspächer (eds.), *Advances in Real-Time Systems*,
DOI 10.1007/978-3-642-24349-3_4,

Unlike this two-step approach is the idea of *interface-based design* described by de Alfaro and Henzinger in [1, 2]. It proposes a holistic one-step approach toward design and analysis of systems where components have interfaces, and a designer can decide whether two components can be connected and work together based only on the information exposed in their interfaces. Interface-based design avoids modeling the internals of each component, which is often difficult because of the complexity of the components and their proprietary nature.

In interface-based design, components are described by component interfaces. A component interface models how a component can be used, which is in contrast to an abstract component that models what a component does. Through *input assumptions*, a component interface models the expectations that a component has from the other components in the system and the environment. Through *output guarantees*, a component interface tells the other components in the system and the environment what they can expect from this component. The major goal of a good component interface is then to provide only the appropriate information that is sufficient to decide whether two or more components can work together properly. In the context of component interfaces for real-time system performance analysis, the term "properly" refers to questions like: Does the composed system satisfy all requested real-time properties such as delay, buffer, and throughput constraints?

Consequently, in an interface-based real-time system design approach, the compliance to real-time constraints is checked at composition time. That is, the successful composition of a set of components and their interfaces to a complete system design already guarantees the satisfaction of all real-time constraints, and no further analysis steps are required. This leads to faster design processes and partly removes the need for the classical binary search iteration approach to find appropriate system designs.

Additionally, an interface-based real-time system design approach also benefits from the properties of incremental design and independent implementability that are elementary features of an interface-based design. The support for *incremental design* ensures that component interfaces can be composed one-by-one into subsystems in any order. And if at any step a component interface cannot be composed with a subsystem, this already excludes the possibility that the complete system can be composed successfully, and therefore can not work properly. *Refinement* on the other hand is very similar to subtyping of classes in object-oriented programming. A component interface can be refined by another component interface if it accepts at least all inputs of the original interface and produces only a subset of the original outputs. Fulfilling these constraints ensures that components with compatible interfaces can be refined independently and still remain compatible, thus supporting *independent implementability*.

In this chapter, we develop an *interface algebra* for verifying buffer overflow and underflow constraints, and estimating worst-case traversal times in order to verify their compliance with 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 from sensors to actuators while getting processed on multiple processors. Our proposed interface algebra is based on *Rate and Real-Time Interfaces* as proposed in [5, 9], respectively, and is motivated by the concept of *assume/guarantee* interfaces from [1]. In particular, we cast the *Real-Time Calculus* framework from [4,8] within the assume/guarantee interface setting. At a high level, two interfaces are *compatible* when the guarantees associated with one of them comply with the assumptions associated with the other. We describe how to compute the *assumptions* and *guarantees* associated with interfaces based on a *monotonicity* property. This result is then used to verify buffer underflow and overflow constraints, 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 hardware/software component. This involves defining proper abstractions for component inputs and outputs, and internal component relations that meaningfully relate abstract inputs to abstract outputs. Such components can be composed together to create a system model, and together with a model of the environment can be used to perform timing analysis. Several such abstract components are described in Sect. 4.2.
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. In Sect. 4.3 we describe how one can do this for the abstract components introduced in Sect. 4.2.
3. Derive the internal interface relations that relate incoming guarantees and assumptions to outgoing guarantees and assumptions of a component's interfaces. This is also described in Sect. 4.3.

4.2 Timing Analysis of Component Networks

In this section, we describe a timing analysis framework in particular, for verifying buffer overflow and underflow constraints and computing worst-case traversal times for component networks. The framework is based on Real-Time Calculus [4, 8]. This calculus is an adaptation of Network Calculus [6] 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. Component models for greedy shapers, time division multiple access, polling servers, and hierarchical scheduling are described in [10–12]. In Sect. 4.3 we will lift this framework to an interface-based design setting. First we need to introduce the basic models and abstractions that underlie Real-Time Calculus before we describe the three basic abstract components that we consider.

4.2.1 Basic Models

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 [6] are specific instances of VCCs and are more expressive than traditional event stream models such as the periodic, periodic with jitter, sporadic, etc. Arrival curves $\alpha^{\mathrm{l}}(\Delta)$ and $\alpha^{\mathrm{u}}(\Delta)$ denote the minimum and the maximum number of events that can arrive in *any* time interval of length Δ, i.e. $\alpha^{\mathrm{l}}(t-s) \leq R(t) - R(s) \leq \alpha^{\mathrm{u}}(t-s)$ for all $t > s \geq 0$. In addition, $\alpha^{\mathrm{l}}(0) = \alpha^{\mathrm{u}}(0) = 0$. We also denote the tuple $(\alpha^{\mathrm{l}}, \alpha^{\mathrm{u}})$ with α. Service curves characterize the variability in the service provided by a resource. The curves $\beta^{\mathrm{l}}(\Delta)$ and $\beta^{\mathrm{u}}(\Delta)$ denote the minimum and the maximum number of events that can be processed within *any* time interval of length Δ, i.e. $\beta^{\mathrm{l}}(t-s) \leq C(t) - C(s) \leq \beta^{\mathrm{u}}(t-s)$ for all $t > s \geq 0$. In addition, $\beta^{\mathrm{l}}(0) = \beta^{\mathrm{u}}(0) = 0$. We also denote the tuple $(\beta^{\mathrm{l}}, \beta^{\mathrm{u}})$ with β. An event stream modeled by $\alpha(\Delta)$ enters a component and is processed using the resource modeled as $\beta^{\mathrm{l}}(\Delta)$. The output is again an event stream $\alpha'(\Delta)$, and the *remaining* resource is expressed as $\beta'^{\mathrm{l}}(\Delta)$. 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, for details refer to [7].

4.2.2 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 [4, 6], such a component can be modeled by an abstract component *PE* with the following internal component relations[1]:

$$\alpha'^{u}(\Delta) = (\alpha^{u} \oslash \beta^{l})(\Delta), \tag{4.1}$$

$$\alpha'^{l}(\Delta) = (\alpha^{l} \otimes \beta^{l})(\Delta), \tag{4.2}$$

$$\beta'^{l}(\Delta) = \sup_{0 \le \lambda \le \Delta} \left\{ \beta^{l}(\lambda) - \alpha^{u}(\lambda) \right\} := RT(\beta^{l}, \alpha^{u}), \tag{4.3}$$

and the backlog of the input buffer is bounded by:

$$\sup_{0 \le \lambda \le \Delta} \left\{ \alpha^{u}(\lambda) - \beta^{l}(\lambda) \right\}.$$

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:

$$\alpha^{u}(\Delta) \le \beta^{l}(\Delta) + b_{\max}, \quad \forall \Delta \in \mathbb{R}^{\ge 0}. \tag{4.4}$$

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:

$$r(\alpha, c, \Delta) = \begin{cases} \alpha(\Delta - c) & \text{if } (\Delta > c) \wedge (\Delta \neq 0) \\ 0 & \text{if } (\Delta \le c) \vee (\Delta = 0) \end{cases} \tag{4.5}$$

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:

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

[1] See the Appendix at the end of the chapter for definitions of the operators $\otimes$ and $\oslash$.

If all events of the input stream must be processed by a PE component within a relative deadline D, then for the stream to be schedulable we must have that $\mathrm{Del}(\alpha^{\mathrm{u}}, \beta^{\mathrm{l}}) \leq D$ which can be written as:

$$\beta^{\mathrm{l}}(\Delta) \geq r(\alpha^{\mathrm{u}}, D, \Delta), \quad \forall \Delta \in \mathbb{R}^{\geq 0}.$$

The above inequality gives us an expression for the minimum service in component PE that is required in order to meet a deadline constraint.

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 (4.3), 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.

4.2.3 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 $\rho(\Delta) = (\rho^{\mathrm{l}}(\Delta), \rho^{\mathrm{u}}(\Delta))$, i.e. $\rho^{\mathrm{l}}(t-s) \leq P(t) - P(s) \leq \rho^{\mathrm{u}}(t-s)$ for all $t > s \geq 0$. What needs to be guaranteed is that the playout buffer never *overflows or underflows*. Following results from Real-Time and Rate Interfaces [5], for a PB component with input and readout event streams characterized by the VCCs α and ρ, 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 the following constraints are satisfied:

$$\alpha^{\mathrm{l}}(\Delta) \geq \rho^{\mathrm{u}}(\Delta) - B_0, \quad \forall \Delta \in \mathbb{R}^{\geq 0}, \tag{4.6}$$

$$\alpha^{\mathrm{u}}(\Delta) \leq \rho^{\mathrm{l}}(\Delta) + B_{\max} - B_0, \quad \forall \Delta \in \mathbb{R}^{\geq 0}. \tag{4.7}$$

These are component-wise constraints that guarantee for a PB component to never overflow or underflow.

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

$$\rho^{\mathrm{l}}_{\tau}(\Delta) = r(\rho^{\mathrm{l}}, \tau, \Delta) \tag{4.8}$$

is the lower readout curve "shifted to the right" by the initial playback delay $\tau \geq 0$ necessary to accumulate B_0 events. Similarly to a PE component, meeting a relative deadline constraint of D in a PB component would require for the input stream that we have:

$$\rho^{\mathrm{l}}_{\tau}(\Delta) \geq r(\alpha^{\mathrm{u}}, D, \Delta), \quad \forall \Delta \in \mathbb{R}^{\geq 0},$$

where r is the shift function defined in (4.5).

4.2.4 Earliest Deadline First Component

The EDF component is similar to the PE component 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 BCET_i, a worst-case execution time WCET_i, and a relative deadline D_i where $0 \leq \mathrm{BCET}_i \leq \mathrm{WCET}_i \leq D_i$. 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 β and event streams characterized by arrival curves α_i, all tasks are processed within their deadlines if and only if:

$$\beta^{\mathrm{l}}(\Delta) \geq \sum_{i=1}^{N} r(\alpha_i^{\mathrm{u}}, D_i, \Delta), \quad \forall \Delta \in \mathbb{R}^{\geq 0}, \tag{4.9}$$

using the shift function r from (4.5). The output streams can be characterized by arrival curves computed for all streams i as:

$$\alpha'^{\mathrm{u}}_i(\Delta) = r(\alpha_i^{\mathrm{u}}, -(D_i - \mathrm{BCET}_i), \Delta), \tag{4.10}$$

$$\alpha'^{\mathrm{l}}_i(\Delta) = r(\alpha_i^{\mathrm{l}}, (D_i - \mathrm{BCET}_i), \Delta), \tag{4.11}$$

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

$$\alpha_i^{\mathrm{u}}(D_i) \leq b_{i\,\max}, \quad \forall i. \tag{4.12}$$

The EDF component schedulability condition (4.9) can be related to the demand bound functions described by Baruah et al. in [3]. Given that this condition is satisfied, the service curve provided to each stream can be modeled with a burst-delay function as defined in [6] which is computed for each stream i as:

$$\beta^{\mathrm{l}}_{D_i}(\Delta) = \begin{cases} +\infty & \text{if } \Delta > D_i \\ 0 & \text{otherwise} \end{cases} \tag{4.13}$$

The WCTT experienced by an event from stream i can be computed as $\mathrm{Del}(\alpha_i^{\mathrm{u}}, \beta^{\mathrm{l}}_{D_i})$ which is upper bounded by D_i.

4.2.5 Worst-Case Traversal Times of Component Networks

The worst-case traversal time for 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" [6]. Following results from Network Calculus, this leads to the following computation for the WCTT.

For an input event stream α traversing a sequence of components which consists of a set of PEs, a set of PBs, and a set of EDF components denoted as $\mathscr{PE}$, $\mathscr{PB}$ and $\mathscr{EDF}$, respectively, the worst-case traversal time that an event can experience can be computed as $\mathrm{Del}(\alpha^{\mathrm{u}}, \beta_{\mathscr{PE}} \otimes \rho_{\mathscr{PB}} \otimes \beta_{\mathscr{EDF}})$ with $\beta_{\mathscr{PE}} = \bigotimes_{c \in \mathscr{PE}} \beta^{\mathrm{l}}_c$, $\rho_{\mathscr{PB}} = \bigotimes_{c \in \mathscr{PB}} \rho^{\mathrm{l}}_{\tau\, c}$, and $\beta_{\mathscr{EDF}} = \bigotimes_{c \in \mathscr{EDF}} \beta^{\mathrm{l}}_{D_i\, c}$, where β^{l}_c is the service availability of PE component c, $\rho^{\mathrm{l}}_{\tau\, c}$ is the lower readout curve for PB component c as defined with (4.8), and $\beta^{\mathrm{l}}_{D_i\, c}$ is the service availability provided to the stream served with relative deadline D_i by EDF component c as defined with (4.13). A WCTT constraint on the sequence of components $\mathrm{Del}(\alpha^{\mathrm{u}}, \beta_{\mathscr{PE}} \otimes \rho_{\mathscr{PB}} \otimes \beta_{\mathscr{EDF}}) \leq D$ can be written as follows:

$$\beta_{\mathscr{PE}} \otimes \rho_{\mathscr{PB}} \otimes \beta_{\mathscr{EDF}} \geq r(\alpha^{\mathrm{u}}, D, \Delta), \quad \forall \Delta \in \mathbb{R}^{\geq 0}, \tag{4.14}$$

using the shift function r from (4.5).

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 constraint propagation.

In our setup each component has two disjoint sets of input and output ports I and J. 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 $\phi^O(J)$. Here, working correctly means that the component satisfies all real-time constraints such as buffer underflow/overflow or delay constraints [9].

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

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

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

In the following, we will just use α to denote the characterization of *any* VCC that is an input or an output of an abstract component.

Following the introduced notation, for any VCC α, we can define the input and output predicates for some component input i and output j as $\phi_i^I(\alpha_i) = (\alpha_i \vdash \alpha_i^{\mathrm{A}})$ and $\phi_j^O(\alpha_j) = (\alpha_j \vdash \alpha_j^{\mathrm{G}})$, respectively, where α^{A} and α^{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 [9]. This way, we combine interface theory with constraint 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 as shown in Fig. 4.1. We connect the interfaces, i.e. the corresponding guarantee and assume curves, as $\forall (j,i) : \quad (\alpha^{\mathrm{G}}(i) = \alpha^{\mathrm{G}}(j)) \wedge (\alpha^{\mathrm{A}}(j) = \alpha^{\mathrm{A}}(i))$.

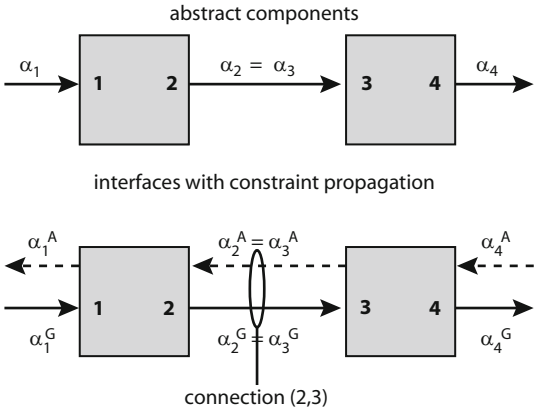

Fig. 4.1 Relation between input/output values of abstract components and assume/guarantee variables in interfaces with constraint propagation

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 that:

$$(\forall i \in I : \alpha_i \vdash \alpha_i^{\mathrm{G}} \vdash \alpha_i^{\mathrm{A}}) \Rightarrow (\forall j \in J : \alpha_j \vdash \alpha_j^{\mathrm{G}} \vdash \alpha_j^{\mathrm{A}}), \tag{4.15}$$

where the component has inputs I and outputs J. Then if for a network of components, the relation $\alpha_i^{\mathrm{G}} \vdash \alpha_i^{\mathrm{A}}$ 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 (4.15) 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 in Sect. 4.2.

To this end, as we are dealing with stateless interfaces, I and J can be related by a *transfer function*, e.g. $J = F(I)$. 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 4.1. 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 $((\widetilde{\alpha}_i : i \in I) \vdash (\alpha_i : i \in I)) \Rightarrow ((\widetilde{\alpha}_j : j \in J) \vdash (\alpha_j : j \in J))$ where $(\alpha_j : j \in J) = F(\alpha_i : i \in I)$ and $(\widetilde{\alpha}_j : j \in J) = F(\widetilde{\alpha}_i : i \in I)$.

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 described in Sect. 4.2 satisfy this monotonicity condition, see for example the transfer functions (4.1), (4.2), (4.3), (4.10), and (4.11).

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 α_{i*}^{G}. This one we replace by α_{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 α_{i*}^{A} needs to comply to all partial "inverses".

Theorem 4.1. *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:

$$\alpha_j^{\mathrm{G}} = F_j(\alpha_i^{\mathrm{G}} : i \in I), \quad \forall j \in J, \tag{4.16}$$

and the input assumes are computed such that:

$$\forall j \in J \;\; \exists i^* \in I : \left(F_j(\alpha_i^{\mathrm{G}} : i \in I)\Big|_{\alpha_{i^*}^{\mathrm{G}} \leftarrow \alpha_{i^*}^{\mathrm{A}}} \;\vdash\; \alpha_j^{\mathrm{A}} \right), \tag{4.17}$$

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

Then (4.15) holds.

Proof. Let us assume that for all input ports $i \in I$ we have $\alpha_i \vdash \alpha_i^{\mathrm{G}}$, see (4.15). Using the monotonicity of F, we can now see that $(\forall i \in I : \alpha_i \vdash \alpha_i^{\mathrm{G}}) \Rightarrow F(\alpha_i : i \in I) \vdash F(\alpha_i^{\mathrm{G}} : i \in I) \Rightarrow (\forall j \in J : \alpha_j \vdash \alpha_j^{\mathrm{G}})$.

We still need to show that $(\forall i \in I : \alpha_i^{\mathrm{G}} \vdash \alpha_i^{\mathrm{A}}) \Rightarrow (\forall j \in J : \alpha_j^{\mathrm{G}} \vdash \alpha_j^{\mathrm{A}})$ using the construction in (4.16). At first note that this expression is equivalent to $\forall j \in J \;\; \exists i^* \in I : \left((\alpha_{i^*}^{\mathrm{G}} \vdash \alpha_{i^*}^{\mathrm{A}}) \Rightarrow (\alpha_j^{\mathrm{G}} \vdash \alpha_j^{\mathrm{A}}) \right)$. We also know that for any $i^* \in I$ we have $(\alpha_{i^*}^{\mathrm{G}} \vdash \alpha_{i^*}^{\mathrm{A}}) \Rightarrow ((\alpha_i^{\mathrm{G}} : i \in I) \vdash (\alpha_i^{\mathrm{G}} : i \in I)\,|_{\alpha_{i^*}^{\mathrm{G}} \leftarrow \alpha_{i^*}^{\mathrm{A}}})$.

Because of the monotonicity of F we can derive that for any $i^* \in I$ we have $(\alpha_{i^*}^{\mathrm{G}} \vdash \alpha_{i^*}^{\mathrm{A}}) \Rightarrow (F(\alpha_i^{\mathrm{G}} : i \in I) \vdash F(\alpha_i^{\mathrm{G}} : i \in I)\,|_{\alpha_{i^*}^{\mathrm{G}} \leftarrow \alpha_{i^*}^{\mathrm{A}}})$, and using (4.16) we find $\forall j \in J \;\; \exists i^* \in I$ such that $((\alpha_{i^*}^{\mathrm{G}} \vdash \alpha_{i^*}^{\mathrm{A}}) \Rightarrow (F_j(\alpha_i^{\mathrm{G}} : i \in I) \vdash F_j(\alpha_i^{\mathrm{G}} : i \in I)\,|_{\alpha_{i^*}^{\mathrm{G}} \leftarrow \alpha_{i^*}^{\mathrm{A}}}) \Rightarrow (\alpha_j^{\mathrm{G}} \vdash \alpha_j^{\mathrm{A}}))$. □

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 in Sect. 4.2.

4.3.1 Processing Element

Now, using the relation between interface values, assumptions and guarantees in (4.15), and following the results from Theorem 4.1, we can deduce that the equations describing the output guarantees are equivalent to those for the abstract component, i.e. (4.1) and (4.2), but instead of using values, we use interface guarantees. Therefore, we have:

$$\alpha'^{\mathrm{uG}}(\Delta) = (\alpha^{\mathrm{uG}} \oslash \beta^{\mathrm{lG}})(\Delta),$$
$$\alpha'^{\mathrm{lG}}(\Delta) = (\alpha^{\mathrm{lG}} \otimes \beta^{\mathrm{lG}})(\Delta).$$

In order to calculate the input assumptions of the PE abstract component, we need to determine inverse relations corresponding to (4.1), (4.2), and (4.4). Following results from Network Calculus [6], 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 (4.2), we get the inequalities $\alpha^{\mathrm{lA}} \geq \alpha'^{\mathrm{lA}} \oslash \beta^{\mathrm{lG}}$ and $\beta^{\mathrm{lA}} \geq \alpha'^{\mathrm{lA}} \oslash \alpha^{\mathrm{lG}}$. Here we use the duality relation between the $\oslash$ and $\otimes$ operators (see the Appendix). Similarly, from the pseudo-inverses of (4.1), we get the inequalities $\beta^{\mathrm{lA}} \geq \alpha^{\mathrm{uG}} \oslash \alpha'^{\mathrm{uA}}$ and $\alpha^{\mathrm{uA}} \leq \beta^{\mathrm{lG}} \otimes \alpha'^{\mathrm{uA}}$. Inverting the buffer overflow constraint (4.4) is trivial and we get the inequalities $\alpha^{\mathrm{uA}} \leq \beta^{\mathrm{lG}} + b_{\max}$ and $\beta^{\mathrm{lA}} \geq \alpha^{\mathrm{uG}} - b_{\max}$.

If a PE component shares the service it receives with other lower priority PE components, the remaining service is bounded by (4.3). In terms of output guaranteed values, this can be expressed as $\beta'^{\mathrm{lG}}(\Delta) = RT(\beta^{\mathrm{lG}}, \alpha^{\mathrm{uG}})$ where the RT operator is defined in (4.3). 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:

$$\begin{aligned}
\alpha^{\mathrm{uA}} &= \min\left\{\beta^{\mathrm{lG}} \otimes \alpha'^{\mathrm{uA}}, \beta^{\mathrm{lG}} + b_{\max}, RT^{-\alpha}\left(\beta'^{\mathrm{lA}}, \beta^{\mathrm{lG}}\right)\right\}, \\
\alpha^{\mathrm{lA}} &= \alpha'^{\mathrm{lA}} \oslash \beta^{\mathrm{lG}}, \\
\beta^{\mathrm{lA}} &= \max\left\{\alpha'^{\mathrm{lA}} \oslash \alpha^{\mathrm{lG}}, \alpha^{\mathrm{uG}} \oslash \alpha'^{\mathrm{uA}}, \alpha^{\mathrm{uG}} - b_{\max}, RT^{-\beta}\left(\beta'^{\mathrm{lA}}, \alpha^{\mathrm{uG}}\right)\right\}. \qquad (4.18)
\end{aligned}$$

The interface connections for two PE components are illustrated in Fig. 4.2a.

4.3.2 Playout Buffer

For a PB component, the relations are simpler. We only need to determine the inverse relations for the buffer constraints (4.6) and (4.7), which directly yield the following relations:

$$\begin{aligned}
\alpha^{\mathrm{uA}} &= \rho^{\mathrm{lG}} + B_{\max} - B_0, \\
\alpha^{\mathrm{lA}} &= \rho^{\mathrm{uG}} - B_0, \\
\rho^{\mathrm{uA}} &= \alpha^{\mathrm{lG}} + B_0, \\
\rho^{\mathrm{lA}} &= \alpha^{\mathrm{uG}} - (B_{\max} - B_0). \qquad (4.19)
\end{aligned}$$

The interface connections for a single PB component are illustrated in Fig. 4.2b.

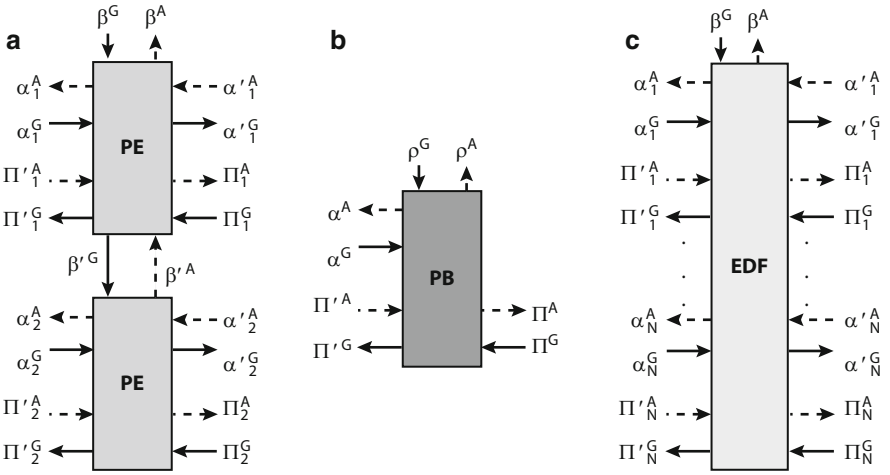

Fig. 4.2 Interface models for: (**a**) two PE components processing two streams with FP scheduling, (**b**) PB component, and (**c**) EDF component processing N streams

4.3.3 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. (4.10) and (4.11). They only need to be expressed in terms of interface variables instead of values for all streams i:

$$\alpha'^{\mathrm{uG}}_i(\Delta) = r(\alpha^{\mathrm{uG}}_i, -(D_i - \mathrm{BCET}_i), \Delta),$$

$$\alpha'^{\mathrm{lG}}_i(\Delta) = r(\alpha^{\mathrm{lG}}_i, (D_i - \mathrm{BCET}_i), \Delta),$$

using the definition of the shift function r in (4.5).

Similarly, for the resource and buffer constraints, (4.9) and (4.12), we obtain:

$$\beta^{\mathrm{lG}}(\Delta) \geq \sum_{i=1}^{N} r(\alpha^{\mathrm{uG}}_i, D_i, \Delta), \quad \forall \Delta \in \mathbb{R}^{\geq 0},$$

$$\alpha^{\mathrm{uG}}_i(D_i) \leq b_{i\,\max}, \quad \forall i.$$

Determining the input assumptions of the EDF component also involves finding the pseudo-inverse functions of the relations. Finding the input assumptions for the upper arrival curves involves inverting (4.9), (4.10), and (4.12). 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:

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

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

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

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 (4.11) or $\alpha_i^{\mathrm{lA}}(\Delta) \geq \alpha_i'^{\mathrm{lA}}(\Delta + (D_i - \mathrm{BCET}_i))$ for all i. Therefore, we can determine the following assume interface function for the lower curve of each input data stream:

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

using the shift function r as defined in (4.5).

Similarly, for the assume of the lower service curve we invert (4.9) which yields the inequality $\beta^{\mathrm{lA}}(\Delta) \geq \sum_{i=1}^{N} r(\alpha_i^{\mathrm{uG}}, D_i, \Delta)$. Therefore, the input assume for the lower service curve of an EDF component can be determined as:

$$\beta^{\mathrm{lA}}(\Delta) = \sum_{i=1}^{N} r(\alpha_i^{\mathrm{uG}}, D_i, \Delta). \tag{4.20}$$

The interface model for the EDF component is illustrated in Fig. 4.2c.

4.3.4 Worst-Case Traversal Time Interface

We develop an additional type of interface to alleviate design of systems with WCTT constraints that can span a network of components. It is an interface-based interpretation of the analytical computation of WCTTs with (4.14).

The "complies to" relation $\vdash$ for this interface connection is defined as $\Pi^{\mathrm{G}}(\Delta) \vdash \Pi^{\mathrm{A}}(\Delta) = (\forall \Delta : \Pi^{\mathrm{G}}(\Delta) \geq \Pi^{\mathrm{A}}(\Delta))$, where Π^{A} expresses the minimum service requested from all subsequent components such that the WCTT constraint

is satisfied, and Π^{G} expresses the minimum service guaranteed by all subsequent components.

Computing the guarantee for a sequence of components follows directly from (4.14) and can be done with $\Pi^{\mathrm{G}} = \beta^{\mathrm{G}}_{\mathscr{P}\mathscr{E}} \otimes \rho^{\mathrm{G}}_{\mathscr{P}\mathscr{B}} \otimes \beta^{\mathrm{G}}_{\mathscr{E}\mathscr{D}\mathscr{F}}$. Connecting a PE component to the sequence would change the combined service to $\Pi'^{\mathrm{G}} = \beta^{\mathrm{lG}} \otimes \Pi^{\mathrm{G}}$ where β^{lG} is the lower service guaranteed by the PE. Similarly, connecting a PB component we would have $\Pi'^{\mathrm{G}} = \rho^{\mathrm{lG}}_{\tau} \otimes \Pi^{\mathrm{G}}$, where $\rho^{\mathrm{l}}_{\tau}(\Delta)$ is the lower guaranteed shifted readout curve as defined with (4.8). For an EDF component, we have $\Pi'^{\mathrm{G}} = \beta^{\mathrm{lG}}_{D_i} \otimes \Pi^{\mathrm{G}}$ where $\beta^{\mathrm{lG}}_{D_i}$ is the service curve provided to the stream when processed with a relative deadline D_i as defined in (4.13).

From (4.14), we can also compute the assume on the combined service of a sequence of components as $\Pi^{\mathrm{A}} = r(\alpha^{\mathrm{uG}}, D, \Delta)$ which expresses the minimum necessary service in order to meet a WCTT constraint of D for the input α^{uG}. Propagating the assume value through a sequence of components can be done for the three types of components by inverting (4.14) as follows:

$$\mathrm{PE} : \Pi^{\mathrm{A}} = \Pi'^{\mathrm{A}} \oslash \beta^{\mathrm{lG}}, \quad \mathrm{PB} : \Pi^{\mathrm{A}} = \Pi'^{\mathrm{A}} \oslash \rho^{\mathrm{lG}}_{\tau}, \quad \mathrm{EDF} : \Pi^{\mathrm{A}} = \Pi'^{\mathrm{A}} \oslash \beta^{\mathrm{lG}}_{D_i}.$$

We can also compute component-wise constraints on the resources provided to each component given the resource assumption from preceding components Π'^{A}, and the resource guarantee from subsequent components Π^{G}. We can do this for three types of components as follows:

$$\mathrm{PE} : \beta^{\mathrm{lA}} \geq \Pi'^{\mathrm{A}} \oslash \Pi^{\mathrm{G}}, \quad \mathrm{PB} : \rho^{\mathrm{lA}}_{\tau} \geq \Pi'^{\mathrm{A}} \oslash \Pi^{\mathrm{G}}, \quad \mathrm{EDF} : \beta^{\mathrm{lA}}_{D_i} \geq \Pi'^{\mathrm{A}} \oslash \Pi^{\mathrm{G}}.$$

The above constraints can be combined with the previously computed input assumes for the resources of the three components with (4.18), (4.19), and (4.20). 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 too. The WCTT interfaces for the PE, PB, and EDF components are shown in Fig. 4.2a–c.

4.4 Illustrative Example

In this section we show how our proposed theory can be applied to an example system shown in Fig. 4.3. It represents a typical distributed embedded system for video-/signal-/media-processing. Communication is not modeled explicitly however, this can be done by adding additional components if necessary.

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

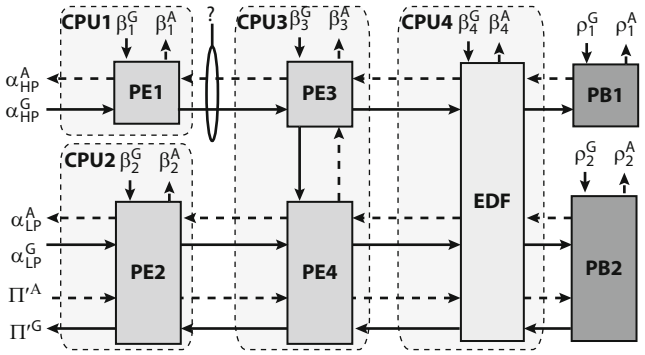

Fig. 4.3 Interface model of an example stream processing system

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. 4.3. 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 *PE*1 and *PE*2 which are mapped separately to *CPU*1 and *CPU*2, respectively. Afterwards, they are processed by components *PE*3 and *PE*4 which are mapped to *CPU*3. The tasks share the resource using FP scheduling where stream *HP* is given higher priority. Additionally, streams are processed by two tasks mapped to *CPU*4 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 *PB*1 and *PB*2. 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 $p = 25$, $j = 40$, $d = 0.1$ ms, and for *LP* stream they are $p = 25$, $j = 30$, $d = 0.1$ ms. Each data packet from the two streams has a constant processing demand of 1 cycle for all tasks. *CPU*1 is fully available with service availability of 0.3 cycles/ms. For *CPU*2, *CPU*3, and *CPU*4, the respective service availabilities are 0.3, 0.4, and 0.4 cycles/ms. Components *PE*3 and *PE*4 have internal buffer sizes of 2 and 3 packets, respectively. These buffers should never overflow. The EDF

component schedules tasks processing streams *HP* and *LP* with relative deadlines of 8 and 10 ms, respectively, with both buffers being limited to 3 packets. These buffers should also never overflow. Components *PB*1 and *PB*2 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 200 ms.

Results We consider three different scenarios of the system's parameters. In each of them, we check the compatibility of component *PE*1 with the partially designed system when all other components are already connected. Compatibility is checked by only inspecting the interface connection between *PE*1 and the system which is marked with "?" in Fig. 4.3. 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. 4.4a. It shows that the guarantee on the output stream rate $\alpha^{\mathrm{G}}_{PE1}$ expressed by *PE*1s interface is compatible with the assumption on the input rate $\alpha^{\mathrm{A}}_{PE3}$ expressed by *PE*3s interface.

Case II The WCTT constraint on the *LP* stream is decreased to 192 ms. This leads to incompatibility between components *PE*1 and *PE*3 which reveals in the interface connection as shown in Fig. 4.4b.

Case III The maximum buffer size of component *PB*2 is decreased to 5 packets which leads to incompatibility as shown in Fig. 4.4c. This reveals how constraints in component *PB*2 are propagated to multiple interfaces in the rest of the system in order to guarantee their satisfaction.

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

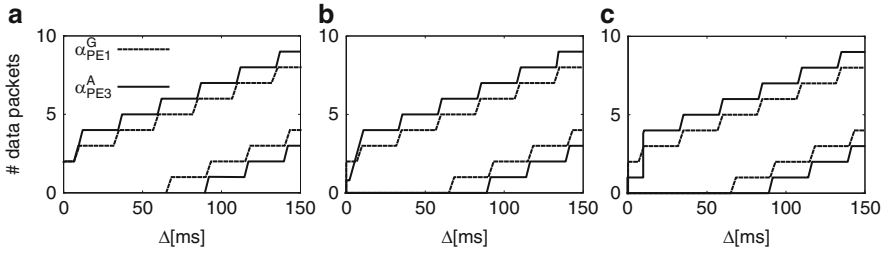

Fig. 4.4 Interface connection between the output guarantee of component $PE1$ and the input assumption of component $PE3$ shows: (**a**) compatibility, (**b**) incompatibility when WCTT for stream LP is reduced to 192 ms, and (**c**) incompatibility when buffer of component $PB2$ is decreased to 5 packets

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 chapter, we are concerned with buffer and WCTT constraints however, one can imagine developing similar interfaces for power, energy, and temperature constraints.

4.5 Concluding Remarks

In this chapter we proposed an interface algebra for checking whether multiple components of an embedded system may be composed together while satisfying their buffer and 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

The min-plus algebra convolution $\otimes$ and deconvolution $\oslash$ operators are defined as:

$$(f \otimes g)(\Delta) = \inf_{0 \le \lambda \le \Delta} \{f(\Delta - \lambda) + g(\lambda)\},$$

$$(f \oslash g)(\Delta) = \sup_{\lambda \ge 0} \{f(\Delta + \lambda) - g(\lambda)\}.$$

The duality between $\otimes$ and $\oslash$ states that: $f \oslash g \le h \iff f \le g \otimes h$.
The inverses of the $RT(\beta, \alpha)$ are defined as:

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

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

with solutions:

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

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

References

1. de Alfaro L, Henzinger TA (2001) Interface theories for component-based design. In: Proceedings of the first international workshop on embedded software, EMSOFT '01, Springer, London, pp 148–165
2. de Alfaro L, Henzinger TA (2005) Interface-based design. In: Broy M, Gruenbauer J, Harel D, Hoare C (eds) Engineering theories of software-intensive systems, NATO Science Series: Mathematics, physics, and chemistry, vol 195. Springer, Berlin, pp 83–104
3. Baruah S, Chen D, Gorinsky S, Mok A (1999) Generalized multiframe tasks. Real-Time Syst 17(1):5–22
4. Chakraborty S, Künzli S, Thiele L (2003) A general framework for analysing system properties in platform-based embedded system designs. In: Proceedings of the conference on design, automation and test in Europe, vol 1. IEEE Computer Society, Washington, DC, pp 10,190–10,195
5. Chakraborty S, Liu Y, Stoimenov N, Thiele L, Wandeler E (2006) Interface-based rate analysis of embedded systems. In: Proceedings of the 27th IEEE international real-time systems symposium, RTSS '06, IEEE Computer Society, Washington, DC, pp 25–34
6. Le Boudec JY, Thiran P (2001) Network calculus: A theory of deterministic queuing systems for the internet, LNCS, vol 2050. Springer, Berlin
7. Maxiaguine A, Künzli S, Thiele L (2004) Workload characterization model for tasks with variable execution demand. In: Proceedings of the conference on design, automation and test in Europe, vol 2. IEEE Computer Society, Washington, DC, pp 21,040–21,045
8. Thiele L, Chakraborty S, Naedele M (2000) Real-time calculus for scheduling hard real-time systems. In: Circuits and Systems, 2000. Proceedings of the 2000 IEEE international symposium on ISCAS 2000 Geneva, vol 4, pp 101–104 (2000)
9. Thiele L, Wandeler E, Stoimenov N (2006) Real-time interfaces for composing real-time systems. In: Proceedings of the 6th ACM & IEEE international conference on embedded software, EMSOFT '06, ACM, New York, pp 34–43
10. Wandeler E, Thiele L (2006a) Interface-based design of real-time systems with hierarchical scheduling. In: Proceedings of the 12th IEEE real-time and embedded technology and applications symposium, RTAS '06, IEEE Computer Society, Washington, DC, pp 243–252
11. Wandeler E, Thiele L (2006b) Optimal TDMA time slot and cycle length allocation for hard real-time systems. In: Proceedings of the 2006 Asia and South Pacific design automation conference, ASP-DAC '06, IEEE Press, Piscataway, NJ, pp 479–484
12. Wandeler E, Maxiaguine A, Thiele L (2006) Performance analysis of greedy shapers in real-time systems. In: Proceedings of the conference on design, automation and test in Europe: Proceedings of the European Design and Automation Association, 3001 Leuven, Belgium, pp 444–449