Cyclic Dependencies in Modular Performance Analysis

Bengt Jonsson  
Dept. Information Technology  
Uppsala University  
Sweden  
bengt@it.uu.se

Simon Perathoner  
Computer Engineering and Networks Lab.  
ETH Zurich, Switzerland  
perathoner@tik.ee.ethz.ch

Lothar Thiele  
Computer Engineering and Networks Lab.  
ETH Zurich, Switzerland  
thiele@tik.ee.ethz.ch

Wang Yi  
Dept. Information Technology  
Uppsala University  
Sweden  
iy@it.uu.se

ABSTRACT

The Modular Performance Analysis based on Real-Time Calculus (MPA-RTC), developed by Thiele et al., is an abstraction for the analysis of component-based real-time systems. The formalism uses an abstract stream model to characterize both workload and availability of computation and communication resources. Components can then be viewed as stream transformers. The Real-Time Calculus has been used successfully on systems where dependencies between components, via either workload or resource streams, are acyclic. For systems with cyclic dependencies the foundations and performance of the formalism are less well understood.

In this paper, we develop a general operational semantics underlying the Real-Time Calculus, and use this to show that the behavior of systems with cyclic dependencies can be analyzed by fixpoint iterations. We characterize conditions under which such iterations give safe results, and also show how precise the results can be.

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

General Terms
Performance, Design, Theory

Keywords
Performance Analysis, Real-Time Calculus, Fixpoint Iteration

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

EMSOFT ’08, October 19–24, 2008, Atlanta, Georgia, USA.  
Copyright 2008 ACM X-XXXXX-XX-X/XX/XX ...$5.00.

1. INTRODUCTION

Complex embedded systems very often consist of parallel or distributed computing elements that are exchanging data via some communication system. In addition, the applications that are running on these platforms can be described as communicating processes. In the case of real-time requirements, it is necessary to investigate the interaction of the processes and the communication tasks with the resources they are using. Because of the interference caused by the concurrent use of resources the evaluation of the timing behavior of the whole embedded system is a challenging task.

One approach to handling the associated computational complexity is to adopt a component-based approach where the verification and validation can be done in an incremental process. Based on appropriate abstractions, the interaction of each individual component with its environment and the available resources is analyzed and abstracted into a corresponding interface representation. The overall system behavior is then determined by an appropriate composition mechanism.

For example this approach can be successfully applied in a situation with static scheduling, in which each task is allocated pre-determined timing slots of CPU time. This approach, however, suffers problems of inflexibility. The paradigm of static fixed-priority scheduling offers more flexibility. In this case, the classical schedulability analysis uses maximal blocking time as a measure of available computation resources. In its standard form, however, it has problems to handle variabilities in task parameters, e.g., highly varying computation times and jitter.

The approach of Real-Time Calculus [9] provides a framework for a compositional analysis. Composition is possible in terms of processes (functional composition), scheduling policies (interaction) and resources (distributed operation). The underlying abstraction is able to handle complex event and resource patterns and therefore, can handle situations with a high degree of non-determinism. The approach characterizes each component by a time-invariant transfer function, which operates on upper and lower bounds on the number of events and computation resources available in all time intervals. Despite of the fact that the approach has been proven to be useful in case studies and investigations on benchmark applications, fundamental issues in linking the abstractions to an operational approach are not yet investigated.
A particular problem of great practical relevance is the handling of cyclic networks of components. The natural approach would be to compute the system characteristics captured in the Real-Time Calculus by means of a fixpoint computation, starting from some initial approximation. However, it is not known to what extent the resulting fixpoint is faithful to an underlying operational behavior of the system, or how to best choose initial approximations.

In this paper, we propose a simple operational model of distributed systems of processes and relate it to characteristics in the Real-Time Calculus. On this basis, we prove central properties about the faithfulness of fixpoints computed using the abstractions in Real-Time Calculus. The main result is a method that leads to the optimal fixpoint, i.e., makes best usage of the underlying abstraction. In addition, the approach presented in this paper is not restricted to Real-Time Calculus. The results can easily be transferred to related abstractions of streams, resources, and their interaction.

The paper is organized as follows. In the next section, we introduce the basics of the framework for Modular Performance Analysis with Real-Time Calculus, accompanied by a motivating simple example. In Section 3, we present a general operational model of components and systems, and of specifications in some formalism for analyzing resource and timing properties, and prove correctness results for fixpoints. In Section 4, we specialize these results to the Real-Time Calculus, and discuss two techniques for obtaining initial approximations for iterations. Section 5 contains a simple example to illustrate sensitivity of fixpoints on parameters in boundary cases. Thereafter, we show the practical usefulness of our models on a simple case study and present concluding remarks.

**Related Work.** Jersak et al. [2] have considered a special case of cyclic dependencies in their periodic-with-jitter event model, in which there are functional cycles of events for task activation. The approach is limited in terms of the underlying abstraction and only informally makes statements about convergence properties. In particular, no results on the dependence of the fixpoint on initial conditions are provided.

The general problem of analyzing cycles in the RTC has been considered by Schioel et al. [5]. They consider a similar framework as in our paper under the constraint that the time domain starts at 0. Hence they must use weaker equations than usually considered in the RTC (e.g., [9]) in order to establish guaranteed sound results. Their treatment relies on several assumptions. For instance, the result about the correctness of any fixpoint of the RTC equations ignores the potential problem of zero-delay cycles. These zero-delay cycles can occur if a continuous flow model is used for the system modeling. In addition, further properties of the obtained fixpoint solutions are assumed, e.g., “non-blocking”. The present paper gives a comprehensive and more detailed treatment of correctness of fixpoints. We also state the restrictions of our models on a simple case study and present concluding remarks.

2. **REAL-TIME CALCULUS**

In this section, we describe the framework of Modular Performance Analysis based on Real-Time Calculus. In order to introduce the problem, we also describe a simple system architecture for which the performance analysis is complicated by a cyclic dependency.

MPA-RTC is a compositional abstraction for the modeling and analysis of distributed real-time systems. The formalism has its roots in Network Calculus [4] developed to reason about timing properties of event and resource streams that flow through a network of computation and communication components. The MPA-RTC abstraction has shown to provide valuable analysis results in several industrial case studies [6, 10].

There are slightly different treatments of Real-Time Calculus, depending on whether the time domain has an initial point (taken to be 0) or whether it goes unboundedly into the past. The presentation in this section will cover both cases.

2.1 Event Stream Model

A stream of events can be characterized by a differential arrival function $R : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{N}$, where $R(s, t)$ for $s \leq t$ denotes the sum of events that arrive in the time interval $[s, t)$ (including $s$ but not $t$), with $R(s, s) = 0$. If the time domain has an initial point 0, then of course $0 \leq s$.

The Real-Time Calculus abstracts from concrete execution traces described by arrival functions $R$ and bounds all possible traces of an event stream with a pair of arrival curves $\alpha(\Delta) = [\alpha^l(\Delta), \alpha^u(\Delta)]$ where $\alpha^l(\Delta)$ denotes the lower arrival curve and $\alpha^u(\Delta)$ the upper arrival curve of the event stream. Informally, a lower arrival curve $\alpha^l : \mathbb{R} \rightarrow \mathbb{R} \rightarrow \mathbb{N}$ is a monotone superadditive function with $\alpha^l(0) = 0$, which states that in any half-open time interval of length $\Delta$ at least $\alpha^l(\Delta)$ events will arrive. An upper arrival curve $\alpha^u : \mathbb{R} \rightarrow \mathbb{R} \rightarrow \mathbb{N}$ is a monotone subadditive function, with $\alpha^u(0) = 0$, which states that in any half-open time interval of length $\Delta$ at most $\alpha^u(\Delta)$ events will arrive.

We introduce the notation $R \models [\alpha^l, \alpha^u]$ to denote that $\alpha^l(\Delta) \leq R(s, s + \Delta) \leq \alpha^u(\Delta)$ for all $s \in \mathbb{R}$ and $\Delta \in \mathbb{R} \rightarrow \mathbb{R}$. For a given event stream described by an arrival function $R$, the tightest arrival curves $\alpha_R^l, \alpha_R^u$ that model the stream are given by

$$
\alpha_R^l(\Delta) = \inf_{s \in \mathbb{R}} R(s, s + \Delta)
$$

$$
\alpha_R^u(\Delta) = \sup_{s \in \mathbb{R}} R(s, s + \Delta)
$$

(1) 

2.2 Resource Model

In analogy with differential arrival functions, the availability of a computation or communication resource is represented by a resource stream, which can be described by a differential service function $C : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R} \rightarrow \mathbb{R}$, where $C(s, t)$ denotes the sum of available resource units in the time interval $[s, t)$ with $C(s, s) = 0$. Continuing the analogy, the Real-Time Calculus abstracts from concrete service func-
tions $C$ and uses a pair of lower and upper service curves $\beta(\Delta) = [\beta^l(\Delta), \beta^u(\Delta)]$ to model the availability of a resource. Informally, a lower service curve $\beta^l: \mathbb{R}^{\geq 0} \rightarrow \mathbb{R}^{\geq 0}$ is a monotone superadditive function with $\beta^l(0) = 0$, which gives a lower bound $\beta^l(\Delta)$ on the number of available computation units in any interval of length $\Delta$. An upper service curve $\beta^u: \mathbb{R}^{\geq 0} \rightarrow \mathbb{R}^{\geq 0}$ is a monotone subadditive function with $\beta^u(0) = 0$, which gives an upper bound $\beta^u(\Delta)$ on the number of available resource units in any interval of length $\Delta$. Again we use the notation $C = [\beta^l, \beta^u]$ to denote that $\beta^l(\Delta) \leq C[s, s+\Delta] \leq \beta^u(\Delta)$ for all $s \in \mathbb{R}$ and $\Delta \in \mathbb{R}^{\geq 0}$. For a given concrete resource availability described by a service function $C$, the tightest service curves $\beta^l, \beta^u$ that model the resource availability are given by

$$
\begin{align*}
\beta^l(\Delta) &= \inf_{s \in \mathbb{R}} C[s, s + \Delta] \\
\beta^u(\Delta) &= \sup_{s \in \mathbb{R}} C[s, s + \Delta]
\end{align*}
$$

(2)

### 2.3 Component Model

Components are the basic building blocks of a system. They are the implementation of tasks that process event streams and run on shared resources, e.g., a computing or communication subsystem. An external view of a component is that it receives a stream of input events, and a stream of available resources, from which it produces a stream of outgoing events and a stream of remaining resources. We assume that component behavior is deterministic. The incoming and outgoing event streams can be modeled by differential arrival functions, $R$ and $R'$, respectively, and the available and remaining resource by differential service functions, $C$ and $C'$, respectively. This view is illustrated in Figure 1(a).

![Concrete and abstract component](image)

*(a) Component model: $R \xrightarrow{\varphi} R'$ and $C \xrightarrow{\alpha, \beta} C'$.*

In the figure, we use $\varphi$ to denote the transformation which represents the behavior of the component. We can view $\varphi$ as a transfer function from input to output, i.e., $R'(C') = \varphi(R, C)$. In the MPA-RTC framework such a concrete component is modeled by an abstract component in the domain of arrival and service curves. When doing so, we use an arrival curve $\alpha$ and a service curve $\beta$ to denote constants on $R$ and $C$. For any $R$ and $C$ with $R \equiv \alpha$ and $C \equiv \beta$, the component produces outputs $R'$ and $C'$ with $R' \equiv \alpha'$ and $C' \equiv \beta'$. The abstract component is illustrated in Figure 1(b).

![Abstract component](image)

*(b) Abstract component: $\alpha \rightarrow \alpha'$ and $\beta \rightarrow \beta'$.*

In the figure, we use a function $\Phi$ to denote the transformation which represents the abstract component. This function provides, for each input arrival and service curves $\alpha, \beta$, the output arrival and service curves $\alpha', \beta'$. The function $\Phi$ is different for different types of components with different behaviors.

A typical example for an abstract component in the context of the MPA-RTC modeling framework is a so-called Greedy Processing Component (GPC). It models a task that

is triggered by the events of the incoming event stream which queue up in a FIFO buffer. The task processes the events in a greedy fashion, while being restricted by the availability of resources. The processing is performed within some specified execution time. For simplicity, we assume that it takes exactly one computational unit to process one event. This assumption can be relaxed by appropriate rescalings, which we will not consider further.

For a GPC component, the following internal relations between the incoming arrival curves $[\alpha^l, \alpha^u]$ and service curves $[\beta^l, \beta^u]$ and the corresponding outgoing arrival curves $[\alpha'^l, \alpha'^u]$ and service curves $[\beta'^l, \beta'^u]$ are stated and proved in [8]:

$$
\begin{align*}
\alpha'^u &= \min \{(\alpha^u \otimes \beta^u) \otimes \beta^l, \beta^u\} \\
\alpha'^l &= \min \{(\alpha^l \otimes \beta^u) \otimes \beta^l, \beta^l\} \\
\beta'^u(\Delta) &= \max\{\inf_{0 \leq \lambda \leq \Delta}(\beta^u(\lambda) - \alpha^l(\lambda)), 0\} \\
\beta'^l(\Delta) &= \sup_{0 \leq \lambda \leq \Delta}(\beta^l(\lambda) - \alpha^u(\lambda))
\end{align*}
$$

An important observation here is that these relations are correct for RTC with unbounded past (under the assumption of bounded buffer occupancy), but they are not correct for RTC with an initial time point. In particular, the equations for $\beta'^u$ and $\alpha'^u$ are too tight. In this case, the relations derived in [7] must be used: $\alpha' = \alpha^l \otimes \beta^l$ and $\beta'^u(\Delta) = \sup_{0 \leq \lambda \leq \Delta}(\beta^u(\lambda) - \alpha^l(\lambda))$.

### 2.4 System Performance Model

In order to analyze the performance of a distributed system, we need to build a system performance model. This model has to reflect the flow of data in the system and also needs to represent its hardware architecture. In particular, it has to model the mapping of tasks to computation or communication resources, as well as the scheduling policies adopted on shared resources.

In the MPA-RTC framework such a performance model is constructed by composing abstract components to a network. The performance model depicted in Figure 2 shows a simple example of component composition. It represents a system consisting of two tasks $T_1$ and $T_2$ which serially process an incoming event stream, and share a processing resource according to a preemptive fixed priority policy, with task $T_2$ having higher priority than task $T_1$. Task $T_1$ is triggered by input events from the environment. It produces output events which trigger task $T_2$. Task $T_2$ outputs events to the environment. In the performance model we represent the two tasks with two abstract Greedy Processing Components. We reflect the dataflow described above by connecting the abstract event stream output of component $T_1$ with the abstract event stream input of component $T_2$. Similarly, we connect the abstract resource stream output of component $T_2$ with the abstract resource stream input of component $T_1$ in order to model the scheduling policy for the shared processing resource; task $T_2$ has full access to the processor, while task $T_1$ receives only the processing resources left over by task $T_2$.

In general, scheduling policies for shared resources are modeled by the way the abstract resources $\beta$ are distributed among the different abstract components. For some scheduling policies, such as EDF or TDMA abstract components with appropriate transfer functions have been introduced.
2.5 A Simple Example

We use the system represented in Figure 2 as simple motivating example for our work, since it contains a cyclic dependency which inhibits the modular performance analysis.

![Figure 2: Performance model for example system](image)

For simplicity, we assume that both T1 and T2 need exactly 1 unit of resources to service an event. Further we assume that the incoming stream of events from the environment to T1 is constrained by the arrival curves \([\alpha^1_t, \alpha^1_u]\), and that the incoming stream of resources to T2 is constrained by the service curves \([\beta^2_t, \beta^2_u]\). We do not consider any overhead for the context switching between the two tasks.

We want to compute bounds for the event streams from T1 to T2 and the resource stream from T2 to T1. However, the characterization of this streams is not straightforward since there is a cyclic dependency between them (T1 triggers T2 while T2 preempts T1). Let us denote a particular characterization of these streams by a quadruple \((\alpha^1, \alpha^u, \beta^2, \beta^u)\). The transfer functions in equation (3) imply the following relationships between the curves (assuming unbounded past):

\[
\begin{align*}
\alpha^u &= \min \{ (\alpha^1 \otimes \beta^u) \otimes \beta^2, \beta^u \} \\
\alpha^1 &= \min \{ (\alpha^1 \otimes \beta^u) \otimes \beta^2, \beta^u \} \\
\beta^u(\Delta) &= \max \{ \inf_{\lambda \leq \Delta} (\beta^2(\lambda) - \alpha^1(\lambda)), 0 \} \\
\beta^2(\Delta) &= \sup_{0 \leq \lambda \leq \Delta} (\beta^2(\lambda) - \alpha^u(\lambda))
\end{align*}
\]

Let us denote the quantities \((\alpha^1_t, \alpha^1_u, \beta^2_t, \beta^2_u)\) by \(\Sigma_t\), and the quantities \((\alpha^1, \alpha^u, \beta^2, \beta^u)\) by \(\Sigma_h\); let us ignore the specification of the output streams. We denote the pair \((\Sigma_t, \Sigma_h)\) with \(\Sigma\) and we use \(\Psi\) to represent the mapping from one specification \(\Sigma\) to another \(\Sigma^*\) by means of the equations (4).

It is then natural to expect that a specification of the behavior of the system can be obtained as a fixpoint of \(\Psi\), i.e. as a solution of the equation \(\Sigma = \Psi(\Sigma)\). A natural way to compute such a fixpoint is to start from some initial approximation, i.e. by starting from some tuple \(\Sigma_0\), and computing the sequence \(\Sigma_0, \Sigma_1, \Sigma_2, \ldots\) where \(\Sigma_{k+1} = \Psi(\Sigma_k)\) in the hope that the sequence will converge to a limit \(\Sigma^*\).

However, the correctness of such a fixpoint iteration in the context of MPA-RTC has not been formally justified so far. In particular several questions need to be answered:

- Will the iteration always converge to a limit \(\Sigma^*\)?
- If so, how does \(\Sigma^*\) correspond to the behavior of the system?
- How to choose the initial approximation \(\Sigma^0\)?

To illustrate that fixpoints are not in general unique, consider again the above system. Let \(\beta^2(\Delta) = \beta^2(\Delta) = \Delta\), i.e., giving T2 full access to the resource, and let an event arrive every second time unit, i.e.,

\[
\alpha^1(\Delta) = \left\lfloor \frac{\Delta}{2} \right\rfloor \quad \text{and} \quad \alpha^2(\Delta) = \left\lfloor \frac{\Delta}{2} \right\rfloor.
\]

The optimal (and correct) fixpoint is the one which forces the same behavior on the internal event stream as on the input event stream, i.e., \([\alpha^1, \alpha^u] = [\alpha^1_t, \alpha^1_u]\). However, a much worse fixpoint is one which gives no information at all, given by a specification of the event stream \([\alpha^1, \alpha^u]\) such that

\[
\alpha^1(\Delta) = 0 \quad \text{and} \quad \alpha^u(\Delta) = \left\lfloor \frac{\Delta}{2} \right\rfloor.
\]

and a specification of the resource stream \([\beta^2, \beta^u]\) such that

\[
\beta^2(\Delta) = 0 \quad \text{and} \quad \beta^u(\Delta) = \Delta.
\]

To investigate these questions, in the next section we provide a more general framework for specifying quantitative properties in component-based systems, in which we prove results about the above questions under certain assumptions.

3. STREAMS AND FIXPOINTS

In this section, we prove results about correctness of fixpoint calculations, which are valid for a class of formalisms that specify quantitative properties of component-based systems, such as MPA or Symta/S, see [2]. We provide an abstract description of such a formalism, and thereafter define conditions under which fixpoints are correct. We here give a treatment which considers both the case where system behavior starts at an initial time point, taken to be 0, and when it has no initial time point, i.e., the time domain is the set \(\mathbb{R}\) of all real numbers.

Streams. We assume that the behavior of a system or a component is observed at a set of streams, which are the observable connections between components and between components and the environment. A trace on a set \(V\) of streams is a function \(\sigma : V \rightarrow (\mathbb{R} \times \mathbb{R})^{\mathbb{Z}^+}\), which for each stream \(v \in V\) provides a function \(\sigma(v)\) from time-intervals to observations. We take \(\mathbb{R}^{\mathbb{Z}^+}\) as the range of observations, since we intend to model accumulation of some resource or observable. For instance \(\sigma(v)(s,t)\) could denote the number of events that have passed or the amount of CPU resources that have been available in the interval \([s,t)\). We assume \(\sigma(v)(s,t) + \sigma(v)(t,u) = \sigma(v)(s,u)\) for \(s \leq t \leq u\). If time starts at 0, we assume \(0 \leq s \leq u\). We let \(\overline{T}(V)\) denote the set of traces on a set \(V\) of streams. The restriction of a trace \(\sigma\) on \(V\) to a subset \(V' \subseteq V\) is denoted \(\sigma|_{V'}\).

Component Behavior. A component is equipped with a set of input streams \(V_I\) and a set \(V_O\) of output streams. We let its behavior be characterized by a behavior mapping \(\varphi : \overline{T}(V_I) \rightarrow \overline{T}(V_O)\), which maps any input trace \(\sigma_I\) on the input streams \(V_I\) to an output trace \(\varphi(\sigma_I)\) on the output streams \(V_O\).
System Behavior: A system of components has a set \( V_I \) of external input streams, and a set \( V_O \) of internal streams and external output streams. Each stream in \( V_O \) is an output stream of some components. Some output streams go to other components (they can be thought of as internal), and some to the environment. By putting together all component behavior transformers, we get a system behavior transformer \( \psi : \text{Tr}(V_I \cup V_O) \rightarrow \text{Tr}(V_I \cup V_O) \) which maps any trace on the streams of the system to another trace on system streams. The mapping \( \psi \) will preserve the traces on input streams, and will generate traces on internal and output streams according to the behavior mappings of components.

When forming a system from deterministic components, it is reasonable to expect that any trace \( \sigma_I \) on input streams \( V_I \) induces a unique trace on the output streams. To establish this formally, some form of causality between inputs and outputs of the system must be assumed. We will define a property, called simulatability, which roughly means that the system has no zero-delay cycles. We first consider the case when time starts at 0, thereafter indicate how to adapt the results to an unbounded past.

Let a time vector on a set \( V \) be a mapping \( \tau : V \mapsto \mathbb{R} \) from streams in \( V \) to time values. For a value \( t \in \mathbb{R} \), let \( \overline{t} \) denote the time vector which maps all streams in its domain to \( t \). For two time vectors, \( \tau' \) on \( V \), let \( \tau \leq \tau' \) denote that \( \tau(v) \leq \tau'(v) \) for all \( v \in V \). For a time vector \( \tau \) on \( V \) and set of streams \( V' \subseteq V \), we say that two traces \( \sigma \) and \( \sigma' \) on \( V' \) agree up to \( \tau \), denoted \( \sigma \simeq \tau, \sigma' \), if, \( \forall (v,s,t) \in \sigma'(v)(s,t) \) whenever \( s \leq t \leq \tau(v) \) for all \( v \in V \) and \( \forall (v,s,t) \in \sigma(v)(s,t) \) for all \( v \in \overline{V}' \). Intuitively, time vectors will be used to denote how much of a system trace has been constructed in a simulation. A time vector need not map all streams to the same time value, since it may be possible to know the state of some streams at once, which agrees with \( \sigma' \) up to \( \tau \).

Definition 1. A system, represented by behavior transformer \( \psi \), is simulatable if for each input trace \( \sigma_I \in \text{Tr}(V_I) \) there is an increasing sequence \( \tau_0, \tau_1, \ldots \) of time vectors on \( V_O \), called the simulation sequence for \( \sigma_I \), with \( \lim \tau_i = \overline{0} \) and \( \lim \tau_i(v) = \infty \) for all \( v \in V_O \), such that for all traces \( \sigma \), \( \sigma' \in \text{Tr}(V_I \cup V_O) \) with \( \sigma(v) = \sigma'(v) \); and \( \forall i \geq 0 \) : \( \sigma \simeq \tau_i, \sigma' \) implies \( \psi(\sigma) \simeq \tau_{i+1}, \psi(\sigma') \).

Note that \( \sigma \simeq \tau_i, \sigma' \) implies that \( \sigma \) and \( \sigma' \) agree on \( V_I \). Intuitively, a system is simulatable if one can advance time vectors stepwise, at each step compute new outputs from previously known inputs for some longer time, and such that the whole trace can be obtained in \( \omega \) steps.

For a simulatable system, with behavior transformer \( \psi \), with given input trace \( \sigma_I \), we can construct the resulting system trace as the limit of a sequence of traces, as follows.

Let \( \tau_0, \tau_1, \tau_2, \ldots \) be the simulation sequence for \( \sigma_I \). Define the sequence of traces \( \sigma^0, \sigma^1, \sigma^2, \ldots \), where \( \sigma^i \) is any trace with \( \sigma^i(v) = \sigma_I(v) \) and \( |\sigma^i+1| = |\psi(\sigma^i)| \). By simulatability, the sequence \( \sigma^0, \sigma^1, \sigma^2, \ldots \) converges, since \( \sigma^i \simeq \tau_i, \sigma^j \) for any \( i, j \geq 0 \). The limit \( \sigma \) which we can denote by \( \psi(\sigma_I) \), is defined to be the actual trace on input \( \sigma_I \).

Let us consider how to adapt this to the situation when time goes unboundedly into the past. Then there is no initial time point for defining traces. For a time vector \( \tau \), say that \( \sigma^0 \) is a possible system trace up to \( \tau \) if \( \sigma^0 \simeq \tau, \psi(\sigma^0) \). Let \( \tau \) be an arbitrary real-valued time vector on \( V_O \). We say that the system is simulatable from \( \tau \) if whenever \( \sigma^0 \) is a possible system trace up to \( \tau \) with \( \sigma^0(v) = \sigma_I(v) \), there is an increasing sequence \( \tau_0, \tau_1, \ldots \) of time vectors on \( V_O \), starting with \( \tau_0(\tau) = \tau \), called the simulation sequence for \( \sigma^0 \) from \( \tau \), with \( \lim \tau_i(\tau) = \infty \) for all \( v \in V_O \), such that for all traces \( \sigma, \sigma' \in \text{Tr}(V_I \cup V_O) \) with \( \sigma(v_I) = \sigma'(v_I) \); we have that \( \sigma \simeq \tau_i, \sigma' \) implies \( \psi(\sigma) \simeq \tau_{i+1}, \psi(\sigma') \) for all \( i \geq 0 \). We say that the system is simulatable if it is simulatable from any real-valued time vector.

If the system is now simulatable from \( \tau \), we can construct an actual system trace \( \psi(\sigma^0) \) in the same way as above, starting from \( \sigma^0 \). We say that \( \psi(\sigma^0) \) is the (uniquely defined) actual system trace on input \( \sigma_I \), which agrees with \( \sigma^0 \) up to \( \tau \).

Component Specifications. Let us now consider specifications of traces and behaviors. Abstractly, a specification \( \Sigma \) on a set \( V \) of streams is a mapping \( V \mapsto 2^{(\mathbb{R} \times \mathbb{R})} \), which maps each stream \( v \in V \) to a set \( \Sigma(v) \) of traces on stream \( v \) (of course, it should satisfy some properties of being “well-behaved”). As one of many possibilities, a specification can prescribe bounds on the number of elements transmitted over a stream in certain time intervals. For a set \( V \) of streams, let \( \text{Spec}(V) \) denote the set of specifications on streams \( V \). If both \( \sigma \) and \( \Sigma \) use the set \( V \) of streams, we use \( \sigma \models \Sigma \) to denote \( \forall v \in V, \sigma(v) \in \Sigma(v) \).

A component with input streams \( V_I \) and output streams \( V_O \), characterized by the behavior mapping \( \varphi \), can be specified by a specification mapping \( \Phi : \text{Spec}(V_I) \rightarrow \text{Spec}(V_O) \) from specifications on the set \( V_I \) of input streams to specifications on the set \( V_O \) of output streams. This mapping should be correct wrp. to \( \varphi \), i.e., have the property that whenever \( \sigma_I(v) = \Sigma(v_I) \) for all input streams \( v_I \in V_I \), then \( \varphi(\sigma_I)(v_O) = \Phi(\Sigma)(v_O) \) for all output streams \( v_O \in V_O \).

System Specifications. Just as for system behaviors, we can put together all component specification mappings into a system specification transformer \( \Psi : \text{Spec}(V_I \cup V_O) \rightarrow \text{Spec}(V_I \cup V_O) \). From correctness of component specification mappings, it follows that \( \Psi \) is correct wrp. to the system behavior transformer \( \varphi \), i.e., whenever \( \sigma \models \Sigma \), then \( \varphi(\sigma) \models \Phi(\Sigma) \). That is, \( \forall v \in V, \sigma(v) \subseteq \Sigma(v) \Rightarrow \varphi(\sigma)(v) \subseteq \Phi(\Sigma)(v) \).

We now let \( \Psi \) be a system specification transformer which is correct wrp. to \( \varphi \). We assume a specification \( \Sigma_I \) on \( V_I \) and a trace \( \sigma_I \) on \( V_I \) with...
σ₁ \models Σ₁. Let τ₀, τ₁, τ₂, \ldots be the simulation sequence for σ₁, and let ψ(σ₁) be the actual system trace on input σ₁.

We shall assume that for an actual system trace σ with Σ₁\vDash Σ₁, there is a strongest specification Σ which agrees with Σ₁ on V₁, such that σ \models Σ. We denote this specification by Ψ. In MPA-RTC, this strongest specification can be obtained from Equations (1) and (2).

**Theorem 1.** If the specification Σ⁺ is satisfiable and also agrees with Ψ on V₁, then

\[ ψ(σ₁) \models \preceq, Ψ(Σ⁺) \]

for all \( i \geq 0 \).

Proof. Let τ₀, τ₁, τ₂, \ldots be the simulation sequence for σ₁. Let ψ(σ₁) be the actual system trace given σ₁, obtained as the limit of the sequence \( \sigma^0, \sigma^1, \sigma^2, \ldots \), where \( \sigma^i \) is any trace with \( \sigma^i|V₁ = σ₁ \), and where \( \sigma^{i+1} = ψ(σ^i) \). We shall prove by induction over \( i \) that \( \sigma^i \models \preceq, Ψ(Σ⁺) \). The base case \( \sigma^0 \models \preceq, Σ⁺ \) follows from the assumption that \( σ₀|V₁ = σ₁ \) and that \( Σ⁺ \) agrees with Σ₁ on V₁ and is satisfiable. For the inductive step, assume \( \sigma^i \models \preceq, Ψ(Σ⁺) \), i.e., that there is a trace \( σ^i \) with \( σ^i \models \preceq, Ψ(Σ⁺) \). By correctness of Ψ with respect to ψ, we get that \( ψ(σ^i) \models \preceq, Ψ(Σ⁺) \). Since the system represented by ψ is simulatable, we get \( ψ(σ^i) \preceq, ψ(σ') \). Hence \( ψ(σ') \models \preceq, Ψ(Σ⁺) \). The conclusion of the theorem follows by noting that \( ψ(σ_i) \preceq, σ' \) for all \( i \).

**Convergence.** Theorem 1 does not say anything about convergence of the sequence \( Σ⁺₀, Ψ(Σ⁺₀), Ψ²(Σ⁺₀), \ldots \). Let us formalize conditions under which this can be attained.

For a set \( V \) of streams, the relation \( \models \) introduces a natural partial order \( \preceq \) on the set \( \text{Spec}(V) \) of specifications over \( V \), by defining \( Σ \preceq Σ' \) iff \( σ \models Σ \) implies \( σ \models Σ' \) for any \( σ \in \text{Tr}(V) \). We shall assume that \( \text{Spec}(V) \) with the partial order constitutes a cpo, i.e., that any chain \( Σ₀ \preceq Σ₁ \preceq Σ₂ \preceq \cdots \) has a least upper bound \( \sqcup \Sigmaₙ \). We finally assume that any specification Σ is a safety property, which means that if \( ∀τ, σ \preceq Σ, τ \models Σ \), then also \( σ \models Σ \).

We shall assume that for any trace \( σ \in \text{Tr}(V) \) there is a least (strongest) specification which is satisfied by \( σ \).

A specification mapping Φ is monotone if Σ \preceq Σ' implies Φ(Σ) \preceq Φ(Σ'). It is continuous if

\[ Φ(\sqcup \Sigmaₙ) = \sqcup \Phi(Σₙ) \]

for any chain \( Σ₀ \preceq Σ₁ \preceq Σ₂ \).

We can now strengthen Theorem 1, so that it also guarantees convergence.

**Theorem 2.** Assume, in addition to previous assumptions, that Ψ is monotone and continuous. Then, among all specifications which agree with Σ₁ on V₁, and are satisfied by at least one actual system trace σ with \( σ|V₁ \models Σ₁ \), Ψ has a unique smallest fixpoint Σ⁺ which is satisfied by all such traces. Furthermore Σ⁺ can be obtained as the limit of the sequence \( Σ₀, Ψ(Σ₀), Ψ²(Σ₀), \ldots \) of approximations if \( Σ₀ \) is a specifications which agree with Σ₁ on V₁, and is satisfied by at least one actual system trace σ with \( σ|V₁ \models Σ₁ \), and is such that \( Σ₀ \preceq Σ⁺ \).

Proof. We continue using the previously introduced notation. Let σ be some trace of the system such that \( σ|V₁ \models Σ₁ \). Let Ψ be the strongest specification, which agrees with Σ₁ on V₁, such that \( σ \models Ψ \). By correctness of Ψ we have Σ₁ \preceq Ψ(Σ₁). By monotonicity of Ψ we then get Ψⁿ(Σ₁) \models Ψⁿ⁺(Ψ(Σ₁)) for all \( n \geq 0 \). This means that the sequence \( Ψ(Σ₁), Ψ²(Σ₁), Ψ³(Σ₁), \ldots \) converges to a fixpoint Σ⁺.

By Theorem 1, and the assumption that all properties are safety properties, \( σ^0 \models Σ⁺ \) for any trace \( σ^0 \) of the system such that \( σ^0|V₁ \models Σ₁ \).

Since \( σ^0 \) was an arbitrary initial trace, we get the same fixpoint Σ⁺ if we repeat the above procedure with any other trace \( σ' \) with \( σ'|V₁ \models Σ₁ \), and repeat the above procedure.

We have proven the theorem for initial approximations of form \( Σ₁ \) for some system trace. It remains to consider the case where \( Σ₁ \preceq Σ₂ \preceq Σ₃ \preceq \cdots \) for some Σ. The theorem then follows by noting that by monotonicity Ψⁿ⁺(Σ₁) \models Ψⁿ⁺(Σ₂) \models Σ⁺ for all \( n \geq 0 \), implying that also Σ⁺, Ψ(Σ⁺), Ψ²(Σ⁺), \ldots converges to Σ⁺.

**The case where time starts at -∞.** Let us indicate how the results in this section work out when time has no initial point, and how Theorems 1 and 2 adapt.

We use the notation introduced previously. Let τ be a timevector. Let \( σ₀ \) be a possible system trace up to τ with \( σ₀|V₁ \models Σ₁ \), and let τ₀, τ₁, τ₂, \ldots be the simulation sequence for \( σ₀ \) from τ. Define \( \text{Cont}(σ₀, τ) \) as the set of traces σ such that \( σ \models σ₀ \). We can now derive the following two theorems.

**Theorem 3.** If the specification Σ₀ is satisfied by \( σ₀ \), then

\[ ψ(σ₀) \models \preceq, Ψ(Σ₀) \]

for all \( i \geq 0 \).

Proof. Analogous to that for Theorem 1.

**Theorem 4.** Assume that Ψ is monotone and continuous. Then, among all specifications which agree with Σ₁ on V₁, and are satisfied by at least one actual system trace in \( \text{Cont}(σ₀, τ) \), the transformer Ψ has a unique smallest fixpoint Σ⁺ which is satisfied by all traces σ in \( \text{Cont}(σ₀, τ) \) with \( σ|V₁ \models Σ₁ \). Furthermore Σ⁺ can be obtained as the limit of the sequence \( Σ₀, Ψ(Σ₀), Ψ²(Σ₀), \ldots \) of approximations if \( Σ₀ \) is a specifications which agrees with Σ₁ on V₁, and is satisfied by at least one actual system trace in \( \text{Cont}(σ₀, τ) \), and satisfies \( Σ₀ \preceq Σ⁺ \).

Proof. Analogous to that for Theorem 2.

4. **FIXPOINTS IN RTC**

In this section, we transfer the results of the preceding section to RTC. We must then first show that the general assumptions introduced in Section 3 hold for RTC. This will
concern both the version of RTC with 0 as initial time point and the version where time starts at \(-\infty\).

Consider a system with set \(V\) of streams. In RTC, a trace \(\sigma\) of the system maps each event stream \(v \in V\) to an arrival function \(R\), and maps each resource stream \(v \in V\) to a service function \(C\). To transfer results from Section 3, we should check that the system is simulatable. Not all systems need to be simulatable, but sufficient conditions are given in the following proposition.

**Proposition 1.** If there is a nonzero lower bound on the time needed to process an event by a component, then a system which does not have any cycle of resource streams is simulatable.

**Proof.** (Sketch) For any component, its future output can be foreseen until the next point in time when it receives an event, or the status of its resource input is changed. These points in time form an increasing sequence of time points that converges to \(\infty\). Furthermore, the next such time point is always coinciding with the arrival of an event at some component (either from the environment or from another component). These arrivals happen at an increasing sequence of time points, which can be uniquely inferred from the current time point and the given trace on input streams.

Concerning specifications, we should check that, for a set \(V\) of streams the set of specifications \(Spec(V)\) on \(V\) forms a cpo. Recall that a specification \(\Sigma \in Spec(V)\) maps each event stream \(v \in V\) to a pair of arrival curves \([\alpha^v, \alpha^v]\), and maps each resource stream \(v \in V\) to a pair of service curves \([\beta^v, \beta^v]\). In order to make \(Spec(V)\) a cpo, we augment the set of arrival curves by curves \([\alpha^v, \alpha^v]\) in which \(\alpha^v(\Delta) = \infty\) for all \(\Delta \geq \Delta_0\), for some \(\Delta_0 > 0\). Upper service curves are bounded by the constraint \(\beta^v(\Delta) \leq \Delta\), so we need not to make this extension for service curves. With this augmentation \((Spec(V), \sqsubseteq)\) forms a cpo. It is also easy to see that all specifications in \(Spec(V)\) are safety properties.

We should also check that specification transformers are correct, monotone and continuous: this is checked for each case. For instance, the equations (3) are monotone and continuous. Finally, for each trace \(\sigma\), there is a strongest specification which is satisfied by \(\sigma\): this is obtained as in Equations (1) and (2) in Section 2.

We can now transfer the results from the previous section. Assume a system with input streams \(V\) and output streams \(W\), which has no cycle formed by only resource streams. Assume a nonzero lower bound on the time needed to process an event by a component. Let \(\Psi\) be the established RTC equations for the streams \(V\) of the system: there are correct versions of these both for the case where time starts at 0, and where it starts at \(-\infty\). Let \(\Sigma_t\) be a specification of input streams \(V\). Then Theorem 2 holds for RTC where time starts at 0. For RTC where time starts at 0, we can even prove a slightly stronger version, namely the following.

**Theorem 5.** Assume that \(\Sigma_t\) allows at least one system trace which is eventually periodic; this happens, e.g., if the system is not fully loaded. Then, among all satisfiable specifications which agree with \(\Sigma_t\) on \(V\), \(\Psi\) has a unique smallest fixpoint \(\Sigma^*\). The fixpoint \(\Sigma^*\) is satisfied by all actual system traces \(\sigma\) with \(\sigma|_{V_t} \models \Sigma_t\). Furthermore \(\Sigma^*\) can be obtained as the limit of the sequence \(\Sigma_t, \Psi(\Sigma_t), \Psi(\Psi(\Sigma_t))\) of approximations if \(\Sigma^0\) is any satisfiable specification which agrees with \(\Sigma_t\) on \(V_t\), and satisfies \(\Sigma^0 \sqsubseteq \Sigma^*\).

**Proof.** The difference, in comparison with Theorem 2, is that it is enough to start the iteration with a satisfiable specification \(\Sigma^0\); it need not be satisfied by any actual system trace. To prove this extension, let \(\sigma_p\) be an eventually periodic actual system trace. We note that by Theorem 1, using that \(\sigma_p\) is eventually periodic, there is for each \(\Delta\) an integer \(k\) such that \(\sigma_p\) satisfies \(\Psi(\Sigma^k)\) on all intervals smaller than \(\Delta\) whenever \(i \geq k\). By further considering the structure of RTC equations, we conclude that

\[
\lim_{i \to \infty} \Psi^i(\Sigma^0)(\Delta) \geq \lim_{i \to \infty} \Psi^i(\Sigma_p)(\Delta)
\]

for all \(\Delta\). The theorem follows.

For RTC, in which the time domain goes to \(-\infty\), we can prove a stronger analogue of Theorem 4.

**Theorem 6.** Let \(\tau\) be a time vector, and let \(\sigma^0\) be a trace with \(\sigma^0|_{V_t} \models \Sigma_t\) such that \(\sigma^0 \sqsubseteq \tau, \psi(\sigma^0)\). Assume that \(\Sigma_t\) is satisfied by at least one system trace \(\sigma_p\) which is eventually periodic with \(\sigma^\tau \sqsubseteq \sigma_p\). Then, among all specifications \(\Sigma\) which agree with \(\Sigma_t\) on \(V_t\), and satisfy \(\sigma^0 \sqsubseteq \Sigma\), there is a unique smallest fixpoint \(\Sigma^*\) of \(\Psi\). This fixpoint \(\Sigma^*\) is satisfied by all system traces \(\sigma \in Cont(\sigma^0, \tau)\) with \(\sigma|_{V_t} \models \Sigma_t\). Furthermore \(\Sigma^*\) can be obtained as the limit of the sequence \(\Sigma^0, \Psi(\Sigma^0), \Psi(\Psi(\Sigma^0)), \ldots\) of approximations if \(\Sigma^0\) is a specification which agrees with \(\Sigma_t\) on \(V_t\), and satisfies \(\sigma^0 \sqsubseteq \Sigma\) and \(\Sigma^0 \sqsubseteq \Sigma^*\).

**Proof.** Analogous to the previous theorem.

Theorems 5 and 6 suggest a methodology for finding the optimal fixpoint of \(\Psi\).

1. Construct some trace \(\sigma\) of the system, such that \(\sigma|_{V_t} \models \Sigma_t\), and such that \(\sigma\) satisfies the optimal fixpoint of \(\Psi\). Such a trace could be found, by constructing a simulation which is an actual system trace. The task is made easier by the observation that we need to find only one trace, and can choose one which is as regular as possible, e.g., as an infinitely repeating periodic trace.

2. Let \(\Sigma_\sigma\) be a tightest specification of the above trace, and use \(\Sigma_\sigma\) as an initial approximation \(\Sigma^0\).

3. An alternative way to construct an initial approximation \(\Sigma^0\) is by analytic techniques using long-term rates, as described in the next section.

4. Perform fixpoint iteration by computing the sequence \(\Sigma^0, \Psi(\Sigma^0), \ldots\). This is guaranteed to converge to a limit \(\Sigma^*\) which is the optimal fixpoint of \(\Psi\).

To summarize: the recommended way to compute fixpoints by iteration is to start from a strong initial approximation, which is included in the sought optimal fixpoint and thereafter iterate towards a fixpoint. We observe that the dual approach – starting from a very weak initial approximation and iterate towards a stronger solution – will in most cases yield quite poor precision.

### 4.1 Obtaining an Initial Approximation

Theorem 5 provides a guarantee that a fixpoint solution will exist in RTC, provided that there is some solution to the equations. An initial approximation can be constructed.
We assume that \( n \geq 2 \). If \( n = 2 \), the system is fully loaded, and if \( n > 2 \), the system is (more or less) underloaded. We know that the actual system trace will quickly stabilize to a situation where each event is processed first by \( T_1 \), then by \( T_2 \), so that end-to-end latency is \( 2 \).

As a measure of the “precision” of fixpoints, let us focus on two quantities characterized by specifications of events from \( T_1 \) to \( T_2 \), i.e., arrival curves \([\alpha', \alpha'']\):

- \( \text{maxburst} \), which is the maximum possible number of consecutive events that are allowed to be emitted with exactly one unit time duration, i.e., without any pause in the processing activity of \( T_1 \).

The quantity \( \text{maxburst} \) can be obtained as

\[
\{ \Delta \in \mathbb{R} : \alpha''(\Delta) \geq \Delta \}.
\]

- \( \text{maxgap} \), which is the maximum time interval during which no processing activity is required from \( T_2 \). In \( \mathcal{R} \) which starts at time 0, this quantity is represented as \( \{ \Delta \in \mathbb{R} : \alpha'(\Delta) = 0 \} \), but with unbounded past, the quantity is

\[
\{ \Delta \in \mathbb{R} : \alpha'(\Delta + 1) = 0 \}
\]

since there is always a previous event, which makes \( T_2 \) occupied for one unit time.

Let us now calculate lower bounds on \( \text{maxburst} \) and \( \text{maxgap} \) for a given value of \( n \). We first consider the case where time starts at time 0. We obtain the following relations between \( \text{maxburst} \) and \( \text{maxgap} \):

- \( \text{maxgap} \geq \text{maxburst} + 1 \). This is since \( T_2 \) can be continuously busy for \( \text{maxburst} \) time units, meaning that \( T_1 \) initially can be without processing resources for \( \text{maxburst} \) time units, implying that the first event is emitted by \( T_1 \) after \( \text{maxburst} + 1 \) time units. In the case where time starts at -\( \infty \), this relation should rather be \( \text{maxgap} \geq \text{maxburst} \).

- \( \text{maxburst} \) is at least the largest value \( k \) that satisfies

\[
\begin{align*}
k &\leq \text{maxgap} \\
n(k-1) &\leq \text{maxburst} + k - 1 \quad (6)
n(k-1) &\leq \text{maxburst} + \text{maxgap} - 1
\end{align*}
\]

To see this, note that \( T_1 \) can be without processing resources for \( \text{maxburst} \) time units, thereafter it can obtain \( \text{maxgap} \) consecutive time units of processing resource. To emit \( k \) consecutive events, \( T_1 \) needs to have first a queueing phase, which can have length \( \text{maxburst} \), followed by \( k \) units of processing time.

Thus, we need to have

- \( k \leq \text{maxgap} \), to process the \( k \) events,
- that the \( k \)th event arrives before the \( k - 1 \) first events have been processed; the \( k \)th event arrives at time \( n(k-1) \) after start of the queueing phase, and the \( k - 1 \) first events have arrived \( \text{maxburst} + k - 1 \) units after start of the queueing phase,
- that the \( k \)th event, which can be output at time \( n(k-1) + 1 \) after queueing start, does so before end of processing phase, i.e., before \( \text{maxburst} + \text{maxgap} \).
Let us now consider the case $n = 2$. Then the largest values of $\maxgap$ and $\maxburst$ converge towards $\infty$. To see this, assume that there are largest values $b, g$ for them. Then $g \leq b + 1$ from first relation. thereafter, we also see that the second relation allows $k = b + 1$. For the case $n > 2$, the above equations give no such growth.

We thereafter consider the case where time starts at $-\infty$. In this case, the optimal fixpoint of the RTC equations will be a tightest characterization of actual system trace, i.e., the fixpoint is as precise as possible. We have confirmed this by calculating the fixpoint in our MPA tool. We do not obtain the same growth in $\maxburst$ and $\maxgap$ as in the case where time starts at 0, since the equations that mutually bound $b$ and $g$ in terms of each other are slightly different (see above).

This example illustrates a system, where under full load we get perfect precision if we use the RTC equations that are correct for unbounded past, but no precision at all if we use RTC equations that are correct from time 0.

6. EXPERIMENTAL ILLUSTRATION

In this section, we show with a concrete example how the fixpoint iteration described in the previous sections succeeds in the analysis of a distributed system with a cyclic dependency. Consider the system depicted in Figure 3. It consists of a sequence of three tasks $T_1$, $T_2$ and $T_3$ that process a periodic event stream $\alpha_I$. The system has two computation resources with availability $\beta_I(\Delta) = \beta_{II}(\Delta) = \Delta$. The first resource is shared by $T_1$ and $T_3$ according to a fixed priority scheduling policy with $T_3$ having higher priority than $T_1$. The system contains a cyclic dependency since $T_1$ indirectly triggers $T_3$ while $T_3$ preempts $T_1$.

![Figure 3: Performance model for experiment](image)

We assume that the input event stream $\alpha_I$ is strictly periodic with a period $P$ of 10 time units, i.e.

$$\alpha'I(\Delta) = \left\lfloor \frac{\Delta}{10} \right\rfloor \quad \text{and} \quad \alpha''I(\Delta) = \left\lceil \frac{\Delta}{10} \right\rceil$$

Further we assume that $T_1$, $T_2$ and $T_3$ have constant processing times of 4, 7 and 5 time units, respectively. We want to compute bounds for the event streams denoted with $\alpha'$, $\alpha''$ and for the service stream denoted with $\beta'$.

In order to find an appropriate initial value for the fixpoint iteration we first simulate the system execution. For this simulation we use the PESIMDES simulator environment. Since the system architecture is simple, the simulation could also be performed by hand without much effort. We observe that after an initial transitory phase, the task $T_1$ produces outputs events for the task $T_2$ with a recurring timing pattern. In particular it produces events with consecutive distances of 4 time units, 16 time units, 4 time units, 16 time units etc. This makes it easy to characterize the behavior $\sigma$ of the event stream $T_1$-$T_2$ and find the tightest specification $\Sigma_\sigma$ of this behavior in terms of upper and lower arrival curves. The corresponding arrival curves are depicted in Figure 4.

![Figure 4: Tightest specification of the simulated trace T1-T2](image)

We then use this arrival curves as initial values for the fixpoint iteration. In particular we model the tasks $T_1$, $T_2$ and $T_3$ with three Greedy Processing Components GPC1, GPC2 and GPC3 and repeatedly compute their outgoing arrival and service curves in the following order: GPC2, GPC3, GPC1. After 4 iterations we reach a fixpoint, i.e. all the arrival curves and service curves in the performance model are stable. The obtained characterizations of $\alpha'$, $\alpha''$ and $\beta'$ are depicted in the Figures 5, 6 and 7, respectively.

![Figure 5: Bounds for the event stream T1-T2](image)

7. CONCLUSION

We have performed a detailed study of correctness and preciseness of fixpoints obtained by solving the RTC equations for systems with cyclic dependencies. This has been performed by developing an operational semantics. Under mild assumptions (“no zero-delay cycles”), we have proven that any satisfiable fixpoint correctly characterizes all allowed system behaviors. The results indicate that the recommended way to compute fixpoints by iteration is to start from a strong initial approximation, which is included in
the sought optimal fixpoint and thereafter iterate towards a
fixpoint, obtaining successively weaker and weaker approxima-
tions as the iteration progresses.

A major problem is to find a good initial approximation
for the iteration. We consider two ways to find this approxi-
mation: one is to construct a simple periodic simulation
of the system, from which the initial approximation is de-
termined, another is to calculate the long-term rates of input
streams, use them to calculate stable long-term rates at in-
ternal channels, and thereafter extract a strongest initial
approximation.

We considered the precision obtainable by the optimal fix-
point. One example illustrated that in boundary cases (full
load), small variations in the setup can have large effects on
precision. An experiment indicated an often typical effect:
that the jitter is often overapproximated to some extent by
the optimal fixpoint: this is an inherent consequence of the
fact that RTC does not model phase correlations between
different components. Future work includes to understand
better how the obtained precision depends on parameters in
the input specification.

Let us make an additional comparison with the work by
Schisler et al. [5]. An interesting idea, originated by them
is to find long-term rates of streams in order to find initial
specifications for the fixpoint iteration. They do not char-
acterize when such long-term rates exist. In our paper, we
have proven that such long-term rates always exist whenever
a specification is satisfiable.

Appendix: Min-Max Algebra
The min-plus convolution ⊗ and min-plus deconvolution ⊘ of f
and g are defined as:

\[(f \otimes g)(\Delta) = \inf_{0 \leq \lambda \leq \Delta} \{f(\Delta - \lambda) + g(\lambda)\}\]
\[(f \oslash g)(\Delta) = \sup_{\lambda \geq 0} \{f(\Delta + \lambda) - g(\lambda)\}\]

A curve f is sub-additive, if

\[f(a) + f(b) \geq f(a + b) \forall a, b \geq 0\]

and super-additive, if

\[f(a) + f(b) \leq f(a + b) \forall a, b \geq 0\]

8. REFERENCES
[1] M. Fekete. Über die Verteilung der Wurzeln bei gewissen
algebraischen Gleichungen mit ganzzahligen Koeffizienten.
for complex embedded applications. Int. J. of Embedded
programming. In IFIP 74, pages 471–475. North-Holland,
1974.
Theory of Deterministic Queuing Systems for the Internet.
Network calculus for real time analysis of embedded
systems with cyclic task dependencies. In G. Hu, editor,
Proc. 20th International Conference on Computers and
Their Applications, CATA 2005, March 16–18, 2005,
space exploration of network processor architectures. In
Network Processor Design: Issues and Practices, Volume 1,
pages 55–89. 2002.
framework for evaluating design tradeoffs in packet
processing architectures. In 39th Design Automation
Conference (DAC 2002), pages 880–885, New Orleans LA,
USA, June 2002. ACM Press.
[8] E. Wandeler. Modular Performance Analysis and
Interface-Based Design for Embedded RealTime Systems.
characterization of event streams in analysis of hard
real-time applications. In IEEE Real-Time and Embedded
Technology and Applications Symposium, pages 450–461,
2004.
System architecture evaluation using modular performance
analysis - a case study. Software Tools for Technology
Proc. CONCUR ’93, Theories of Concurrency: Unification
and Extension, volume 715 of Lecture Notes in Computer