DISS. ETH NO. 19619

Compositional Design and Analysis of Distributed, Cyclic, and Adaptive Embedded Real-Time Systems

A dissertation submitted to

ETH ZURICH

for the degree of

Doctor of Sciences

presented by

NIKOLAY NIKOLAEV STOIMENOV

Bachelor of Computer Science, First Class Honours,
The University of Adelaide, Australia

born April 25, 1979

citizen of Bulgaria

accepted on the recommendation of

Prof. Dr. Lothar Thiele, examiner
Prof. Dr. Petru Eles, co-examiner

2011

Institut für Technische Informatik und Kommunikationsnetze
Computer Engineering and Networks Laboratory

TIK-SCHRIFTENREIHE NR. 124

Nikolay Nikolaev Stoimenov

Compositional Design and Analysis of Distributed, Cyclic, and Adaptive Embedded Real-Time Systems

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

A dissertation submitted to
ETH Zurich
for the degree of Doctor of Sciences

Diss. ETH No. 19619

Prof. Dr. Lothar Thiele, examiner
Prof. Dr. Petru Eles, co-examiner

Examination date: April 27, 2011

Abstract

Embedded systems are computer systems that are deeply integrated in and interact with the physical world. The physical world often imposes strict timing constraints on these systems. Therefore, the correct operation of such systems depends not only on the values of the produced results, but also on their timing. Such systems are called real-time systems.

Methods for system-level performance analysis play an integral part during the early design phase of embedded real-time systems. They support the analysis of various non-functional performance characteristics, and alleviate the choice of important design decisions before much time and resources are invested in detailed implementations.

Compositional formal methods for performance analysis are able to quickly provide hard upper and lower bounds on performance characteristics. However, such analytical methods are limited in their scope and accuracy as they cannot incorporate many system details in the analysis.

Interface-based design has been proposed in order to unify the steps of designing a system and analyzing it. It supports the paradigm for correct-by-construction design in the domain of embedded real-time systems. It can significantly shorten the design time of complex distributed embedded real-time systems.

Recently, a method for Modular Performance Analysis based on Real-Time Calculus has been proposed that is also connected with the principles of interface-based design in the framework of Real-Time Interfaces. Both of them support the analysis and design of complex distributed embedded real-time systems. This thesis builds on these results and extends them in several directions. The main contributions of this work are summarized as follows:

- A novel framework for interface-based design of distributed embedded real-time systems is proposed. It includes properties

such as incremental design, independent-implementability, and refinement. It unifies many existing compositional performance analysis and interface-based design methods.

- Novel models and methods for interface-based design are proposed that support the analysis and design of distributed embedded real-time systems which have buffer overflow and underflow constraints, end-to-end delay constraints, variable execution demands of tasks, and complex resource sharing policies.

- A novel method for compositional performance analysis of marked graphs is proposed that can be used for distributed embedded real-time systems with cyclic data dependencies, finite buffers with blocking write semantics, variable execution demands of tasks, non-deterministic event streams and resource behaviors, and complex resource sharing policies.

- A novel method for mode change performance analysis of multi-mode embedded real-time systems is proposed that can be used for systems with non-deterministic event streams and resource behaviors, variable execution demands of tasks, complex resource sharing policies, and various mode change protocols.

- A novel scheduling server based on time division multiple access is proposed that can be reconfigured during run-time, and can guarantee the real-time properties of applications during reconfigurations.

Zusammenfassung

Eingebettete Systeme sind Computersysteme, die in eine physikalische Umgebung eingebunden sind und mit dieser intensiv interagieren. Oft müssen eingebettete Systeme strenge Zeitbedingungen einhalten, die von der physikalischen Umgebung auferlegt werden. Die korrekte Ausführung des Systems hängt in solchen Fällen nicht nur von den berechneten Ergebnissen ab, aber auch vom Zeitpunkt, an dem die Ergebnisse produziert werden. Solche Systeme werden allgemein als Echtzeit-Systeme bezeichnet.

In frühen Entwicklungsphasen von eingebetteten Echtzeit-Systemen spielen Methoden zur Leistungsbewertung auf Systemebene eine wesentliche Rolle. Sie ermöglichen die Analyse von verschiedenen Leistungskriterien wie beispielsweise die Ausführungszeit des Systems und erleichtern das Treffen von wichtigen Entwurfsentscheidungen bevor Zeit und Ressourcen für die eigentliche Implementierung des Systems aufgewendet werden.

Modulare formale Methoden zur Leistungsbewertung können in kurzer Zeit sichere obere und untere Schranken für Leistungskriterien von Systemen bestimmen. Diese analytischen Verfahren sind allerdings in ihrem Anwendungsbereich und ihrer Genauigkeit eingeschränkt, da sie viele Systemdetails nicht in die Analyse einbinden können.

Die interface-basierte Entwicklung von eingebetteten Echtzeit-Systemen wurde eingeführt, um die beiden Schritte des Entwurfs und der Analyse eines Systems zu vereinen. Diese Art von Entwicklung folgt dem Paradigma des correct-by-construction, d.h. Systeme werden so entworfen, dass die Leistungskriterien garantiert erfüllt sind. Interface-basierte Entwicklung kann zudem die Entwicklungszeit von komplexen verteilten Echtzeit-Systemen massgeblich verkürzen.

Vor kurzem wurde die Modular Performance Analysis, eine auf Real-Time Calculus basierte Methode zur Leistungsbewertung vorgestellt. Diese Methode ist eng mit dem Verfahren der Real-Time Interfaces

verknüpft, die die Prinzipien der interface-basierten Entwicklung umsetzt. Beide Verfahren unterstützen die Analyse und den Entwurf von komplexen verteilten eingebetteten Echtzeit-Systemen. Diese Dissertation baut auf beide Verfahren auf und erweitert sie in mehrere Richtungen. Die wichtigsten Beiträge dieser Arbeit können wie folgt zusammengefasst werden:

- Ein neues Framework zur interface-basierten Entwicklung von verteilten eingebetteten Echtzeit-Systemen wird vorgestellt. Das Verfahren erfüllt die Eigenschaften des inkrementellen Entwurfs, der unabhängigen Implementierbarkeit und der Verfeinerung. Es vereint verschiedene existierende modulare Methoden zur Leistungsbewertung und interface-basierten Entwicklung.

- Neue Modelle und Methoden für interface-basierte Entwicklung werden vorgeschlagen, die die Analyse und den Entwurf von verteilten eingebetteten Echtzeit-Systemen mit folgenden Eigenschaften ermöglichen: Einschränkungen bzgl. Puffer-Überlauf/Unterlauf, Einschränkungen bzgl. End-to-end Latenzen, variable Ausführungszeiten von Tasks, Ressourcen-Sharing anhand komplexer Verfahren.

- Eine neue Methode zur modularen Analyse von markierten Graphen wird vorgestellt, die für verteilte eingebettete Echtzeit-Systeme mit einer oder mehreren der folgenden Eigenschaften eingesetzt werden kann: Zyklische Abhängigkeiten zwischen Komponenten, endliche Puffer mit blocking-write Semantik, variable Ausführungszeiten von Tasks, nicht-deterministische Ereignisströme oder Resourcenverfügbarkeit, Ressourcen-Sharing anhand komplexer Verfahren.

- Ein neues Verfahren zur Analyse von Mode-Wechseln in eingebetteten Echtzeitsystemen mit mehreren Ausführungsmodi wird eingeführt. Das Verfahren unterstützt nicht-deterministische Erignisströme und Resourcenverfügbarkeit, variable Ausführungszeiten von Tasks, komplexe Verfahren für Ressourcen-Sharing sowie verschiedene Protokolle für Mode-Wechsel.

- Ein neuer auf Zeitmultiplexing aufbauender Scheduling-Server wird vorgestellt, der zur Laufzeit umkonfiguriert werden kann. Der Server garantiert die Einhaltung der Echtzeit-Anforderungen von Applikationen auch während der Umkonfigurierung.

Acknowledgement

This thesis describes joint work with Lothar Thiele, Ernesto Wandeler, Samarjit Chakraborty, Simon Perathoner, Jian-Jia Chen, Luca Santinelli, Giorgio Buttazzo, and Yanhong Liu. It would not have been possible to complete it without the numerous interactions with these excellent researchers.

I am particularly grateful and indebted to Lothar Thiele for giving me the opportunity to study and perform research under his supervision. I thank him for his great patience in working with me. I have learnt a lot from him during our meetings and discussions.

I would also like to thank Petru Eles for agreeing to be my co-examiner and moreover, agreeing to read this rather lengthy thesis.

I am grateful for having such wonderful working environment and colleagues at the institute. I will particularly miss the interesting discussions about cars and sport with Clemens Moser. I am very grateful to Iuliana Bacivarov and Cosmin Roman for their friendly support whenever I needed it. I will miss our gym training with Cosmin. I would like to thank especially Monica Fricker. Without her support, I would have been still looking for an apartment in Zurich.

I would like to express my gratitude to the supervisors of my Honours thesis at the University of Adelaide: Rob Esser, Charles Lakos, and David Knight. Without their encouragement and support, I would have never considered pursuing a PhD degree. I am especially grateful to David for our enduring friendship and the ever interesting discussions. I would like to also thank my friends in Australia: Fr John Shanahan, Fr Joe O'Mara, Pam Doddrell, Mary Myla Andamon, and Ani and Nasko Parashkevovi.

I am grateful to my friends from Varna who have always supported me during my studies abroad. I am grateful to Rumena Gradinarova for being such a wonderful teacher. And I am especially grateful to Tihomir Kyurchev for always being there whenever I needed him.

My deepest thanks go to my parents, Veselina and Nikolay. They have always supported me during my studies and my greatest hardships.

Last but not least, my gratefulness goes to my partner Lyubomira Petkova. Without your care and help, I would have never managed to go through this on my own.

The work presented in this thesis was supported by the National Center of Competence in Research in Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation (SNF) under grant number 5005-67322. In addition, this research has been supported by grants from the European Network of Excellence ARTIST2, ArtistDesign, and the European Community Seventh Framework Programme FP7/2007-2013 under grant number 216008 project Predator. This support is gratefully acknowledged.

Contents

1

Introduction

Embedded systems are computer systems that are deeply integrated in and interact with the physical world. Examples of applications that use such systems traditionally have been nuclear power plants, railway switching systems, automotive and avionics electronics, air traffic control, telecommunications, but today they also include wireless sensor nodes, multimedia systems, mobile phones, tablets, gaming consoles, electronic book readers, building automation systems, and many others.

An embedded system is a special-purpose information processing system. It is specialized in performing specific functions under various constraints in a specific application domain. Functions and constraints are usually known at design time and this information can be used for customization and optimization of the system.

The typical applications are *distributed* as they usually consist of many parallel tasks which are mapped onto different *heterogenous* hardware components in the platform. Very often, several tasks are mapped on a single processing element that they need to time share by the use of a resource scheduling policy. Hardware components communicate and interact through one or many communication networks. Therefore, the functional and non-functional properties of the system depend not only on the actual processing done by the application tasks, but also on the chosen mapping, the selected resource scheduling policies, and the data stream interactions on the common communication network. These and also the fact that processing elements can take independent resource access decisions make the system design challenging [TW06].

Moreover, the domain-specific nature of embedded systems can lead to high specialization of the used hardware components. This not only allows a designer to select hardware components that are optimal for the specific functions that they are going to perform, but it also makes the choice of components much more difficult. For example, in the automotive industry a single embedded control

unit (ECU) contains a communication controller, memory, a central processing unit (CPU), and I/O interfaces. But depending on the function and the environment of the ECU, a designer may have the choice between using a general purpose processor, microcontroller, digital-signal processor (DSP), field-programmable gate array (FPGA), or application-specific integration circuits (ASIC). Similarly, the communication network between ECUs may consist of various heterogenous subnetworks each of them characterized by specific communication protocols, scheduling policies, and topologies. This vastly increases the complexity of a design.

The physical world imposes strict timing constraints on these systems. The times of their reactions must be in strict concordance with the times of events in the real world. Therefore, the correct operation of such systems depends not only on the values of the produced results, but also on their timing. Such systems are called *real-time* systems. Examples are signal-processing applications, automatic manufacturing systems, avionics and automotive applications, and many others. A system is called *hard real-time* if violating the timing constraints can be fatal for the system or its environment, and soft real-time otherwise. A typical example in an automotive system of a real-time constraint is a constraint on the end-to-end delay experienced by data that is read by a sensor, possibly processed by several ECUs and communicated over several communication buses, and finally written to an actuator. A designer must ensure the timing predictability of such systems in advance. Therefore, various performance analysis methods play an important role during the early design phase.

1.1 System-Level Performance Analysis

A major challenge in the design of distributed embedded systems is determining essential performance characteristics of a system in the early design phase before much time and resources are invested in detailed implementations. A designer needs to decide on important questions such as: which functions should be implemented in hardware and which in software (partitioning), which hardware components should be used (allocation), and how the different software tasks should be distributed between the chosen hardware components (mapping). In addition, she needs to verify whether a particular design meets all timing, memory, power, energy, and temperature constraints. Moreover, she may want to optimize a design by answering the questions: what are the minimum required processor speeds in order to meet all constraints, what are the minimum required communication bandwidths, etc.

Answering these multiple and often conflicting questions creates a *large design space*. This has lead to development of *automatic design space exploration* tools that usually employ techniques for multi-objective decision making, black-box optimization, and randomized search algorithms [KTZ05]. Performance analysis methods play an important role in design space exploration. The different choices for the chosen hardware components, scheduling policies, or mapping of software tasks to different computation and communication resources need to be not only evaluated against the system specification, but also compared to each other in order to find the set of optimal choices. The automatic exploration of these design choices imposes constraints on the performance analysis methods for *short analysis times* because of the large design space, and ability to work with *incomplete information* as the detailed system specification may not be available at an early design phase.

The methods for performance analysis of distributed embedded systems can be classified as: simulation-based, holistic, and compositional, see [TW06]. *Simulation-based* methods are the most widely used, in particular the ones based on SystemC [GLMS02]. These methods have a large scope of application as they can take into account many dynamic interactions existing in a system. They have the possibility to model a system in any desired level of detail in order to obtain as accurate performance characteristics as possible. On the other hand, such high detailed modeling brings high complexity and high computation times. Performance evaluation quickly becomes the bottleneck in the design space exploration.

Another disadvantage of simulation-based methods is their insufficient corner case coverage. Usually they are not able to compute worst-case and best-case values of a performance characteristic but only compute values that are specific to particular traces of system inputs. It is often impossible for a designer to generate all possible system inputs and interactions, and therefore a simulation-based method cannot show if a system meets its specification constraints under all possible conditions.

While system-level analysis of complex embedded systems is currently mainly based on simulation, there exists also a large body of research that aims at developing *formal analytical methods*. These methods typically abstract the necessary properties of the concrete system. In contrast to simulation-based methods that inherently suffer from insufficient corner case coverage, the latter typically allow to obtain *hard upper and lower bounds* on the performance results, and are therefore also applicable in the analysis of hard real-time embedded systems. The concept of abstraction plays an important role in these methods as it allows to model large classes of non-deterministic system behaviors,

system inputs, environment and resource interactions, and not only specific instances of them.

The typical requirements for such performance analysis methods are to be able to compute *correct and accurate bounds* on the performance characteristics. Correct means that there exists no reachable system state or feasible reaction of the system environment that lead to violations of the computed bounds. While accuracy means that the determined bounds are as close as possible to the actual worst-case and best-case performance characteristics exhibited by the system during run-time [TW06].

Holistic performance analysis methods are an extension of classical formal methods for timing analysis of uniprocessor systems toward distributed systems [TC94, PEP02]. These methods handle the communication system similarly to processing nodes, and integrate the computation and communication schedulability analysis. They are able to handle various resource sharing policies, complex input arrivals with release jitter, and timing correlations between input arrivals. However, because of their holistic nature, these methods do not scale well as their complexity grows with the number of system components and different scheduling policies used. As discussed before, modern distributed embedded systems are highly heterogeneous in terms of the underlying hardware platform, the diverse concurrently running applications, and the different scheduling policies. Therefore, compositionality (modularity) is a key requirement for a fast performance analysis method.

1.1.1 Compositional Performance Analysis

Compositional methods typically abstract the properties of the concrete hardware/software components into so-called *abstract components* that build the fundament for performance analysis. Abstract components can represent various composable entities, such as tasks, resources, and scheduling disciplines. Instances of inputs and outputs of such components represent all first class citizens of the analysis method. They are abstract representations of relevant properties of the concrete hardware/software components, such as resource capabilities or event streams.

The SymTA/S method [RJE03] is based on classical formal methods for timing analysis of uniprocessor systems but instead of being extended to specific cases of heterogeneous distributed embedded systems as in holistic methods, existing results are applied in a compositional manner to the individual abstract components. The performance characteristics are computed locally and propagated through the system by the use of different well-known abstractions for event task arrival patterns and

interfaces for conversions between them. On the other hand, the Modular Performance Analysis (MPA) framework [CKT03b, WTVL06] is an analytical approach based on the Real-Time Calculus (RTC) [TCN00], which has its foundations in methods for worst-case analysis of communication networks (Network Calculus) [Cru91a, Cru91b].

MPA is an example of formal analytical performance analysis approaches that can determine guaranteed performance limits. While these techniques can compute hard performance bounds, they abstract from the complex interactions and state dependent behavior in the system. MPA uses a unifying model for the representation of different event patterns in the form of *arrival curves* known from the communication domain [Cru91a] which generalize on any previously known event task activation models. In addition, it uses a similar concept called *service curves* to represent the resources and their computational or communication capabilities, which allows MPA to model any combination of resource sharing policies in heterogeneous distributed embedded systems as well as complex hierarchical scheduling schemes. The detailed modeling of the capabilities of the shared resources and the event streams can lead to highly accurate performance results, see for example [CKT+03a].

An MPA model is a performance network of components, where application tasks are mapped to computation and communication resources. One can differentiate between three main entities: *event streams* represented as arrival curves, *resource streams* represented as service curves, and *application tasks* represented as processing components. The application tasks are activated by the event streams and process them by considering their interaction with the resource streams. On a higher level, the model is a network of components that communicate with each other through *event interfaces*. Performance metrics for the whole application are computed by combining the behavior of the individual components. This modularity aspect achieves short analysis times even for large systems. Typical performance metrics computed with MPA are upper and lower bounds on buffer levels, end-to-end delays experienced by events, and the number of events that can be processed in a time unit (throughput).

1.1.2 Challenges

Developing system-level performance analysis methods has a lot of challenges. They are very often the reason for the limited scope of the methods or their lack of accuracy. Typically the methods have difficulties coping with: non-deterministic event task activations, non-deterministic resource behaviors, complex resource sharing policies, resource access interferences, complex task activation schemes, complex processing

semantics, variable execution demands, and workload correlations. Most of them have been addressed before to different extents in the scientific literature, for an overview see [Max05, Wan06, Hai10, Sch11]. Here we give a brief overview of several challenges that are addressed in this thesis:

- *Cyclic data dependencies:* Many modern media-, signal-, and image-processing applications are specified with dataflow graphs that have cycles. Similarly distributed systems that have finite buffers with blocking write semantics are very often modeled with cyclic dataflow graphs. Compositional performance analysis methods face enormous challenges in the analysis of cyclic dataflow applications. The cycles in the information flow between the individual processes of an application lead to global, system-wide state dependencies. As a result, the timing behavior of a process (and as a result its use of the available resources) not only depends on predecessor processes that provide the data streams that are to be processed, but also on successor processes and the process itself.
- *Adaptive behavior:* Very often systems are required to change their functionality or characteristics during run-time due to changes inside of them or changes in their environments. Such changes can be observed in different operating modes for the applications in the system, different characteristics of the outside environment, or different characteristics of the hardware platform and the resources that it provides. A designer must ensure that the system meets its specification and real-time requirements not only when system characteristics are stable, but also when they are changing, e.g. when the system is in the process of a mode change. Specific abstractions need to be introduced to performance analysis methods that are able to capture the dynamics of the system when its characteristics are being changed.

1.2 Interface-Based Design

System-level performance analysis methods are used for analysis of distributed embedded real-time systems subsequently to the specification and design of the system. Only after completion of the design step, performance analysis is applied to the system design in a second step. The analysis result will then answer the 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.

In contrast to this two-step approach for design and posterior analysis is the idea for interface-based design [dAH01b, dAH05]. 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 while meeting real-time requirements based only on the information exposed in their 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 what a component expects from the other components in the system and the environment. Through *output guarantees*, a component interface informs the rest of the system and the environment what they can expect from this component. Typically such assumptions and guarantees specify intervals for permissable values of specific system parameters such as input data rates, processor speeds, communication bandwidths, etc. Composition of components into a system design is allowed only if their interface assumptions and guarantees are compatible which would automatically ensure satisfaction of all constraints in the system specification.

Interface-based design techniques extend the scope of system-level performance analysis methods by allowing them to answer more efficiently questions such as: given the arrival rates of the input streams and the system throughput requirements, what is the optimal buffer size for a component that is being added to a partially-designed architecture; can the scheduler for a particular processing element be replaced by a different scheduler without violating any of the real-time constraints of all existing components; what is the minimum required processing frequency such that none of the real-time constraints in the system are violated, etc. Previously, all of these questions would have been answered by trying many different designs with components that have different parameters, and subsequent performance analysis of the whole system would show whether a design satisfies the specification or not.

MPA promotes interface-based design of embedded systems with the concept of Real-Time Interfaces [Wan06]. It supports the design phase of embedded real-time systems by proposing interfaces whose compatibility means that the real-time properties of the system are satisfied. Assumptions and guarantees of component interfaces are specified with arrival and service curves. Compatibility between two interfaces is also expressed in terms of these entities. Real-Time Interfaces provide mechanisms to propagate component constraints through the system. Guarantees and assumptions are not any longer static but adapt according to the changing system constraints. Such changes occur when new components are added to the systems, existing components are

removed or change their parameters, or the system environment changes its requirements.

1.3 Aim of the Thesis

The results presented in this thesis aim at defending the following hypotheses:

1. *It is possible to develop compositional analytical performance analysis methods that can compute correct and accurate performance results for heterogenous distributed embedded real-time systems that have various complexities.*

2. *In addition, it is possible to extend such compositional performance analysis methods and connect them with principles of interface-based design in order to make the system design phase more efficient and less time consuming.*

Complexities in the above hypotheses may include: *multiple applications mapped to multiple computing and communication resources, memory and delay constraints, variable execution demands of tasks, complex resource sharing policies, non-deterministic event streams behavior, non-deterministic resource behavior, cyclic data dependencies, finite buffers with blocking write semantics, and adaptive behavior*.

1.4 Thesis Outline and Contributions

The thesis is divided into two major parts. Part I focuses on formalizing, generalizing, and extending the scope of compositional methods for performance analysis and interface-based design. Part II focuses on proposing new techniques for performance analysis of adaptive embedded real-time systems. In summary, the thesis contains the following topics and contributions:

Part I: Compositional Design and Analysis for Distributed Embedded Systems

This part focuses on formalizing and generalizing the existing framework for interface-based design with Real-Time Interfaces, extending it to distributed systems, and proposes a compositional method for performance analysis of systems with cyclic data dependencies.

Chapter 2: Interface-Based Design with Adaptive Real-Time Interfaces

This chapter formalizes and generalizes the framework of Real-Time Interfaces. The main contributions are as follows:

- The properties of independent implementability, incremental design, and refinement of components and interfaces are formally defined. Conditions for their existence in an interface-based design framework are derived.
- The notions of adaptive interfaces and transfer functions are formally defined. They support the design by providing mechanisms to propagate constraints between components.
- Existing compositional frameworks for performance analysis and interface-based design are unified.

Chapter 3: Interface Algebra with Rate Interfaces

This chapter extends the framework of Real-Time Interfaces. The main contributions are as follows:

- The framework of Real-Time Interfaces is extended toward distributed multiprocessor embedded real-time systems.
- The new results are applied on a realistic multiprocessor video-processing system that has buffer overflow and underflow constraints, and fixed priority or earliest deadline first scheduled components.
- A new interface is added to the framework of Real-Time Interfaces in order to allow it to include global real-time constraints that span networks of components, and not only single components.

Chapter 4: Compositional Analysis for Real-Time Systems with Cyclic Dataflow

This chapter extends the scope of compositional performance analysis methods toward distributed embedded real-time systems with cyclic data dependencies. The main contributions are as follows:

- The Modular Performance Analysis method is extended toward the class of systems modeled with marked graphs. Unlike any previously existing approaches, the new method takes accurately into account the interactions of the graph processes with resources, and accurately models the resource scheduling policies.

- The proposed analysis method can be embedded into previously existing compositional methods for performance analysis.
- The proposed analysis method yields performance characteristics of accuracy higher than any of the previously existing approaches. This is illustrated with a case study of two realistic signal-processing applications running concurrently on a multiprocessor platform.

Part II: Analysis of Adaptive Embedded Systems

This part focuses on proposing performance analysis methods for embedded real-time systems with adaptive behavior.

Chapter 5: Analysis of Adaptive Applications

This chapter develops a performance analysis method for multi-mode embedded real-time systems that run applications with several operating modes. The main contributions are as follows:

- The Modular Performance Analysis framework is extended toward uniprocessor multi-mode embedded real-time systems with fixed priority scheduling. Unlike any previously existing approaches, the new method can be applied to systems with any complex event task activation pattern, and any complex resource behavior.
- The method can be applied for analysis of mode change protocols with and without offsets.

Chapter 6: Analysis of Adaptive Schedulers

This chapter develops a performance analysis method for multi-mode embedded real-time systems that provide reconfigurable resource reservations. The main contributions are as follows:

- A framework for adaptive resource reservations based on time division multiple access is proposed. Each application is served by an Adaptive Server with Guarantees that can change its parameters during run-time without violating the real-time constraints of the application.
- Resource reconfiguration scenarios are classified, algorithms that preserve the real-time constraints are proposed for each scenario, and analysis of the resource guarantees provided by each algorithm is developed.

Part I

Compositional Design and Analysis for Distributed Embedded Systems

2

Interface-Based Design with Adaptive Real-Time Interfaces

A major challenge in the design process of complex real-time embedded system is the analysis of essential characteristics of a system architecture at an early design stage. The goal is to support the design decisions as early as possible and before much time and resources are invested in detailed implementations. Essential characteristics for embedded real-time systems are for example whether maximum delay and throughput constraints are satisfied, what the on-chip memory requirements are, or how different architectural elements must be dimensioned.

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.

In contrast to the two-step approach for design and posterior analysis is the idea for *interface-based design* described by de Alfaro and Henzinger [dAH01b, dAH05]. They propose 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. Such a theory supports the crucial properties of *refinement*, *incremental design*, and *independent implementability*.

This chapter develops an interface-based design theory for real-time systems. It formalizes and generalizes the theory of *Real-Time Interfaces* introduced in [WT05b, WT06a]. It formalizes and elaborates on the

concepts of abstract components and Adaptive Real-Time Interfaces. Properties such as refinement and incremental design are defined. The framework is very general and powerful, and it is applicable to many previously known compositional performance analysis methods for real-time systems.

2.1 Introduction

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. If at any composition a component interface cannot be composed with a subsystem, this already excludes the possibility that the complete system can 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*.

Besides these properties, the recently proposed interface theory of Real-Time Interfaces [WT05b, WT06a] also supports dynamic *adaptivity*. These interfaces not only expose enough information to resolve the composability with other component interfaces, but also they change their assumptions and guarantees when new components are added (removed) to (from) a partially designed system following principles of constraints propagation.

The contributions of this chapter can be summarized as follows:

- We formalize and generalize the framework of Real-Time Interfaces introduced in [WT05b, WT06a].
- Important aspects like independent implementability, refinement and incremental design are discussed and corresponding conditions are derived, see Sections 2.3 and 2.4.
- Traditional assume / guarantee interface theory is enriched with the notion of a transfer function that enables expressing relations between assume and guarantee inputs and outputs, and is crucial in the definition of adaptive interfaces.
- The new notion of adaptive interfaces supports the design by providing mechanisms to propagate constraints, for example (end-to-end) delays, computing and communication resources, energy and buffer spaces, see Section 2.4. Guarantees and assumptions are not any longer static but adapt according to the current system environment. This can be used to answer synthesis questions at design time, and to adapt a system at run-time when the requirements from the environment change.
- We unify a large set of modular and interface-based methods for real-time systems design, see Section 2.6.
- The use of Adaptive Real-Time Interfaces is discussed using a set of different examples, and a larger system in Section 2.7.

In the following description of Real-Time Interfaces, we will make a distinction between *abstract components* and their *adaptive interfaces*. Both terms are widely used and there are many interpretations available. Before specifying formally the meaning of both terms in the context of real-time interfaces, let us introduce them informally.

Abstract components describe building blocks for a system-wide analysis. They can represent various composable entities, such as tasks,

resources, and scheduling disciplines. Instances of inputs and outputs of such components represent all first class citizens of the analysis method. They are abstract representations of relevant properties of the concrete hardware/software components, such as resource capabilities or event streams. In summary, an abstract component provides a mathematical abstract model of a hardware/software component.

An interface, on the other hand, should expose enough information about a component as to make it possible to predict if two or more components can work properly together by looking only at their interfaces [dAH05]. *Adaptive interfaces*, as used in the context of real-time systems [WT05b, WT06a], not only allow for such an analysis of a given real-time system, but also support the design by providing mechanisms to propagate constraints, for example (end-to-end) delays, computing and communication resources, buffer spaces, and energy. Guarantees and assumptions are not any longer static but adapt according to the changing system constraints. Such changes occur when new components are added to the systems, existing components are removed or change their parameters, or the system environment changes its requirements. Such changes can be during the design phase of a system when different design choices are evaluated, or at run-time when the requirements toward the system may change dynamically.

The fact that many well-known analysis methods can be represented makes the described framework widely applicable. Therefore, the simple examples used in this chapter should be understood only as illustrations of the main principles.

Section 2.2 continues with a brief discussion on the related work. Section 2.3 describes the essential properties for a component-based design and analysis framework for embedded real-time systems. Section 2.4 describes these properties for an interface-based design framework and introduces the concept of Adaptive Real-Time Interfaces. Section 2.5 discusses possible extensions of the proposed basic framework. Section 2.6 discusses several frameworks for compositional performance analysis that can be unified in the proposed framework. Section 2.7 shows the applicability of the framework for design and analysis of a realistic system. Finally, Section 2.8 concludes this chapter with a brief summary and a discussion.

2.2 Related Work

An analytical framework for system-level performance analysis was proposed in [RJE03]. It uses a number of well-known abstractions to capture the timing behavior of event streams, and provides additional

interfaces between them. Traditional schedulability analysis results are then used to analyze a component-based real-time system design. The framework presented in [WRM+05], on the other hand, uses the notion of traditional software component standards such as CORBA. It proposes extensions for the support of real-time services. The approach proposed in [SL03], composes real-time components by making use of hierarchical scheduling. Finally, the approach to modular performance analysis proposed in [CKT03b] relies on Network Calculus [LBT01] and its extensions to the domain of real-time embedded systems known as Real-Time Calculus [TCN00]. It is based on a general event and resource model, and allows to analyze complex systems with hierarchical scheduling and arbitration. It can take computation as well as communication resources into account. All of these methods for performance analysis are only suitable for the two-step design process where a fully designed system is verified a posteriori. They cannot be used directly in a holistic one-step interface-based design process.

Traditional interface-based design theories focus on stateful interfaces [dAH01a, dAHS02, CdAHS03], while the work presented here is based on stateless assume / guarantee (A/G) interfaces. There exists a recent approach that also relies on stateless A/G interfaces proposed in [HM06], but it is limited to using bounded-delay resource models with earliest deadline first (EDF) scheduling. Similarly, an interface-based framework for compositional design of real-time systems is proposed in [ESSL06], but it is limited only to hierarchically scheduled systems that use rate monotonic (RM) or EDF scheduling under the periodic resource model.

The theory presented in this chapter not only generalizes the work presented in [HM06, ESSL06], but it is also able to empower a large class of real-time system-level analysis methods with the principles of interface-based design. The presented theory is thereby an analysis, extension and generalization of the basic Real-Time Interfaces that were first introduced in [WT05b, WT06a].

2.3 Abstract Components

As has been previously described, abstract components in the context of performance analysis represent the building blocks of an analysis method. At first, single abstract components will be described.

2.3.1 Single Abstract Component

The abstract components considered throughout this chapter can be formally defined as follows:

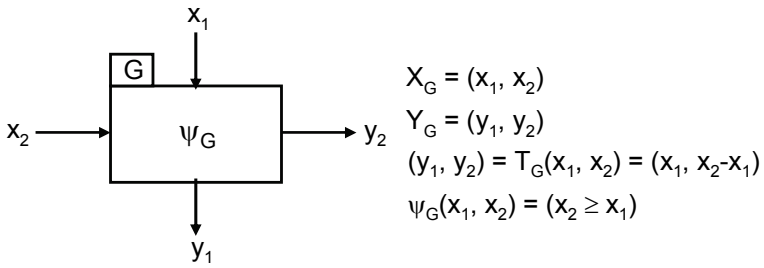

Fig. 2.1: A simple abstract component *G* representing the analysis of a shared bus from Example 2.2

Definition 2.1: (Abstract Component) *An* abstract component *F is the tuple* (X, Y, T, Ψ) *where X is a set of input variables, Y is a set of output variables, T is the transfer function* $Y = T(X)$*, and* Ψ *is a predicate over the input variables X. An abstract component* can work properly *iff* Ψ *holds for some valuation of the input variables X.*

The transfer function $T(X)$[1] represents the transformation of input values X to output values Y in the analysis, i.e. $Y = T(X)$. The predicate $\Psi(X)$ restricts the scope in which the underlying component can be used. It formalizes the notion that a component can work properly, i.e. it meets certain real-time requirements.

In the chapter we provide a running example that is not directly related to real-time analysis but it serves well to illustrate the basic concepts.

Example 2.2: *Let suppose that packet streams share a communication unit. The analysis simply adds the data rates of the streams and requires that the accumulated rate does not exceed the available bandwidth.*

Figure 2.1 represents a corresponding abstract component G where x_2 *represents the available bandwidth and* x_1 *the bandwidth used by a single packet stream. Note that the outgoing packet stream* y_1 *may trigger a computation demand on a computing unit that is attached to the communication unit.* y_2 *denotes the remaining bandwidth available to other streams.* Ψ_G *denotes that the component works properly only if the available bandwidth is larger than the requested one. In this example, the input and output variables are non-negative real numbers. In general, they could however be of any type. In particular, in real-time analysis, functions over independent variables are often used, e.g. arrival and service curves, demand bound functions, etc. Moreover, the predicate may also be used to describe other kinds of constraints such as memory, energy, or delays.*

[1] Unless there are ambiguities, we will not distinguish between a set of variables and their valuation, e.g. depending on the context, X can represent a set of input variables or their values.

2.3.2 Network of Abstract Components

Outputs of abstract components can be connected to inputs which leads to a network of abstract components.[2] The inputs (outputs) of such a network are those inputs (outputs) of its abstract components that are not connected to some output (input).

Definition 2.3: (Network of Abstract Components) *A* network of abstract components $\mathbb{F}$ *consists of connected abstract components* $F \in \mathbf{F}$ *that form a* connection graph. *If an output is connected to an input, then the variables are identical; unconnected variables are assumed to be different. Connections are point-to-point, i.e. an output is connected to at most one input and vice versa.*[3] *The following elements are defined:*

- *Inputs of* $\mathbb{F}$*:* $X_{\mathbb{F}} = (\bigcup_{F\in\mathbf{F}} X_F)\backslash(\bigcup_{F\in\mathbf{F}} Y_F)$
- *Outputs of* $\mathbb{F}$*:* $Y_{\mathbb{F}} = (\bigcup_{F\in\mathbf{F}} Y_F)\backslash(\bigcup_{F\in\mathbf{F}} X_F)$
- *Predicate of* $\mathbb{F}$*:* $\Psi_{\mathbb{F}} = \bigwedge_{F\in\mathbf{F}} \Psi_F$
- *The transfer function* $T_{\mathbb{F}}$ *of* $\mathbb{F}$ *is determined by concatenating* T_F, $F \in \mathbf{F}$, *according to the connection graph, i.e.* $Y_{\mathbb{F}} = T_{\mathbb{F}}(X_{\mathbb{F}}) \Rightarrow \bigwedge_{F\in\mathbf{F}}(Y_F = T_F(X_F))$ *is satisfiable for all valuations of* $X_{\mathbb{F}}$.

A network of abstract components can work properly *iff* $\Psi_{\mathbb{F}}$ *is satisfiable for some inputs* $X_{\mathbb{F}}$.

In this section, we will only look at networks of abstract components whose connection graph does not contain directed cycles. Possible extensions are discussed in Section 2.5. Moreover, it should be noted that a network of abstract components as defined in Definition 2.3 is in the form of an abstract component again, i.e. it is defined by sets of input and output variables, a transfer function and a predicate over the input variables. In order to understand the important notion of working properly in the context of networks of components, we introduce the notion of a subnetwork.

Definition 2.4: (Subnetwork of Abstract Components) *A* subnetwork $\mathbb{G}$ *of a network* $\mathbb{F}$ *is denoted as* $\mathbb{G} \subseteq \mathbb{F}$ *and consists of abstract components in* $\mathbf{G} \subseteq \mathbf{F}$ *that form a subgraph of the connection graph, i.e. deleting abstract components* $\mathbf{F}\backslash\mathbf{G}$ *and incident connections.*

[2] For the sake of simplicity, the above component model does not define types that would restrict possible compositions. It will be straightforward to extend it.

[3] This is without any restriction in generality. If an output needs to be connected to several inputs, then an additional one-to-many component will do.

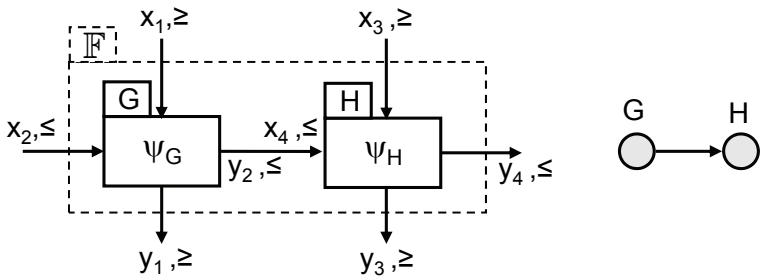

Fig. 2.2: A network of components $\mathbb{F}$ consisting of two abstract components G and H from Example 2.6. The associated partial orders are shown next to each input and output variable. The connection graph is also shown on the right

Similarly we need to define what it means to connect two networks of components and when they are compatible.

Definition 2.5: (Connection and Compatibility for Abstract Components) Connecting *two networks* $\mathbb{G}$ *and* $\mathbb{H}$ *to form a network* $\mathbb{F}$ *is denoted as* $\mathbb{G} \parallel \mathbb{H} = \mathbb{F}$. *We call* $\mathbb{G}$ *and* $\mathbb{H}$ compatible *(*$\mathbb{G} \sim \mathbb{H}$*) if the network* $\mathbb{G} \parallel \mathbb{H}$ can work properly.

Now the above concepts can be used to continue with the running Example 2.2.

Example 2.6: *Figure 2.2 represents a simple network of two abstract components where each is of the form defined in Example 2.2.*

Here, two packet streams with bandwidth x_1 *and* x_3 *share a common bus with bandwidth* x_2. x_4 *and* y_2 *ought to be just one single variable (to match the previous definitions), but the original names from G and H are used to simplify the presentation. We have* $X_{\mathbb{F}} = (x_1, x_2, x_3)$, $Y_{\mathbb{F}} = (y_1, y_3, y_4)$, $\Psi_{\mathbb{F}}(x_1, x_2, x_3) = (x_2 \geq x_1) \wedge (x_2 - x_1 \geq x_3)$ *and* $(y_1, y_3, y_4) = T_{\mathbb{F}}(x_1, x_2, x_3) = (x_1, x_3, x_2 - x_1 - x_3)$. *Of course,* $\Psi_{\mathbb{F}}$ *is satisfiable for some input* $X_{\mathbb{F}}$ *and therefore, the abstract components can work properly.*

2.3.3 Incremental Design

Following [dAH05], we want to ensure the property of *incremental design*, i.e. if a network of abstract components can work properly, then it can be composed in any order from subnetworks and these subnetworks can work properly also. In other words, if a subnetwork can not work properly, then the whole network can not. Obviously, the reverse direction can not be expected: If two networks can work properly, their composition may not. This is described formally in the following theorem.

Theorem 2.7: (Incremental Design with Abstract Components) *Given a network* $\mathbb{F}$*. If* $\mathbb{F}$ *can work properly then any sub-network* $\mathbb{G} \subseteq \mathbb{F}$ *can work properly.*

Proof. If $\mathbb{F}$ can work properly, then $\Psi_{\mathbb{F}} = \bigwedge_{F \in \mathbf{F}} \Psi_F$ is satisfied for some input valuation $X_{\mathbb{F}}$, see Definition 2.3. Following the definition of the transfer function $T_{\mathbb{F}}$, we can determine for such an input the values of all internal variables by concatenation of functions T_F, $F \in \mathbf{F}$. For any subnetwork $\mathbb{G} \subseteq \mathbb{F}$, we can set the input variables to the same values as in $\mathbb{F}$. As $\bigwedge_{F \in \mathbf{F}} \Psi_F$ was satisfied, $\bigwedge_{G \in \mathbf{G} \subseteq \mathbf{F}} \Psi_G$ is satisfied too and $\mathbb{G}$ can work properly.

■

2.3.4 Refinement

In a design process, there is often the demand for independent implementability. One can replace the implementation of a subnetwork by another one, as long as the new abstract component representing this new subnetwork *refines* the original one. If the original network could work properly, then this design step should not violate this property. This way, the implementation of subsystems can be performed independently.

In order to allow refinement, we need to define the class of monotone abstract components.

Definition 2.8: (Monotone Abstract Component) *To each input and output variable of an abstract component, we associate an individual partial order denoted as* $\geq$*, i.e. a binary relation which is reflexive, antisymmetric, and transitive.*[4] *An abstract component F is called* monotone*, if for all* $X_F, \widehat{X}_F$ *we have that:*

$$X_F \geq \widehat{X}_F \Rightarrow T_F(X_F) \geq T_F(\widehat{X}_F) \,,$$
$$X_F \geq \widehat{X}_F \wedge \Psi_F(X_F) \Rightarrow \Psi_F(\widehat{X}_F) \,.$$

If monotone abstract components are connected*, then the partial orders assigned to connected input/output pairs must be identical.*

It can simply be seen that the composition of monotone abstract components is monotone again as we just compose monotone functions and conjunct monotone predicates. Without loss of generality we consider only monotone abstract components. In Chapter 3 we will show that

[4] If we write $X \geq \widehat{X}$ for sets of variables X and $\widehat{X}$, then the comparison is done using the binary relation that is specific for each variable.

many useful abstract components used for performance analysis are indeed monotone. Based on the notion of monotonicity, we can now define the refinement of an abstract component.

Definition 2.9: (Refinement for Abstract Components) *Given a monotone abstract component G that can work properly. Then G'* refines *G denoted as ($G \geq G'$) if:*

- *The sets of input and output variables of G and G' are equal.*[5]
- $(\Psi_G(X) \Rightarrow \Psi_{G'}(X)) \wedge (T_G(X) \geq T_{G'}(X))$ *for all valuations of input variables X.*

According to the above definition, a refined abstract component has a weaker predicate. In addition, the transfer functions satisfy $Y \geq T_G(X) \Rightarrow Y \geq T_{G'}(X)$ for all Y (using the respective partial order for each variable). Note that the abstract component in Definition 2.9 may also denote a network of abstract components, see Definition 2.3. Therefore, the notion of refinement holds for both single abstract components and networks of abstract components.

Theorem 2.10: (Refinement for Abstract Components Preserves Compatibility) *Given a network of monotone abstract components $\mathbb{F}$ that can work properly and an arbitrary partition $\mathbb{F} = \mathbb{G} \parallel \mathbb{H}$. If we refine $\mathbb{G}$ to $\mathbb{G}'$ ($\mathbb{G} \geq \mathbb{G}'$) then $\mathbb{F}' = \mathbb{G}' \parallel \mathbb{H}$ can work properly, i.e. $\mathbb{G}' \sim \mathbb{H}$.*

Proof. As $\mathbb{F}$ can work properly, $\bigwedge_{F \in \mathbf{F}} \Psi_F$ can be satisfied for some valuation of input variables $X_{\mathbb{F}}$. The variables in the network with G replaced by G' according to Definition 2.9 are denoted as $\widetilde{X}_{G'}$, $\widetilde{Y}_{G'}$, $\widetilde{X}_H$, $\widetilde{Y}_H$. In the original network they are denoted as X_G, Y_G, X_H, Y_H. As all transfer functions are monotone and because of Definition 2.9, we have $\widetilde{X}_{G'} \leqq X_G$, $\widetilde{Y}_{G'} \leq Y_G$, $\widetilde{X}_H \leq X_H$, and $\widetilde{Y}_H \leq Y_H$. Because $(\Psi_G(X_G) \Rightarrow \Psi_{G'}(X_G))$, $\widetilde{X}_{G'} \leq X_G$ and the monotonicity of Ψ and T (see Definition 2.8), the predicates of all abstract components in the new system are satisfied.

■

The concepts of refinement and independent implementability can now be illustrated in the running example.

[5] In order to reduce the notational overhead, we restrict the refinement of an abstract component to changes of its predicate and transfer function only. In a similar way to [dAH05], the number of inputs and outputs could also be changed.

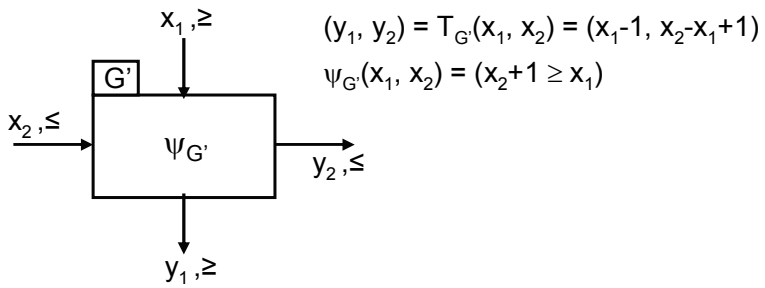

Fig. 2.3: The refined component G' ($G \geq G'$) used in Example 2.11, where G is shown in Fig. 2.1

Example 2.11: *Given the small network* $\mathbb{F}$ *in Fig. 2.2, we now consider the partial orders of the different variables. As all variables are simple real numbers and not complex data types such as tuples (e.g. representations of event streams in the form of 'period and jitter' or 'burst size and average rate') or curves (e.g. arrival and service curves), we only use the conventional 'greater or equal'* $\geq$ *for variables* x_1, x_3, y_1, y_3 *and 'less or equal'* $\leq$ *for variables* x_2, x_4, y_2, y_4*. Then the partial orders of connected inputs and outputs match.*

The transfer function $T_{\mathbb{F}}$ *and predicate* $\Psi_{\mathbb{F}}$ *are monotone, see Definition 2.8. For example, as* $\Psi_{\mathbb{F}}(x_1, x_2, x_3) = (x_2 \geq x_1) \wedge (x_2 - x_1 \geq x_3)$ *we have* $x_1 \geq \tilde{x}_1 \wedge x_2 \leq \tilde{x}_2 \wedge x_3 \geq \tilde{x}_3 \wedge [(x_2 \geq x_1) \wedge (x_2 - x_1 \geq x_3)] \Rightarrow (\tilde{x}_2 \geq \tilde{x}_1) \wedge (\tilde{x}_2 - \tilde{x}_1 \geq \tilde{x}_3)$.

A refined component G' *shown in Fig. 2.3, i.e.* $G \geq G'$*, could be characterized by* $(y_1, y_2) = T_{G'}(x_1, x_2) = (x_1 - 1, x_2 - x_1 + 1)$ *and* $\Psi_{G'} = (x_2 + 1 \geq x_1)$*, see Definition 2.9. Looking at* $T_{G'}$*, one can see that the component delivers a packet stream with a smaller bandwidth* ($y_1 < x_1$) *and provides more communication bandwidth to other packet streams via* y_2 *where* $y_2 > x_2 - x_1 + 1$.

If we would replace G *in Fig. 2.2 by* G'*, we would obtain a new network called* $\mathbb{F}'$ *according to Theorem 2.10. We can now compute the new predicate* $\Psi_{\mathbb{F}'}(x_1, x_2, x_3) = (x_2 + 1 \geq x_1) \wedge (x_2 - x_1 + 1 \geq x_3)$*. Then, it can easily be verified that Theorem 2.10 holds. In other words, if predicate* $\Psi_{\mathbb{F}}$ *holds for a certain valuation of inputs then also* $\Psi_{\mathbb{F}'}$ *holds, and therefore,* $\mathbb{F}'$ *can work properly.*

2.4 Adaptive Real-Time Interfaces

An adaptive interface of an abstract component not only exposes enough information to predict whether a composition with other components is compatible, but in addition, (a) it is adaptive as it changes assumptions and guarantees when connected components and their predicates change, new components are added, or existing components are removed and (b) it distributes constraints globally through the whole network. At first, we will describe the main concept informally.

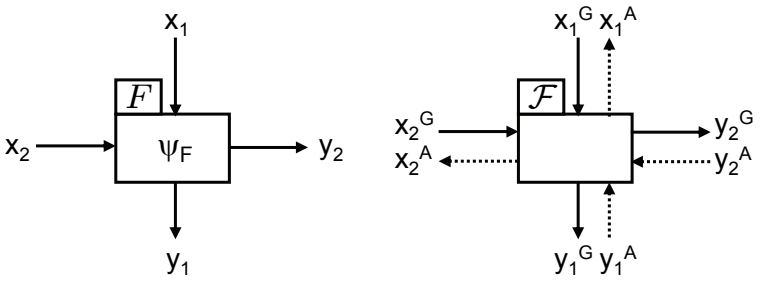

Fig. 2.4: A simple abstract component F and its adaptive interface representation

Figure 2.4 shows an abstract component and the corresponding adaptive interface representation. The abstract component variables represent some abstraction of the actual component behavior, where $X_F = (x_1, x_2)$, $Y_F = (y_1, y_2)$ represent abstractions of its inputs and outputs, respectively. The component works properly if $\Psi_F(X_F)$ is satisfied and it can work properly, if $\Psi_F(X_F)$ is satisfiable.

If we make the transition from an abstract component F to its adaptive real-time interface $\mathcal{F}$, see Fig. 2.4, then we have input and output variables and implicit partial orders. The variables denoted with superscript G are called guaranteed values, $X^G_{\mathcal{F}} \geq X_F$, $Y^G_{\mathcal{F}} \geq Y_F$. Their meaning is that the network of abstract components works properly whenever the abstract component variables are 'smaller' than the guaranteed values in the corresponding network of interfaces. The predicate Ψ_F has been converted into the new assume variables $X^A_{\mathcal{F}}$ (denoted with superscript A). Their meaning is that whenever we have $X^A_{\mathcal{F}} \geq X^G_{\mathcal{F}}$, then Ψ_F is satisfied[6]. The adaptive interface makes the predicate Ψ_F explicit in the form of additional *input assume* variables. They also appear at outputs of other adaptive interfaces as *output assume* variables. Therefore, the following interpretations can be given:

- *Input assume variables*: The input assume variables $X^A_{\mathcal{F}}$ describe the assumption of the component toward the environment or other components. If we have $X^A_{\mathcal{F}} \geq X^G_{\mathcal{F}}$, then (a) the corresponding component works properly and (b) the requests of all connected components (indicated by Y^A_F) are satisfied, i.e. $Y^A_{\mathcal{F}} \geq Y^G_{\mathcal{F}}$.

- *Output assume variables*: The output assume variables $Y^A_{\mathcal{F}}$ describe the assumption of the environment or other components toward F, i.e. they request that $Y^A_{\mathcal{F}} \geq Y^G_{\mathcal{F}}$.

[6]One may extend the framework and consider explicitly predicates in the abstract interface also, see Section 2.5.2.

With this framework a particular combination of assumptions and guarantees can be determined for each component which will reflect current system inputs, constraints and requirements.

Next we will see how input and output assume and guarantee variables are connected by transfer functions. This way, it is possible to represent adaptive behavior of a component, e.g. if it changes its behavior depending on *local* requirements, this will be reflected in the input and output assumes and guarantees. Moreover, one can decide whether the whole system is able to meet constraints and under what assumptions on its inputs, thereby solving important synthesis questions.

2.4.1 Single Adaptive Interface

Now we can state the formal definition of an adaptive real-time interface.

Definition 2.12: (Adaptive Real-Time Interface) *An* adaptive interface $(X^A, X^G, Y^A, Y^G, T^f, T^b)$ *corresponds to a monotone abstract component* (X, Y, T, Ψ) *with at least one input and is characterized as follows:*

- *A set of input guarantee and input assume variables X^G and X^A, one for each variable in the inputs X of the abstract component. The input guarantee variables X^G are inputs for the adaptive interface, and the input assume variables X^A are outputs for the adaptive interface.*
- *A set of output guarantee and output assume variables Y^G and Y^A, one for each variable in the outputs Y of the abstract component. The output guarantee variables Y^G are outputs for the adaptive interface, and the output assume variables Y^A are inputs for the adaptive interface.*
- *A monotone forward transfer function with $Y^G = T^f(X^G)$.*
- *A backward transfer function with $X^A = T^b(X^G, Y^A)$.*

In addition, we require that:

$$X^A \geq X^G \geq X \Rightarrow Y^A \geq Y^G \geq Y \wedge \Psi(X) \; .$$

According to the above definition, an adaptive interface has a pair of variables (one is an input the other one is an output) for each variable of the corresponding abstract component. The outputs of the adaptive interface, X^A and Y^G, are computed using the forward and backward transfer functions. Therefore, the input assumptions and output guarantees, X^A and Y^G, adapt depending on the propagated requirements and constraints from the other components and the environment. Note that

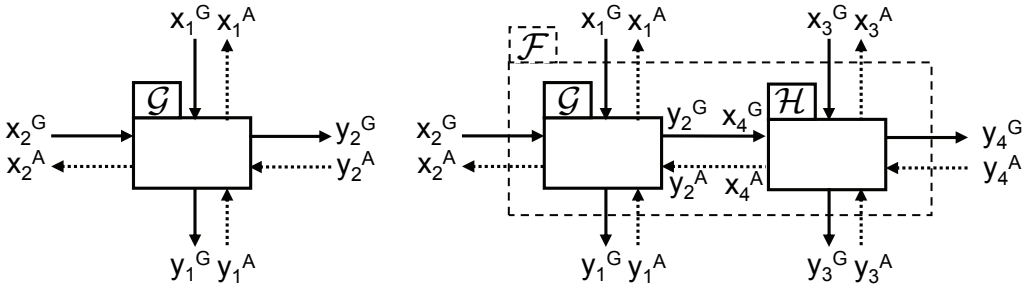

Fig. 2.5: Adaptive interfaces that correspond to the abstract component and the network of abstract components shown in Fig. 2.1 and Fig. 2.2, respectively

the underlying abstract component may also represent a whole set of connected components.

The condition that establishes the relation between an abstract component and its adaptive interface reads as follows: If the input values of the abstract component as well as the corresponding assumptions and guarantees match (using the partial order assigned to each single input), then the same holds for the output values and the predicate of the abstract component is satisfied. Later on, we will determine T^f and T^b such that this requirement is satisfied. But at first, let us illustrate Definition 2.12 with a simple network of two abstract components and their respective interfaces.

Example 2.13: *Figure 2.5 shows on the left hand side an adaptive interface that corresponds to the abstract component shown in Fig. 2.1.*

We have $X^G_{\mathcal{G}} = (x^G_1, x^G_2)$, $X^A_{\mathcal{G}} = (x^A_1, x^A_2)$, $Y^G_{\mathcal{G}} = (y^G_1, y^G_2)$ *and* $Y^A_{\mathcal{G}} = (y^A_1, y^A_2)$. *Let us use the forward and backward transfer functions* $(y^G_1, y^G_2) = T^f_{\mathcal{G}}(x^G_1, x^G_2) = (x^G_1, x^G_2 - x^G_1)$ *and* $(x^A_1, x^A_2) = T^b_{\mathcal{G}}(x^G_1, x^G_2, y^A_1, y^A_2) = (\min\{x^G_2, y^A_1\}, \max\{x^G_1, y^A_2 + x^G_1\})$. *Then we can easily check that* $X^A_{\mathcal{G}} \geq X^G_{\mathcal{G}} \geq X_{\mathcal{G}} \Rightarrow Y^A_{\mathcal{G}} \geq Y^G_{\mathcal{G}} \geq Y_{\mathcal{G}} \wedge \Psi_{\mathcal{G}}(X_{\mathcal{G}})$, *i.e. if the assumptions and guarantees of the inputs match* ($x^A_1 \geq x^G_1$,$x^A_2 \leq x^G_2$), *then those of the outputs also match* ($y^A_1 \geq y^G_1$,$y^A_2 \leq y^G_2$), *and the predicate of the abstract component is satisfied* ($\Psi_{\mathcal{G}}(X^G_{\mathcal{G}}) = (x^G_2 \geq x^G_1)$).

2.4.2 Connecting Adaptive Interfaces

Now we can proceed with the formal definition of the connection of two interfaces $\mathcal{G}$ and $\mathcal{H}$.

Definition 2.14: (Connection of Adaptive Real-Time Interfaces) *The* connection *of two interfaces* $\mathcal{G} \parallel \mathcal{H}$ *follows the connection of the corresponding abstract components, i.e.* $G \parallel H$. *In particular, we have:*

- *If an output y of an abstract component is connected to an input x with $x = y$, then the corresponding assume and guarantees are identical too: $x^G = y^G$ and $y^A = x^A$.*
- *The inputs and outputs of the adaptive interface $\mathcal{F} = \mathcal{G} \parallel \mathcal{H}$ are given by $X^A_{\mathcal{F}} = (X^A_{\mathcal{G}} \cup X^A_{\mathcal{H}}) \backslash (Y^A_{\mathcal{G}} \cup Y^A_{\mathcal{H}})$, $Y^A_{\mathcal{F}} = (Y^A_{\mathcal{G}} \cup Y^A_{\mathcal{H}}) \backslash (X^A_{\mathcal{G}} \cup X^A_{\mathcal{H}})$, $X^G_{\mathcal{F}} = (X^G_{\mathcal{G}} \cup X^G_{\mathcal{H}}) \backslash (Y^G_{\mathcal{G}} \cup Y^G_{\mathcal{H}})$, and $Y^G_{\mathcal{F}} = (Y^G_{\mathcal{G}} \cup Y^G_{\mathcal{H}}) \backslash (X^G_{\mathcal{G}} \cup X^G_{\mathcal{H}})$.*
- *The forward and backward transfer functions $T^f_{\mathcal{F}}$ and $T^b_{\mathcal{F}}$ are determined by composing $T^f_{\mathcal{G}}$ with $T^f_{\mathcal{H}}$, and $T^b_{\mathcal{G}}$ with $T^b_{\mathcal{H}}$, respectively, following the connections of the interfaces inputs and outputs, see Definition 2.3.*

The above definition can easily be extended to the case of a network of adaptive interfaces as in the case of abstract components, see Definition 2.3. The construction of the forward and backward transfer functions T^f and T^b, according to Definition 2.14, by a simple concatenation of functions requires that there are no dependency cycles. The following theorem proves that this is actually the case if the corresponding network of abstract components does not have directed cycles either.

Theorem 2.15: (Transfer Functions for Networks of Adaptive Real-Time Interfaces) *Given a network of monotone abstract components $\mathbb{F}$ whose connection graph is free of directed cycles and a partitioning $\mathbb{F} = \mathbb{G} \parallel \mathbb{H}$. Interfaces $\mathcal{G}$ and $\mathcal{H}$ correspond to the subnetworks $\mathbb{G}$ and $\mathbb{H}$, respectively. Then $T^f_{\mathcal{F}}$ and $T^b_{\mathcal{F}}$ can be determined without dependency cycles, i.e. by simple function composition (concatenation).*

Proof. All guarantee variables can be determined with simple concatenation of forward transfer functions as the connection graph of $\mathbb{F}$ does not contain directed cycles and the interconnection of the corresponding adaptive interfaces follows this structure, and as $Y^G = T^f(X^G)$. The assume variables are determined according to $X^A = T^b(X^G, Y^A)$. Their values depend on the guarantee variables (whose values can be determined without considering dependency cycles) and on assume variables. Because of Definition 2.14, the determination of the assume variables again follows the connection graph, but now with all edges reversed. As this graph also does not contain directed cycles, the theorem follows.

■

From Definition 2.14, we know how to combine a set of adaptive interfaces to a new adaptive interface. In addition, Definition 2.12 states

the condition, that an adaptive interface actually represents an abstract component, i.e. that it can work properly. The above concepts are useful only, if we are able to show that the combined adaptive interface contains enough information to decide whether all abstract components it represents can work properly together.

Theorem 2.16: (Compatibility for Adaptive Real-Time Interfaces) *The composition of two interfaces $\mathcal{F} = \mathcal{G} \parallel \mathcal{H}$ is called compatible ($\mathcal{G} \sim \mathcal{H}$), iff $X^A_{\mathcal{F}} \geq X^G_{\mathcal{F}}$ is satisfiable. Then we have $(\mathcal{G} \sim \mathcal{H}) \Rightarrow (\mathbb{G} \sim \mathbb{H})$, i.e. the corresponding abstract components also can work properly together. In particular, we have:*

$$X^A_{\mathcal{F}} \geq X^G_{\mathcal{F}} \geq X_{\mathbb{F}} \Rightarrow Y^A_{\mathcal{F}} \geq Y^G_{\mathcal{F}} \geq Y_{\mathbb{F}} \wedge \Psi_{\mathbb{F}}(X_{\mathbb{F}}) \,.$$

Proof. Let us first consider two connected abstract components M and N, i.e. $\mathbb{K} = M \parallel N$. As the connection graph does not contain any directed cycles, we can suppose that only outputs of M are connected to inputs of N. The corresponding adaptive interfaces are connected correspondingly. For $\mathcal{M}$ we have $X^A_{\mathcal{M}} \geq X^G_{\mathcal{M}} \geq X_M \Rightarrow Y^A_{\mathcal{M}} \geq Y^G_{\mathcal{M}} \geq Y_M \wedge \Psi_M(X_M)$. As some of the outputs of M are connected to some inputs of N and the other inputs of N are inputs of $\mathbb{K}$, we also have $X^A_{\mathcal{K}} \geq X^G_{\mathcal{K}} \geq X_{\mathbb{K}} \Rightarrow X^A_{\mathcal{N}} \geq X^G_{\mathcal{N}} \geq X_N \wedge \Psi_N(X_N)$. If we combine the arguments for the two components, we obtain $X^A_{\mathcal{K}} \geq X^G_{\mathcal{K}} \geq X_{\mathbb{K}} \Rightarrow Y^A_{\mathcal{K}} \geq Y^G_{\mathcal{K}} \geq Y_{\mathbb{K}} \wedge \Psi_{\mathbb{K}}(X_{\mathbb{K}})$. The same argument can be now recursively applied to the case of interfaces that represent sub-networks of abstract components.

■

The theorem now states that it is sufficient to look only at the input guarantees and assumptions of an adaptive interface in order to determine whether the corresponding abstract components work properly together. That is, if we have $X^A_{\mathcal{F}} \geq X^G_{\mathcal{F}}$, then we can also conclude that all output assumptions from the environment are satisfied as well as the predicates of the abstract components.

Example 2.17: *We are continuing Example 2.13 by connecting the adaptive interfaces $\mathcal{G}$, $\mathcal{H}$ of the abstract components G, H according to the network $\mathbb{F}$ shown in Fig. 2.2. The resulting interface $\mathcal{F}$ and its composition $\mathcal{F} = \mathcal{G} \parallel \mathcal{H}$ are shown in Fig. 2.5 on the right hand side. As stated in Theorem 2.15, the resulting forward and backward transfer functions can be computed by simple composition. We obtain $(y^G_1, y^G_3, y^G_4) = T^f(X^G_{\mathcal{F}}) = (x^G_1, x^G_3, x^G_2 - x^G_1 - x^G_3)$ and $(x^A_1, x^A_2, x^A_3) = T^b(X^G_{\mathcal{F}}, Y^A_{\mathcal{F}}) = (\min\{x^G_2, y^A_1\}, \max\{x^G_1, x^G_1 + x^G_3, x^G_1 + x^G_3 + y^A_4\}, \min\{x^G_2 - x^G_1, y^A_3\})$. One can easily check that $X^A_{\mathcal{F}} \geq X^G_{\mathcal{F}}$ is satisfiable and the abstract components are compatible, i.e. can work together properly.*

2.4.3 Constructing Transfer Functions

Now we will construct the forward and backward functions T^f and T^b in such a way that the relation $X^A \geq X^G \geq X \Rightarrow Y^A \geq Y^G \geq Y \wedge \Psi(X)$ in Definition 2.12 holds.

Theorem 2.18: (Properties of Transfer Functions) *If for all X, Y we have $T^f(X) \geq T(X)$ and $X \leq T^b(X, Y) \Rightarrow Y \geq T^f(X) \wedge \Psi(X)$ then $X^A \geq X^G \geq X \Rightarrow Y^A \geq Y^G \geq Y \wedge \Psi(X)$.*

Proof. We have $X^A \geq X^G \geq X \Rightarrow T^b(X^G, Y^A) \geq X^G \wedge T^f(X^G) \geq T(X) \Rightarrow Y^A \geq T^f(X^G) \wedge Y^G \geq Y \wedge \Psi(X^G) \Rightarrow Y^A \geq Y^G \geq Y \wedge \Psi(X)$. Here we make use of the monotonicity of T,T^f, and Ψ, and Definitions 2.1 and 2.12.

■

The above theorem leads to a simple constructive method to determine T^f, namely $T^f = T$. In the case of T^b, we determine one general possibility next. Let the input variables of the abstract component be denoted as $X = (x_1, ..., x_N)$, then the construction of a large but feasible $T^b(X, Y)$ involves the following three steps:

1. Determine a set of N functions $\widetilde{\Psi}_i(X)$ such that $\widetilde{\Psi}_i(X) = \max\{z \mid \Psi(x_1, \ldots, x_{i-1}, z, x_{i+1}, \ldots, x_N)\}$ for all X.
2. Determine a set of N functions $\widetilde{T}_i(X, Y)$ such that $\widetilde{T}_i(X, Y) = \max\{z \mid Y \geq T^f(x_1, \ldots, x_{i-1}, z, x_{i+1}, \ldots, x_N)\}$ for all X, Y.
3. Finally, we have for all i: $T_i^b(X, Y) = \inf\{\widetilde{\Psi}_i(X), \widetilde{T}_i(X, Y)\}$ for all X, Y.

In steps 1 and 2, max denotes a maximal element of a set where the partial order relation of x_i is used. Let us choose some X, Y such that $X \leq T^b(X, Y)$ holds, then Step 3 yields $x_i \leq \widetilde{\Psi}_i(X)$ and $x_i \leq \widetilde{T}_i(X, Y)$. Because of 1 we find that $\Psi(X)$ holds because $\Psi(x_1, ..., x_{i-1}, \tilde{\Psi}_i(X), x_{i+1}, ..., x_N)$ is satisfied for all $1 \leq i \leq N$, $x_i \leq \widetilde{\Psi}_i(X)$, and Ψ is monotone. Because of 2 we find that $Y \geq T(X)$ for all i holds because $Y \geq T(x_1, ..., x_{i-1}, \widetilde{T}_i(X, Y), x_{i+1}, ..., x_N)$, $x_i \leq \widetilde{T}_i(X, Y)$, and T is monotone.

Example 2.19: *The forward and backward transfer functions for $\mathcal{G}$ as used in Example 2.13 have been determined by the above method. Obviously, we had $T^f_{\mathcal{G}} = T_G$, see also Fig. 2.1. Let us show how to construct $x_2^A = T_2^b(X^G_{\mathcal{G}}, Y^A_{\mathcal{G}})$. For step 2 we obtain $\widetilde{T}_{21} = \min\{z \mid y_1^A \geq z\} = -\infty$ and $\widetilde{T}_{22} = \min\{z \mid y_2^A \leq z - x_1^G\} = y_2^A + x_1^G$. For step 1 we obtain $\widetilde{\Psi}_2 = \min\{z \mid z \geq x_1^G\} = x_1^G$. Note that we have to consider the correct partial orders everywhere. Finally, we obtain $x_2^A = T_2^b(X^G_{\mathcal{G}}, Y^A_{\mathcal{G}}) = \sup\{\widetilde{\Psi}_2, \widetilde{T}_{21}, \widetilde{T}_{22}\} = \max\{x_1^G, y_2^A + x_1^G\}$. This is the same expression as used in Example 2.13.*

In Chapter 3 we will show how to determine the transfer functions for several abstract components using Modular Performance Analysis [CKT03b].

2.4.4 Refinement

In a similar way to the refinement of abstract components, we can define the refinement of adaptive interfaces. Let us suppose that a component is implemented together with an implementation of its adaptive interface. This way, at a system house (that combines the independently implemented components) one can connect the interfaces during design time in order to check whether the components work properly for the specific environment and set of constraints. A designer may also explore different implementations of the components and their respective interfaces, and can be sure that they can be connected and work together properly as long as the new interfaces refine the original ones. One may even adapt the assumptions and guarantees of an interface at run-time in order to perform a system-wide admittance test in case the environment changes (adaptive behavior). In this case, the interfaces are actually implemented on the run-time system.

In both cases it would be useful if the implementation of such a combined component/interface can be performed independently, i.e. involving also a possible change in the interface. Such a change may be necessary for example to reflect a simplified implementation. But it must still be guaranteed that if a network of interfaces worked properly for a certain set of input assumptions and output guarantees, then it also works properly after a refinement of the adaptive interface for the same set.

The following arguments follow closely those in Section 2.3.4 but before we proceed, we need to define antitone functions. A function T is called antitone if for all $X, \widetilde{X}$ we have that:

$$X \geq \widetilde{X} \Rightarrow T(X) \leq T(\widetilde{X}) \; .$$

Definition 2.20: (Refinement for Adaptive Real-Time Interfaces) *Given an adaptive interface $\mathcal{G}$. Then $\mathcal{G}'$* refines *$\mathcal{G}$ ($\mathcal{G} \geq \mathcal{G}'$) if:*

- *The sets of input and output variables of $\mathcal{G}$ and $\mathcal{G}'$ are equal.*
- *$T^f_{\mathcal{G}}(X)$ is monotone in X and $T^b_{\mathcal{G}}(X, Y)$ is monotone in Y, antitone in X.*
- *$T^f_{\mathcal{G}}(X) \geq T^f_{\mathcal{G}'}(X)$ and $T^b_{\mathcal{G}}(X, Y) \leq T^b_{\mathcal{G}'}(X, Y)$ for all valuations of variables X, Y.*

From Definition 2.20 it follows that a refined adaptive interface has a stronger forward and a weaker backward transfer function. Now we can state the main refinement theorem.

Theorem 2.21: (Refinement for Adaptive Real-Time Interfaces Preserves Compatibility) *Given adaptive interfaces and a compatible connection, i.e.* $\mathcal{F} = \mathcal{G} \parallel \mathcal{H}$ *and* $\mathcal{G} \sim \mathcal{H}$. *If we refine* $\mathcal{G}$ *to* $\mathcal{G}'$ ($\mathcal{G} \geq \mathcal{G}'$) *then* $\mathcal{F}' = \mathcal{G}' \parallel \mathcal{H}$ *is also a compatible connection, i.e.* $\mathcal{G}' \sim \mathcal{H}$.

Proof. As $\mathcal{G} \sim \mathcal{H}$ we find from Theorem 2.16 that $X^A_{\mathcal{F}} \geq X^G_{\mathcal{F}}$ is satisfiable for some $X^G_{\mathcal{F}}$. After replacing $\mathcal{G}$ by $\mathcal{G}'$ let the variables in the network be denoted as $\widetilde{X}^{G,A}_{\mathcal{G}'}$, $\widetilde{Y}^{G,A}_{\mathcal{G}'}$, $\widetilde{X}^{G,A}_{\mathcal{H}}$, $\widetilde{Y}^{G,A}_{\mathcal{H}}$. Because of the monotonicity of $T^f_{\mathcal{F}}(X)$ and $T^f_{\mathcal{F}}(X) \geq T^f_{\mathcal{F}'}(X)$ we find $\widetilde{X}^G_{\mathcal{G}'} \leq X^G_{\mathcal{G}}$, $\widetilde{Y}^G_{\mathcal{G}'} \leq Y^G_{\mathcal{G}}$, $\widetilde{X}^G_{\mathcal{H}} \leq X^G_{\mathcal{H}}$, and $\widetilde{Y}^G_{\mathcal{H}} \leq Y^G_{\mathcal{H}}$. In a similar way, because the backward functions are monotone in the guarantee and antitone in the assume inputs, and because $T^b_{\mathcal{F}}(X,Y) \leq T^b_{\mathcal{F}'}(X,Y)$, we find $\widetilde{X}^A_{\mathcal{G}'} \geq X^A_{\mathcal{G}}$, $\widetilde{Y}^A_{\mathcal{G}'} \geq Y^A_{\mathcal{G}}$, $\widetilde{X}^A_{\mathcal{H}} \geq X^A_{\mathcal{H}}$, and $\widetilde{Y}^A_{\mathcal{H}} \geq Y^A_{\mathcal{H}}$. As a result, we find $X^A_{\mathcal{F}'} \geq X^A_{\mathcal{F}} \geq X^G_{\mathcal{F}} \geq X^G_{\mathcal{F}'}$, and therefore $X^A_{\mathcal{F}'} \geq X^G_{\mathcal{F}'}$.

■

2.4.5 Incremental Design

We would ideally expect that the incremental design property for networks of abstract components holds also in the case of networks of adaptive interfaces. Let us suppose that we have a network of abstract components, that can work properly. Summarizing the results we obtained so far, we can draw the following conclusions:

- Any subnetwork of abstract components can also work properly, see Theorem 2.7.
- If we connect two subnetworks $\mathbb{G}$ and $\mathbb{H}$ that are represented by adaptive interfaces $\mathcal{G}$ and $\mathcal{H}$, then $(\mathcal{G} \sim \mathcal{H}) \Rightarrow (\mathbb{G} \sim \mathbb{H})$, see Theorem 2.16. In other words, compatibility of interfaces guarantees that the associated components can work properly together, but not the other way round.

Therefore, the property of incremental design does not hold for adaptive interfaces as defined so far. Composition of adaptive interfaces is not associative and therefore, the success of a design (in terms of compatibility of interfaces) depends on the ordering of compositions.

On the other hand, if we would have a one-to-one correspondence between the compatibility of adaptive components and interfaces, then the property stated in Theorem 2.7 would carry over to adaptive interfaces.

Definition 2.22: (Strict Adaptive Real-Time Interface) *A* strict adaptive interface *is an adaptive interface according to Definition 2.12 that satisfies in addition the following property:*

$$Y^A \geq Y^G \wedge \Psi(X^G) \Rightarrow X^A \geq X^G.$$

Now, we can state the necessary strong relation between abstract components and strict adaptive interfaces.

Theorem 2.23: (Equivalence of Compatibility for Abstract Components and Strict Adaptive Real-Time Interfaces) *Given a composition of two strict interfaces $\mathcal{F} = \mathcal{G} \parallel \mathcal{H}$ where the corresponding network of components $\mathbb{F}$ does not have any outputs*[7]*. Then we have $(\mathcal{G} \sim \mathcal{H}) \iff (\mathbb{G} \sim \mathbb{H})$.*

Proof. The forward direction $(\mathcal{G} \sim \mathcal{H}) \Rightarrow (\mathbb{G} \sim \mathbb{H})$ has already been proved in Theorem 2.16. Let us first consider two connected abstract components A and B, i.e. $\mathbb{C} = A \parallel B$. As the connection graph does not contain any directed cycles, we can suppose that only outputs of A are connected to inputs of B. The corresponding adaptive interfaces are connected correspondingly. For $\mathcal{B}$ we have $Y^A_{\mathcal{B}} \geq Y^G_{\mathcal{B}} \wedge \Psi_B(X^G_{\mathcal{B}}) \Rightarrow X^A_{\mathcal{B}} \geq X^G_{\mathcal{B}}$. As some of the outputs of A are connected to some inputs of B and the other outputs of A are outputs of $\mathbb{C}$, we also have $Y^A_C \geq Y^G_C \wedge \Psi_{\mathbb{C}}(X^G_C) \Rightarrow Y^A_{\mathcal{A}} \geq Y^G_{\mathcal{A}} \wedge \Psi_A(X^G_{\mathcal{A}}) \Rightarrow X^A_{\mathcal{A}} \geq X^G_{\mathcal{A}}$. As the inputs of C contain inputs of $\mathcal{A}$ and $\mathcal{B}$, we finally have $Y^A_C \geq Y^G_C \wedge \Psi_{\mathbb{C}}(X^G_C) \Rightarrow X^A_C \geq X^G_C$. The same argument can now be recursively applied to the case of interfaces that represent subnetworks of abstract components.

■

As a result of Theorem 2.23, there is a one-to-one correspondence between the compatibility relations for interfaces and the corresponding underlying network of abstract components. Therefore, if adaptive interfaces are strict, then they allow for incremental design as defined in [dAH05].

It remains to show, how we can construct the transfer functions of a strict adaptive interface, and when they exist.

[7] This technical condition ensures that we do not impose additional assume/guarantee constraints from the environment that are not present in the corresponding network of abstract components. This is no restriction of generality as one could close open outputs with additional abstract components.

Theorem 2.24: (Strict Forward and Backward Transfer Functions) *We call T^f and T^b strict if they satisfy the conditions of Theorem 2.18 and in addition $\Psi(X) \Rightarrow T^b(X, T(X)) \geq X$ for all X. In this case we have $X^A \geq X^G \iff Y^A \geq Y^G \wedge \Psi(X^G)$.*

Proof. The forward direction has already been proved in Theorem 2.18. Here we show: $Y^A \geq Y^G \wedge \Psi(X^G) \Rightarrow T^b(X^G, Y^A) \geq T^b(X^G, Y^G) \wedge \Psi(X^G) \Rightarrow X^A \geq T^b(X^G, Y^G) \wedge \Psi(X^G) \Rightarrow X^A \geq T^b(X^G, T(X^G)) \wedge \Psi(X^G) \Rightarrow X^A \geq T^b(X^G, T(X^G)) \geq X^G$. Here we use the monotonicity of $T^b(X, Y)$ in Y and Definition 2.12.

■

Finally, we will give a constructive method to determine strict forward and backward transfer functions from T and Ψ. It follows directly the approach taken in Section 2.4.3 but replaces 'a maximal' element by 'the greatest' element:

$$\begin{aligned} T^f(X) &= T(X) \,, \\ \widetilde{\Psi}_i(X) &= grt\{z \mid \Psi(x_1, ..., x_{i-1}, z, x_{i+1}, ..., x_N)\} \,, \\ \widetilde{T}_i(X, Y) &= grt\{z \mid Y \geq T(x_1, ..., x_{i-1}, z, x_{i+1}, ..., x_N)\} \,, \\ T^b_i(X, Y) &= \inf\{\widetilde{\Psi}_i(X), \widetilde{T}_i(X, Y)\} \,, \end{aligned}$$

where grt yields the greatest element of a set (if it exists) and inf denotes the infimum of a set. Note that the sets are partially ordered using the partial order associated to each input and output, see Definition 2.8. For example, inf and grt in the above equations use the partial order associated with x_i and the relations in $Y \geq T(x_1, ..., x_{i-1}, z, x_{i+1}, ..., x_N)$ are computed using the partial order associated with each y_j. It also appears, that T^b can only be computed if the greatest elements of the partially ordered sets exist.

We still need to show that the above equations yield a valid strict backward transfer function with $X \leq T^b(X, Y) \Rightarrow Y \geq T(X) \wedge \Psi(X)$ and $\Psi(X) \Rightarrow T^b(X, T(X)) \geq X$. For the first relation, we find:

$$\begin{aligned} X \leq T^b(X, Y) \Rightarrow & \bigwedge_i (x_i \leq T^b_i(X, Y)) \Rightarrow \\ & \bigwedge_i (x_i \leq \widetilde{T}_i(X, Y)) \wedge \bigwedge_i (x_i \leq \widetilde{\Psi}_i(X)) \Rightarrow \\ & Y \geq T(X) \wedge \Psi(X) \,. \end{aligned}$$

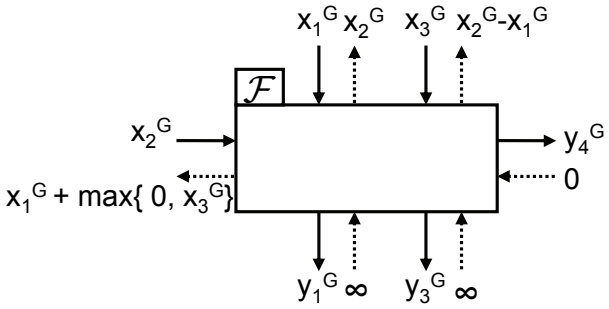

Fig. 2.6: The adaptive interface from Fig. 2.5 but now it shows valuations for the output assumptions and computed values for the input assumptions

Using the definitions of the infimum and greatest element, the second relation can be shown as follows using a proof by contrapositive:

$$
\begin{aligned}
X \not\leq T^b(X, T(X)) \Rightarrow & \bigvee_i (x_i \not\leq T_i^b(X, T(X))) \Rightarrow \\
& \bigvee_i (x_i \not\leq \widetilde{T}_i^b(X, T(X))) \vee \bigvee_i (x_i \not\leq \widetilde{\Psi}_i(X)) \Rightarrow \\
& (T(X) \not\geq T(X)) \vee \overline{\Psi}(X) \Rightarrow \\
& \overline{\Psi}(X) \ .
\end{aligned}
$$

The following example applies the concept of strict transfer functions to our running example.

Example 2.25: *As the forward and backward transfer functions of $\mathcal{G}$ have been already determined in the way described here, they are strict and therefore, the adaptive interface shown in Fig. 2.5 and Example 2.13 is strict too. As a result from Theorem 2.23 we can also conclude that the interface $\mathcal{F}$ as shown on the right hand side of Fig. 2.5 is strict and we have the compatibility relation $(\mathcal{G} \sim \mathcal{H}) \iff (G \sim H)$, see Theorem 2.23. The corresponding transfer functions are given in Example 2.17.*

Now we can show how one can use adaptive interfaces for solving design problems off-line or for adapting to changes in the environment that are either caused by changed requirements (assumptions) or changed properties.

Example 2.26: *Consider Fig. 2.6 which shows the adaptive interface $\mathcal{F}$ again, but now with some (assumed) valuations of the outputs.*

We can summarize some uses of adaptive interfaces at design time and at run-time as follows:

Design Time

Let us suppose, that there are two packet streams with bandwidth requirements x_1^G and x_3^G to be processed, then we can directly compute the requirement toward the bus bandwidth x_2^A. As in the above example, x_2^A does not depend on x_2^G, see Section 2.5.3, we can also set the bus bandwidth to its minimal feasible value $x_2^G = x_1^G + \max\{0, x_3^G\}$, given input demands x_1^G and x_3^G of the traffic streams. After any single change of one of the input guarantees or output assumptions, we need to recalculate all the other assumptions and guarantees.

Run-Time

If the adaptive interface is implemented in the run-time system, the above described process can be used to adapt to new requirements and guarantees of the environment:

1. *Change* one *input guarantee according to a new environment, e.g. adapt the available bus bandwidth or the bandwidth of packet streams while respecting the corresponding assume. The execution of the actual change will be subject to a mode change protocol as will be further elaborated in Chapter 5.*

2. *Recalculate the resulting assumptions in the whole system in order to adapt to the new environment.*

This method can be applied if the underlying network of abstract components is free of undirected cycles, see Section 2.5.3.

We continue with a small example that illustrates the modularity of the whole network. This will be further illustrated with the case study in Section 2.7.

Example 2.27: *Finally, Fig. 2.7 shows a more complex scenario that shows the modularity of the framework described in this chapter.*

Here, we have two independent resources such as a communication unit with bandwidth x_2 and a computing resource with service x_6. Both packet streams pass the communication unit and the computing device. Note that the interfaces could be combined in any order (incremental design property) because of the strict forward and backward transfer functions. As indicated in Fig. 2.7, we could construct interfaces $\mathcal{K}$ and $\mathcal{L}$ that abstract each packet stream (instead of abstracting the resources as in Fig. 2.6).

2.5 Extensions of the Basic Framework

The above framework for interface-based analysis, design, and adaptation of real-time systems can be extended in several ways. The following

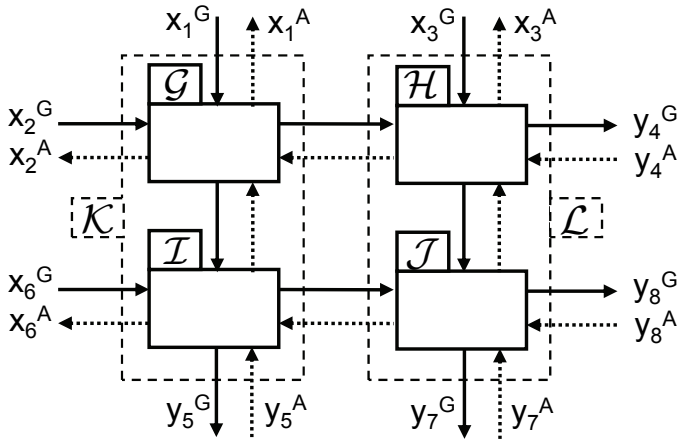

Fig. 2.7: Adaptive interface for a more complex system with two resources

discussion is done in an informal manner and only the major ideas and concepts are briefly presented.

2.5.1 Multiple Assume/Guarantee Pairs

In the adaptive interfaces defined so far, there has been exactly one assume/guarantee pair for each input and output of the corresponding abstract component. One could extend the framework by allowing several of these pairs. This way, it is possible to consider for example upper and lower constraints for a given value. This has been extensively used in the interface-based design with modular performance analysis as will be shown in Chapter 3, see also [WT05b, WT06a].

2.5.2 Additional Predicates in Adaptive Interfaces

Clearly, not all constraints imposed by an environment or by abstract components can be simply phrased into the restricted class of adaptive interfaces that we defined. It is possible to add to each interface additional predicates like the ones defined for abstract components.

2.5.3 Cycle-Free Networks of Abstract Components

We have been restricting the framework to the case where there are no directed cycles in the network of components. We can restrict the scope even more by having the condition that there are no undirected cycles. Then the network of adaptive interfaces has a very interesting and practically important property.

By inspecting the class of cycle-free connection graphs it can easily be shown that in this case for each input and output variable of the network the following statements hold: (1) For any input, the assume value does not change if the corresponding guarantee is changed. (2) For any output, the guarantee value does not change if the corresponding assume is changed.

Consider now the network of adaptive interfaces. We can now select any input guarantee x^G and change it while respecting $x^G \leq x^A$. In other words, for all $x^G \leq x^A$, the system of components still works properly while guaranteeing all constraints. This can be used during design time in order to determine, for example, the maximally allowed input rates or the minimal processor capabilities. If the network of adaptive interfaces is used at run-time, then the change of the environment at a single input can be accepted as long as $x^G \leq x^A$. Afterwards a re-calculation of the interface equations (using forward and backward transfer functions) needs to be performed, and new assumptions toward the environment are calculated.

The same principle does hold for output assumptions also. In particular, the environment can (dynamically) pose a new constraint by changing a single output assume y^A while respecting $y^A \geq y^G$. Examples are changed required properties of the outgoing event streams or required qualities of the remaining resources. Again, such changes can be done either at design time or at run-time. They would also require re-calculation of the interface equations, and the new assumptions should be propagated through the system.

2.5.4 Cycles in Networks of Abstract Components

We have been restricting the scope of the framework to connection graphs that are *free of directed cycles*. Otherwise, for the computation of the transfer function of connected components, fixed point calculations may be necessary. The same holds for the forward and backward transfer functions of the corresponding interfaces also. The present framework provides the necessary algebraic background to formally argue about the existence of these fixed points.

If the network does contain undirected cycles, the framework of Adaptive Real-Time Interfaces as presented here works as described. In contrary to the simple scenario described in Figures 2.5 and 2.6, assumptions and guarantees of a single input or output may depend on each other. In order to find the maximal (in terms of the associated partial order) input guarantee, again a fixed point iteration may be necessary. The investigation of the conditions for convergence are outside the scope of this work.

Note that fixed point calculations in the context of real-time analysis are in use since some time, see [RJE03]. Fixed point calculations for cyclic component networks analyzed with modular performance analysis have been investigated in [JPTY08]. The paper focuses on directed cycles formed by mixed data and resource dependencies. Fixed point calculations for networks with directed cycles in the data flows where the component networks can be represented by marked graphs will be discussed in Chapter 4.

2.6 Unifying Some Common Frameworks for Real-Time Analysis

The presented framework for Adaptive Real-Time Interfaces is powerful enough to unify several existing compositional performance analysis methods and enable them with the concepts of interface-based design. Here we briefly discuss how this can be done.

2.6.1 Hierarchical Scheduling

In [SL04], Shin et al. propose a compositional scheduling framework to determine the schedulability of real-time systems with a set of applications that are scheduled hierarchically. In this framework, the resource demand of a single task is represented as a demand bound function **dbf**, $w \in W$, see [Bar03], and a scheduling component has as input the set of demand bound functions of all tasks that are scheduled by this component. Depending on the associated scheduling strategy, a scheduling component then determines the total demand to schedule all tasks and expresses this again as a demand bound function on its output. Scheduling components can then be composed hierarchically, and the complete system is schedulable if the demand of the scheduling component at the top of the hierarchy can be fulfilled by a dedicated resource. In the context of Real-Time Interfaces, the scheduling components of this framework can be interpreted as abstract components (X, Y, T, Ψ), with a set of inputs $X : x \in W$ and outputs $Y : y \in W$. The transfer function $T(X)$ is applied to compute the outgoing demand bound function, and the predicate $\Psi(X)$ expresses the constraint that a component must be schedulable by a dedicated resource. It can be shown that these abstract components are monotone under the partial order defined as $w \geq \tilde{w} \iff w(t) \geq \tilde{w}(t), \forall t \geq 0$. Based on these abstract components, Adaptive Real-Time Interfaces can then be determined for this scheduling framework following the steps described in this chapter.

2.6.2 SymTA/S

In [RJE03], Richter et al. propose a compositional approach to extend the concepts of classical scheduling theory to heterogeneous distributed systems. In this approach, every single processor or communication link is represented as a component that is analyzed locally. To interconnect the various components, the method relies on a set of standard event arrival patterns that are described as a tuple consisting of a period $p \in P$, a jitter $j \in J$ and a minimum event inter-arrival distance $d \in D$. Based on the arrival patterns of the incoming event streams and on the scheduling policy of the component, the appropriate analysis technique is chosen to compute the worst-case response time of every incoming event stream, and to compute the arrival patterns of the outgoing event streams that will trigger succeeding components. In the context of real-time interfaces, the components of this framework can be interpreted as abstract components (X, Y, T, Ψ), with a set of inputs $X : x = (p, j, d) \in P \times J \times D$ and outputs $Y : y = (p, j, d) \in P \times J \times D$. The transfer function $T(X)$ is applied to compute the outgoing arrival patterns of event streams, and the predicate $\Psi(X)$ expresses the constraints on the maximum allowable response time for every event stream as well as the schedulability of the total component. It can be shown that these abstract components are monotone under the partial order defined as:

$$(p, j, d) \geq (\tilde{p}, \tilde{j}, \tilde{d}) \quad \Longleftrightarrow$$
$$\min\left\{\left\lceil\frac{\Delta + j}{p}\right\rceil, \left\lceil\frac{\Delta}{d}\right\rceil\right\} \geq \min\left\{\left\lceil\frac{\Delta + \tilde{j}}{\tilde{p}}\right\rceil, \left\lceil\frac{\Delta}{\tilde{d}}\right\rceil\right\} \qquad \forall \Delta \in \mathbb{R}^{\geq 0} \; .$$

Based on these abstract components, Adaptive Real-Time Interfaces can then be determined for this compositional framework following the steps described in this chapter.

2.6.3 Modular Performance Analysis

In [CKT03b], an alternative approach for modular performance analysis of real-time embedded system is proposed. In this approach, every task of a system is represented as a component that is analyzed locally. The method is based on arrival curves $\alpha(\Delta) \in \mathcal{A}$ which are a generic event stream model, see [Cru91a, Cru91b, LBT01], and on service curves $\beta(\Delta) \in \mathcal{B}$, a generic resource model. Based on the task processing semantics, a task component relates incoming arrival and service curves that model the input event stream and the available resources to outgoing arrival and service curves that model the output event stream as well as the remaining resources. The analysis method allows to compute delay bounds and

buffer requirements at each task component. Task components are interconnected via their arrival curve inputs and outputs to reflect the flow of data in a system, and via their service curve inputs and outputs to reflect the chosen scheduling policy in a system. In the context of Real-Time Interfaces, task components can also be interpreted as abstract components (X, Y, T, Ψ), with a set of inputs $X : x \in \mathcal{A} \cup \mathcal{B}$ and a set of outputs $Y : y \in \mathcal{A} \cup \mathcal{B}$. The transfer function $T(X)$ equals the internal component relations that are determined according to the processing semantics. The predicate $\Psi(X)$ expresses the constraints on the maximum allowable delay or buffer requirements. It can again be shown that these abstract components are monotone under the partial order defined as $\alpha \geq \tilde{\alpha} \iff \alpha(\Delta) \geq \tilde{\alpha}(\Delta), \forall \Delta \geq 0$ and $\beta \geq \tilde{\beta} \iff \beta(\Delta) \geq \tilde{\beta}(\Delta), \forall \Delta \geq 0$. Based on these abstract components, Adaptive Real-Time Interfaces can again be determined following the steps described in this chapter. Initial Adaptive Real-Time Interfaces for a subset of the modular performance analysis framework are already presented in [WT05b] and [WT06a]. These results will be further extended in Chapter 3 toward distributed systems.

2.7 Case Study

Here we will describe the application of the Adaptive Real-Time Interface framework to the analysis and design of systems with earliest deadline first (EDF) scheduling. It is supposed that the reader is familiar with the basic analysis methods based on supply, demand, and request bound functions as described for example in [SL03, Bar03], and the bounded delay model [MF01]. The models developed here are simplified and their purpose is only to illustrate the use of the Adaptive Real-Time Interfaces framework. Similar results can be shown also for systems with fixed priority scheduling, time division multiple access, servers, or hierarchical scheduling, for details see [WT05b, WT06a, WT06b] and Chapter 3.

2.7.1 Modeling Earliest Deadline First (EDF) Scheduling

First we need to introduce the basic abstract components and their respective interfaces in order to model EDF scheduled systems. Then in Section 2.7.2 we will look closely at a scenario described by Henzinger and Matic in [HM06].

The basic assumptions in our model are the following:

- A single processor characterized by a service curve $\beta(\Delta)$ (also known as supply bound function) which denotes the minimal computation

time available in any time interval of length Δ. For example, a fully available processor with speed 1 is characterized by $\beta(\Delta) = \Delta$.

- A set of tasks τ_i with fixed computation times e_i and relative deadlines D_i.
- The tasks are activated using event streams S_j that are described using arrival curves $\alpha_j(\Delta)$ (also known as request bound functions). Here, $\alpha_j(\Delta)$ denotes the maximal number of events that can arrive in any time interval of length Δ. For example, a periodic event stream with period p_j has $\alpha_j(\Delta) = \lceil \frac{\Delta}{p_j} \rceil$.
- An event stream triggers a chain of tasks, i.e. each event in a stream directly triggers the first task. When an event has been processed by a task, it triggers the next task in the chain, and so on until it is processed by all tasks in the chain.

As an example, we describe one way to model systems with EDF scheduling using the modular performance analysis method as described in [WT06a]. Based on this we will develop the abstract components and their adaptive interfaces. Several basic results about modeling of EDF scheduling will be used here, more details can be found in Chapter 3.

Let us suppose that a task t_i is activated by the events of several event streams, each one characterized by an arrival curve $\alpha_k(\Delta)$, $k \in \mathbf{K}_i$. Then it is activated by the accumulated arrival curve $\alpha_i(\Delta) = \sum_{k \in \mathbf{K}_i} \alpha_k(\Delta)$ which is the sum of the arrival curves of all activating streams. A set of tasks τ_i, $i \in \mathbf{I}$, is schedulable by EDF, iff $\beta(\Delta) \geq \sum_{i \in \mathbf{I}} e_i \cdot \alpha_i(\Delta - D_i)$ for all $\Delta > 0$ where the right hand side is also known as demand bound function[8]. The condition simply states that the total demand for computation time from tasks should always be smaller or equal to the available computation time in any time interval. The output stream of a task τ_i that contains processed events from the stream with arrival curve $\alpha_k(\Delta)$, $k \in \mathbf{K}_i$, is bounded by the arrival curve $\alpha'_k(\Delta) = \alpha_k(\Delta + (D_i - e_i))$ for any $k \in \mathbf{K}_i$.

Using these well-known facts, we can now define basic abstract components and use them to build networks of abstract components that correspond to any particular scenario for systems with EDF scheduling. To this end, we consider the two basic abstract components as shown in Fig. 2.8 on the left hand side.

The top component G describes the use of the available service r to process a task characterized by a relative deadline D and computation time e. Therefore, variable r carries a service curve $\beta(\Delta)$, and a carries the accumulated arrival curves $\alpha(\Delta)$ activating the task. According to the above analysis, the output c carries $\beta(\Delta) - e \cdot \alpha(\Delta - D)$ which now is

[8] We can suppose here that $\alpha(\Delta) = 0$ for all $\Delta \leq 0$.

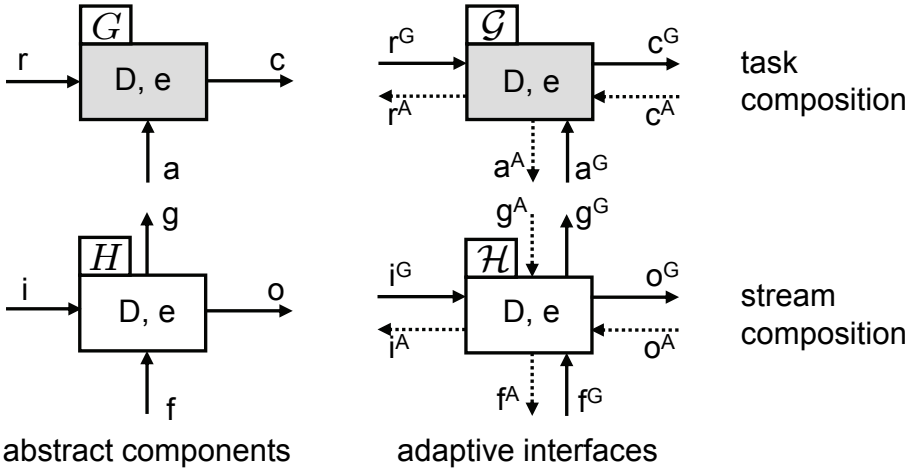

Fig. 2.8: Basic abstract components and interfaces for EDF scheduling

the transfer function of the abstract task component, and represents the unused service that is available for other components connected to output c. The predicate of the abstract task component is $\beta(\Delta) \geq e \cdot \alpha(\Delta - D)$ which represents the EDF schedulability condition. Note that the above transfer function and predicate are monotone with respect to the partial order $\leq$ for r, c where $a \leq b$ iff $a(\Delta) \leq b(\Delta)$, $\forall \Delta > 0$ and $\geq$ for a (defined in a similar way). Defining the operator $\triangleright$ as:

$$(a \triangleright D)(\Delta) = \begin{cases} a(\Delta - D) & \Delta > \max(D, 0) \\ 0 & 0 \leq \Delta \leq \max(D, 0) \end{cases}$$

allows us to write more compactly the transfer function and the predicate as follows:

$$c = r - e \cdot (a \triangleright D) \,, \qquad \Psi = (r \geq e \cdot (a \triangleright D)) \,. \tag{2.1}$$

The bottom abstract component H in Fig. 2.8 denotes the processing of a stream with arrival curve $\alpha(\Delta)$ which is associated to input i. The output arrival curve o can be determined as $\alpha(\Delta + (D - e))$ and g denotes the accumulation of arrival curves of all streams that activate the task, i.e. from the new input i and from other inputs accumulated in f. Using the above shorthand notation, we find:

$$o = i \triangleright (e - D) \,, \qquad g = i + f \,. \tag{2.2}$$

Besides these (monotone) transfer functions, there is no predicate associated to a stream component.

By combining the above two classes of components according to a given scenario, we get a network of abstract components that can be used as described in the previous section. An example is given in Fig. 2.9 on the left hand side.

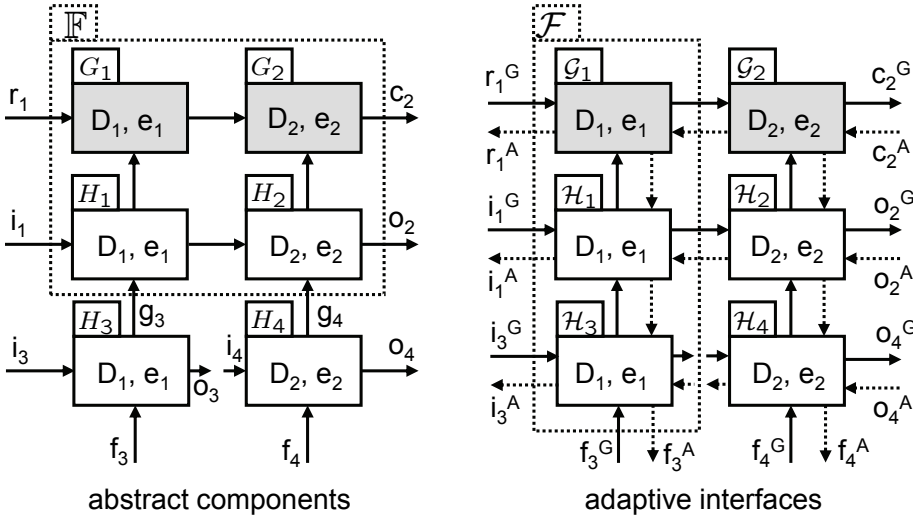

Fig. 2.9: Example of a network of abstract components and interfaces for EDF scheduling

The scenario consists of two tasks τ_1 and τ_2 (there are two task components). There are 3 event streams, namely i_1, i_3, and i_4, where i_1 at first passes τ_1 and then τ_2. Task τ_1 is triggered by i_1 and i_3, and task τ_2 is triggered by i_1 (after passing τ_1) and by i_4. Because of the independent implementability property, see Theorem 2.7, any partitioning in sub-networks is possible (commutativity and associativity of composition). For example, $\mathbb{F}$ denotes an abstract component that describes the resource usage by the two tasks including the processing of stream i_1. The corresponding transfer function and predicate are obtained by a (trivial) composition of the individual transfer functions of the used abstract components G_1, G_2, H_1 and H_2.

In order to allow for solving synthesis problems and propagating constraints, we can also construct the adaptive interfaces corresponding to the abstract components, see Fig. 2.8 on the right hand side. According to Theorem 2.24 and the subsequent method, we can construct strict forward and backward transfer functions.

To this end, we need to define the backward transfer functions according to Theorems 2.18 and 2.24. Whereas this is simple for the usual addition and subtraction operators in (2.1) and (2.2), the operator $\triangleright$ is more difficult to handle. If b is the greatest positive function such that $b \triangleright D \leq a$ then we denote $b = a \triangleleft D$ and find:

$$(a \triangleleft D)(\Delta) = \begin{cases} a(\Delta + D) & \Delta > \max(-D, 0) \\ a(0^+) & 0 < \Delta \leq \max(-D, 0) \\ 0 & \Delta = 0 \end{cases}$$

where $a(0^+)$ denotes $\lim_{\epsilon \downarrow 0} a(\epsilon)$.

Then we can determine the forward and backward transfer functions of the adaptive task interface $\mathcal{G}$ shown in Fig. 2.8 as:

$$c^G = r^G - e \cdot (a^G \triangleright D) \,, \qquad a^A = \frac{1}{e}((r^G - c^A) \triangleleft D) \,, \qquad r^A = c^A + e \cdot (a^G \triangleright D) \,.$$

For the adaptive stream interface $\mathcal{H}$ we find:

$$g^G = i^G + f^G \,, \qquad o^G = i^G \triangleright (e - D) \,, \qquad f^A = g^A - i^G \,,$$
$$i^A = \min\{g^A - f^G, o^A \triangleleft (e - D)\} \,.$$

The above forward and backward transfer functions are strict and therefore, the composition of the adaptive interfaces is associative and commutative which support the independent implementability property. Note that the method described in [HM06] does not guarantee this property. Figure 2.9 presents the same scenario, but now with a different hierarchical component, namely $\mathcal{F}$ which abstracts all activations of task τ_1 and its influence on the system behavior. Note that the combined interface can simply be constructed by composing the forward and backward transfer functions. The system environment can be closed by setting $f_3^G = f_4^G = 0$, $o_2^A = o_3^A = o_4^A = \infty$ and $c_2^A = 0$ for example. Because of the adaptivity of the representation, the suitability of any other environment with other guarantees and assumptions can be easily checked.

Finally, one should note, that the same example could also be handled with a different, more specific, abstraction of resources and event streams, based on the well-known *bounded delay model* introduced in [MF01]. In this case, the structure as shown in Figures 2.8 and 2.9 remains the same, only the data types associated with the variables (the abstraction used in the analysis) as well as the partial orders, transfer functions, and predicates change. The bounded delay model is a much coarser abstraction than general service curves and therefore, the bounds obtained are worse. On the other hand, the analysis and synthesis is computationally much simpler. Figure 2.10 shows the basic abstractions used in the bounded delay model.

The arrival curve corresponding to a periodic event stream with period p is shown in Fig. 2.10(a). Such an arrival curve, but also any other one, can be abstracted using two numbers only, namely burst σ and rate ρ, i.e. $\sigma + \rho \cdot \Delta$, as shown in Fig. 2.10(b). For the periodic task we obtain $\sigma = 1$, $\rho = 1/p$. In a similar way, we can abstract any service curve by the one shown in Fig. 2.10(c) using latency Θ and rate R, i.e. $\max\{0, (\Delta - \Theta) \cdot R\}$. Using the new abstraction, we can now simply replace the transfer functions, the predicates, and the forward and backward

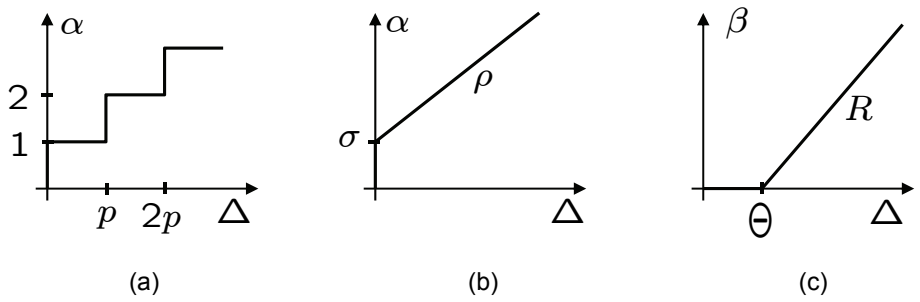

Fig. 2.10: Abstractions used in the bounded delay model: **(a)** a periodic arrival curve that is abstracted by **(b)** a burst and a rate, and **(c)** a rate and a latency resource model that can abstract any service curve

transfer functions. The valuations of variables for r and c are now tuples (Θ, R) with the partial order $\leq$ defined as $a \leq b$ iff $\Theta_a \leq \Theta_b \wedge R_a \leq R_b$. Similarly for variables a, i, o, f, and g, we have the tuples (σ, ρ) with the partial order $\geq$ defined as $a \geq b$ iff $\sigma_a \geq \sigma_b \wedge \rho_a \geq \rho_b$. The transfer functions and predicates can now simply be determined by combining the results described in this chapter and [HM06].

2.7.2 A Comprehensive EDF Example

In this section, we will provide some numerical results for the EDF scenario described in [HM06], i.e. a real-time robotic application adapted from [HKL94]. We have a set of 13 tasks τ_i with $i = 1, \ldots, 13$, computation times e_i and deadlines d_i. They are activated using 5 periodic streams S_k with the nominal periods p_k for $k = 1, \ldots, 5$. The corresponding numbers are given in Tables 2.1 and 2.2.

Tab. 2.1: Task computation times and deadlines for the comprehensive EDF example

τ	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7	τ_8	τ_9	τ_{10}	τ_{11}	τ_{12}	τ_{13}
e	0.2	1.2	1.0	1.0	2.0	0.3	0.8	1.2	1.0	0.5	0.5	0.1	0.5
D	10.48	32.05	21.56	5.79	16.37	9.88	2.30	5.71	3.33	7.51	5.17	1.55	4.86

Tab. 2.2: Stream nominal event periods for the comprehensive EDF example

S	S_1	S_2	S_3	S_4	S_5
p	40	20	5	10	4

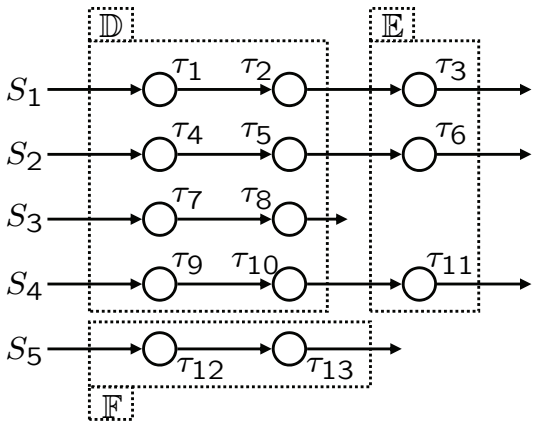

Fig. 2.11: Task chains in the EDF example and the corresponding partitioning into the abstract components $\mathbb{D}$, $\mathbb{E}$, and $\mathbb{F}$

The structure of the application is shown in Fig. 2.11. Stream S_1 activates the task chain $\tau_1 \rightarrow \tau_2 \rightarrow \tau_3$, stream S_2 activates $\tau_4 \rightarrow \tau_5 \rightarrow \tau_6$ and so forth. A possible partitioning of the tasks into abstract components $\mathbb{D}$, $\mathbb{E}$ and $\mathbb{F}$ is also shown. Note that this is an informal representation only as only the tasks are shown here and not the complete abstract components that consist of task and stream subcomponents according to Figures 2.8 and 2.9. The Adaptive Real-Time Interfaces $\mathcal{D}$, $\mathcal{E}$, and $\mathcal{F}$, corresponding to the partitioning shown in Fig. 2.11, are shown in Fig. 2.12.

In terms of an incremental design, let us first connect the interfaces corresponding to the abstract components $\mathbb{D}$ and $\mathbb{E}$, i.e. we form $\mathbb{D} \parallel \mathbb{E}$.

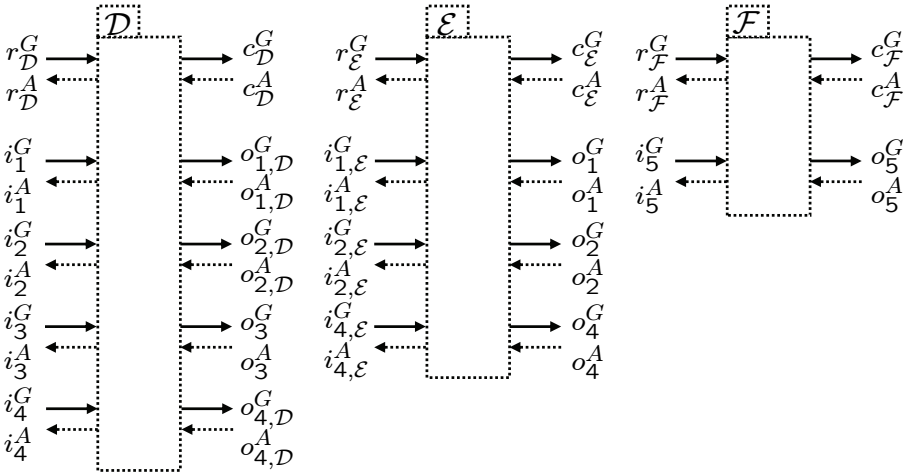

Fig. 2.12: Adaptive Real-Time Interfaces $\mathcal{D}$, $\mathcal{E}$, and $\mathcal{F}$ corresponding to the partitioning in Fig. 2.11

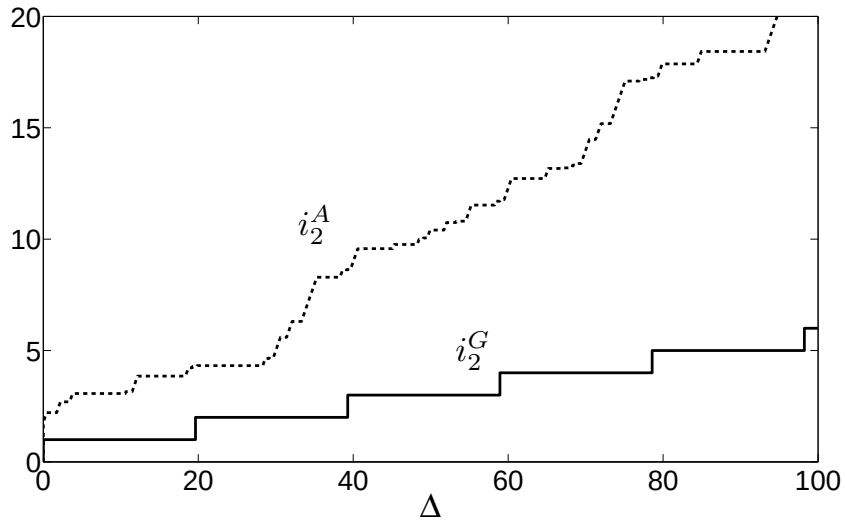

Fig. 2.13: Assume and guarantee arrival curves for stream S_2 if only tasks $\tau_1, ...\tau_{11}$ are processed, i.e. if interfaces $\mathcal{D}$ and $\mathcal{E}$ are connected

We would like to know whether there is an environment such that all real-time constraints are satisfied when processing tasks $\tau_1, ...\tau_{11}$. Therefore, we connect the stream-related connectors of the adaptive interfaces $\mathcal{D}$ and $\mathcal{E}$ as follows, see also Figures 2.8 and 2.9: $i^G_{1,\mathcal{E}} = o^G_{1,\mathcal{D}}$, $o^A_{1,\mathcal{D}} = i^A_{1,\mathcal{E}}$, $i^G_{2,\mathcal{E}} = o^G_{2,\mathcal{D}}$, $o^A_{2,\mathcal{D}} = i^A_{2,\mathcal{E}}$, $i^G_{4,\mathcal{E}} = o^G_{4,\mathcal{D}}$, and $o^A_{4,\mathcal{D}} = i^A_{4,\mathcal{E}}$. In addition, we connect the resource-related interfaces according to $r^G_{\mathcal{E}} = c^G_{\mathcal{D}}$ and $c^A_{\mathcal{D}} = r^A_{\mathcal{E}}$. Now, we connect the combined interfaces to the weakest environment by setting $i^G_k = \alpha_k(\Delta) = \lceil \frac{\Delta}{p_k} \rceil$, $o^A_k = \infty$ for $k = 1, \ldots, 4$, $r^G_{\mathcal{D}} = \beta(\Delta) = \Delta$ and $c^A_{\mathcal{E}} = 0$.

In order to check, whether components $\mathbb{D}$ and $\mathbb{E}$ work together properly, we just need to compute the interface equations as described in the previous section, i.e. those related to the adaptive interface shown in the right hand side of Fig. 2.8. It turns out that the components can work properly, as all interface relations are satisfied. For example, Fig. 2.13 shows that $i^G_2 \leq i^A_2$, i.e. the input arrival curve of stream S_2 is always smaller or equal than what can be properly processed by the combined components $\mathbb{D}$ and $\mathbb{E}$. The area between the two curves represents the remaining 'headroom' for the arrival curve of S_2.

If we also connect component $\mathbb{F}$ in isolation to a weak environment by setting $i^G_5 = \alpha_5(\Delta) = \lceil \frac{\Delta}{p_5} \rceil$, $r^G_{\mathcal{F}} = \beta(\Delta) = \Delta$, $c^A_{\mathcal{F}} = 0$ and $o^A_5 = \infty$, we can observe by checking the interface relations that this component also works properly, i.e. tasks τ_{12} and τ_{13} can be executed on the processor *if no other tasks are running*.

But if try to compose $F \parallel (D \parallel E)$ and compare the assume $r^A_{\mathcal{F}}$ of interface $\mathcal{F}$ with the guarantee $c^G_{\mathcal{E}}$ of interface $\mathcal{E}$, which is the remaining resource available for component $\mathbb{F}$, then we observe that $r^A_{\mathcal{F}} \not\leq c^G_{\mathcal{E}}$, see

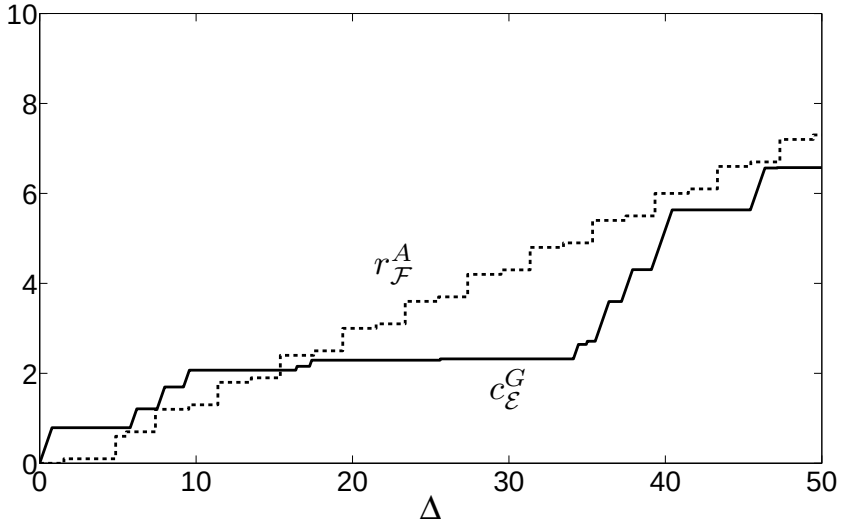

Fig. 2.14: Resource assume $r^A_{\mathcal{F}}$ of interface $\mathcal{F}$ and resource guarantee $c^G_{\mathcal{E}}$ of $\mathcal{D} \parallel \mathcal{E}$ show that the components will not work together properly as $r^A_{\mathcal{F}} \not\leq c^G_{\mathcal{E}}$

Fig. 2.14. In other words, the whole system consisting of all tasks $\tau_1, \ldots, \tau_{13}$ will not work properly. This is checked only by comparing the interfaces of the components.

Indeed, if we connect $\mathbb{D}$, $\mathbb{E}$ and $\mathbb{F}$ to the network $\mathbb{D} \parallel \mathbb{E} \parallel \mathbb{F}$ by setting $r^G_{\mathcal{F}} = c^G_{\mathcal{E}}$ and $c^A_{\mathcal{E}} = r^A_{\mathcal{F}}$ and evaluate all interface equations, we can see that the interface relations are not satisfied. For example, Fig. 2.15 shows that

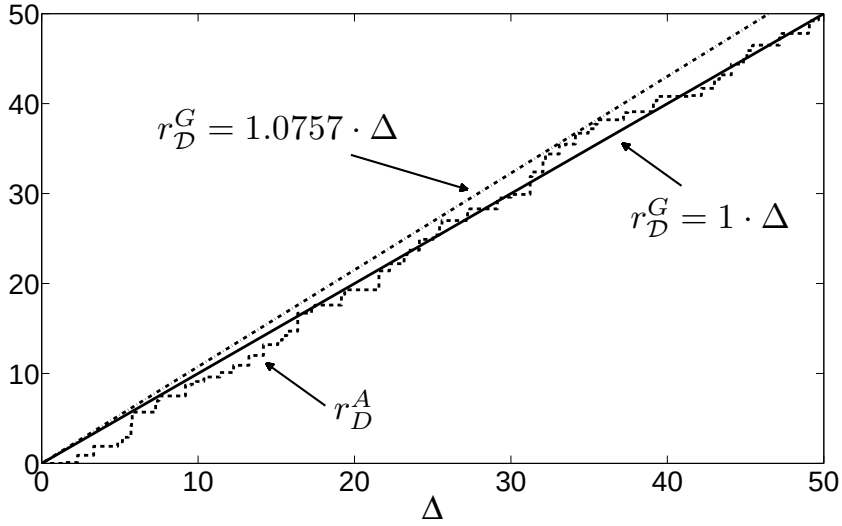

Fig. 2.15: Resource assume $r^A_{\mathcal{D}}$ of $\mathcal{D} \parallel \mathcal{E} \parallel \mathcal{F}$ and the resource guarantee $r^G_{\mathcal{D}}$ for two processor speeds: 1 and 1.0757. Only the second one is compatible with the system (can make all components work together properly)

the assumed resource for the whole system $r^A_{\mathcal{D}}$ is not smaller than the available resource $r^G_{\mathcal{D}} = \beta(\Delta) = 1 \cdot \Delta$ when the processor speed is set to 1. But we can now easily adjust the processor speed to its minimal value such that the system can just work properly, i.e. $r^G_{\mathcal{D}} = \beta(\Delta) = 1.0757 \cdot \Delta$, see also Fig. 2.15.

2.8 Discussion

The chapter generalizes the concept of Real-Time Interfaces that enable compositional analysis of hierarchical and distributed real-time systems. In addition, the concept of adaptive interfaces is formally defined. It allows us to solve design and synthesis problems as the requirements and constraints can be dynamically propagated through the system. In addition, the adaptivity leads to tighter performance results as the properties of the components adapt to the current needs. Moreover, the adaptive interfaces could be implemented in the run-time system which leads to adaptive system behavior that depends on the current requests from the environment.

It should be mentioned that we use a transformational approach to Real-Time Interfaces. On the other hand, this does not prevent us from modeling complex internal behavior of components or non-determinism in streams or components. For example, it has been shown in [WT05a] that stateful and non-deterministic behavior of components can be modeled too. In addition, the abstractions used in Section 2.7.1 describe non-deterministic event streams but one could even use statistical stream models.

The framework presented here can also be used for real-time systems that work in several operating modes. The changing requirements and constraints will automatically propagate through the components when the system changes its operating mode. However, the timing characteristics of the actual switches between the requirements and constraints associated with the different operating modes will be dependent on the mode change protocol that is used [RC04]. Such timing information can be obtained by using mode change analysis techniques such as the ones presented in Chapter 5, and then this information can be used to augment the Adaptive Real-Time Interface. Thus the interface will contain information not only about the bounds of a certain parameter, e.g. maximal input stream rate supported by the system, but it will also have information how quickly the environment can switch between the different parameters that are within these bounds, e.g. what is the minimum delay between the activation of a new input stream and the deactivation of another input stream.

3

Interface Algebra with Rate Interfaces

Interface-based design is now considered to be one of the keys to tackling the increasing complexity of modern embedded systems. The central idea is that different components comprising such systems can be developed independently and a system designer can connect them together only if their interfaces match, without knowing the details of their internals. Chapter 2 developed the theory for an interface-based design framework for real-time systems which is applicable to many existing component-based real-time analysis methods.

This chapter illustrates an interface-based design framework with the Modular Performance Analysis method [CKT03b]. It extends the Real-Time Interfaces theory [WT05b, WT06a], and makes it applicable to systems with multiple distributed processing and communication resources by using the concept of *Rate Interfaces*.

Rate Interfaces are used for compositional (correct-by-construction) design of embedded systems whose components communicate through data streams. Using the associated rate interface algebra, two components can be connected together if the output rate of one component is "compatible" with the input rate of the other component. We formalize this notion of compatibility and show that such an algebra is non-trivial because it has to accurately model the burstiness in the arrival rates of such data streams and the variability in their processing requirements. We discuss how rate interfaces simplify compositional design and at the same time help in functional and performance verification which would be difficult to address otherwise. We illustrate the capabilities of Rate Interfaces through a realistic case study involving a component-based design of a multiprocessor architecture running a picture-in-picture video application.

3.1 Introduction

Most embedded systems today consist of a heterogeneous collection of processing elements, application-specific hardware accelerators and memory modules, which are connected together using some communication subsystem. Typically these components are designed by different vendors, and hence integrating them together and ensuring that the system works correctly often turns out to be a challenging problem. To address this problem, recently there has been a considerable emphasis on interface-based design [dAH01b]. Here, the basic premise is that a system designer using a component only needs to understand the component's interface and not the details of how the functionality offered by the component is implemented. In other words, it should be possible to integrate a set of components if their interfaces "match". This gives rise to two related questions: (i) What kind of information about a component should be exposed by its interface? (ii) How is the notion of two interfaces "matching" technically formulated and realized? Answers to these questions clearly depend on the system being designed. Hence, lately there has been a number of proposals on different kinds of interface specifications and on what constitutes a good notion of "match" for different system design scenarios, see for example [dAH05, WT05b, WT06a, HM06].

Following this line of work, in this chapter we use the concepts of Modular Performance Analysis with Real-Time Calculus as proposed in [TCN00, CKT03b] and combine it with ideas from assume/guarantee (A/G) interfaces [dAH01a] to propose what we refer to as Rate Interfaces. They complement the concept of Real-Time Interfaces introduced in [WT05b, WT06a, WT06b] as Real-Time Interfaces focus on single processor systems with different scheduling policies. Here we focus on distributed multiprocessor systems.

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

The abstractions offered by Rate Interfaces are particularly suited for component-based design of systems that process continuous data streams.

Examples of these might be sensors sensing at a pre-specified rate and sending the data to an actuator for processing (or triggering certain tasks). Other examples might be media processors, network processors, etc., which consist of multiple processing elements (PEs) with each of them running one or more tasks. An input data stream gets processed at the first PE, the partially processed stream then enters the second PE for further processing, and this continues until the stream is fully processed and leaves the system. The timing properties (or the output rate) of a partially processed stream coming out of a PE might be very different from the properties of the input stream. Such a transformation depends on the tasks running on the PE, the scheduling policy used, and the architecture of the PE [RJE03]. In this setup, two PEs can be composed (i.e. work together) if the timing properties of the data stream coming out of the first PE can be guaranteed to be compatible with the timing properties of the input data stream expected by the second PE. Here, compatibility may be defined, for example, as a constraint that the buffer between the two PEs should not overflow.

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

In this chapter, a rate does not only capture the average arrival rate of a data stream, but it also accurately specifies the burstiness in the arrival of a stream over different time scales. Such a detailed specification is necessary for our framework to be useful because on-chip traffic and processing requirements of applications tend to be highly bursty in modern component-based embedded systems, see for example, [RJE03, RvEP02, VM04]. However, this also necessitates an involved algebra for reasoning about such rate specifications.

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

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

compatible interfaces guarantee buffer overflow and underflow constraints.

- Rate Interfaces complement previous results on Real-Time Interfaces with Real-Time Calculus [WT05b, WT06a, WT06b] towards a distributed setting. Therefore, we not only consider tasks that are executed on a single processing element but allow for data streams that are processed on several computing resources and communicated through different communication media. This way, independent composition in terms of data streams and resources is possible.

- We consider variable execution demands of tasks in an interface-based design approach which improves the accuracy of the analysis. Given that many multimedia tasks (especially from the video encoding/decoding domain) exhibit very high variations in their execution requirements, this makes our framework more useable for designing component-based distributed multimedia architectures.

- We consider not only component-wise constraints such as buffer underflow and overflow, but also constraints on networks of components such as end-to-end delays or worst-case traversal times (WCTTs). We develop a novel interface that will allow us to compute WCTTs and verify their compliance to provided deadlines in an incremental manner, i.e. as and when new components are added or removed from the network.

- Lastly, we show that the interface-theoretic formulation of the rate analysis problem has several advantages. These include the possibility of component-level analysis, which is computationally more efficient than the global, monolithic analysis proposed in [LCM06, MKCT04]. This also implies easier component-level design space exploration and resource dimensioning. In other words, questions like "if the scheduler in one particular component is changed, then will all of the buffer constraints in the system still be satisfied?" can now be answered easily. We also show that this formulation enables the modeling of different component-level scheduling policies. We illustrate this for the fixed priority (FP) and the earliest deadline first (EDF) scheduling policies.

The rest of this chapter is organized as follows. Section 3.2 briefly discusses some of the related work. Section 3.3 presents a motivating example that is used throughout the chapter to illustrate the proposed theory. Section 3.4 describes the basics of a component-based

analysis method based on Modular Performance Analysis with Real-Time Calculus. Section 3.5 outlines the basics of assume/guarantee (A/G) interfaces. Section 3.6 shows how they are combined with Real-Time Calculus in order to develop an interface-based design framework for real-time systems that supports constraint propagation. This is followed by our theory of Rate Interfaces in Section 3.7. Section 3.8 presents several extensions of basic Rate Interfaces to take into account variable workloads, support multiple scheduling policies within a component, and model worst-case traversal time constraints that can span several components connected in a network. Section 3.9 presents a case study using the motivating example introduced earlier. This case study illustrates how our proposed interfaces may help in resource dimensioning, incremental design and component refinement. Finally, Section 3.10 summarizes the chapter and outlines some directions for future work.

3.2 Related Work

There have been a few previous attempts to analyze real-time systems in terms of their input/output data rates. More specifically, the *rate analysis* proposed in [MDG98] considered a collection of concurrently executing components that interact through synchronization messages. The problem is then to compute bounds on the execution rates of these components under certain resource constraints. Alternatively, given a set of rate constraints, the problem is to efficiently check if these constraints are consistent. Similarly, [BDN05] addressed the problem of identifying task activation rates for fixed-priority scheduled systems with deadline constraints.

This chapter is largely motivated by our previous efforts to study such rate analysis for buffer-constrained architectures in the context of multimedia processing [LCM06, MKCT04]. In contrast to [BDN05] and [MDG98], which deal with resource and deadline constraints, the problem addressed in [LCM06, MKCT04] was concerned with determining upper and lower bounds on the arrival rates of multimedia streams such that certain buffer overflow and underflow constraints are satisfied. The model specifying the arrival rate (or timing properties) of a stream in [MKCT04] was refined in [LCM06], thereby leading to tighter results.

The interface algebra presented here is an interface-theoretic formulation of the rate analysis problem studied in [LCM06, MKCT04]. In particular, we complement the concept of Real-Time Interfaces by making it applicable to distributed systems with multiple resources. Real-Time Interfaces was proposed in [WT05b, WT06a, WT06b] to enable interface-based design for single processor systems with mixed scheduling policies.

Real-Time Interfaces theory uses concepts from Real-Time Calculus [TCN00, CKT03b] and assume/guarantee interfaces [dAH01b]. Real-Time Calculus is an analytical method for worst-case performance analysis of distributed real-time systems. It is a very general approach based on Network Calculus [Cru91a, Cru91b, LBT01]. It models data streams with arrival curves α, resource availabilities with service curves β, and the processing semantics of the HW/SW components of a system with min/max algebra functions that represent transformations on the arrival and service curves. On the other hand, A/G interfaces are a concept that allows interface-based design of systems. Generally they contain a set of input and output variables, and a predicate ϕ^I on the input variables which states the constraints on the input that a component *assumes*. Another predicate ϕ^O on the output variables represents the *guarantee* that the component will only provide outputs which satisfy ϕ^O if ϕ^I is satisfied.

In Real-Time Interfaces, input variables represent timing properties of resource availabilities, and output variables represent timing properties of unused resources. The predicate ϕ^I represents upper and lower bounds on the necessary resources that make a component schedulable, and the predicate ϕ^O gives guarantees or upper and lower bounds on the resources available for other lower priority components. Such a setting allows composition of components according to a scheduling policy in terms of their resource usage. Real-Time Interfaces have been used to answer schedulability questions for single processor systems with fixed priority (FP), earliest deadline first (EDF), time division multiple access (TDMA), hierarchical scheduling, and scheduling servers [WT06a, WT06b]. Real-Time Interfaces extend A/G interfaces with the concept of constraint propagation which was outlined in Chapter 2, and will be illustrated in Section 3.6.1 too.

This chapter develops the concept of Rate Interfaces to address the rate analysis problem in the context of component-based system design. Rate Interfaces turn out to be a convenient mechanism for component-based design of distributed systems where tasks are partitioned and mapped onto several processing elements communicating via continuous data/event streams. Rate Interfaces combined with Real-Time Interfaces can be used to perform a complete interface-based design of streaming real-time systems where components are composed in terms of resources and data streams.

In Rate Interfaces, the input and output variables represent *rates* (or timing properties) of the input and output data streams for a component. The predicate ϕ^I represents constraints or upper and lower bounds on the rate of the *input* data stream. Similarly, the predicate ϕ^O represents guarantees on the timing properties of the *output/processed* data stream,

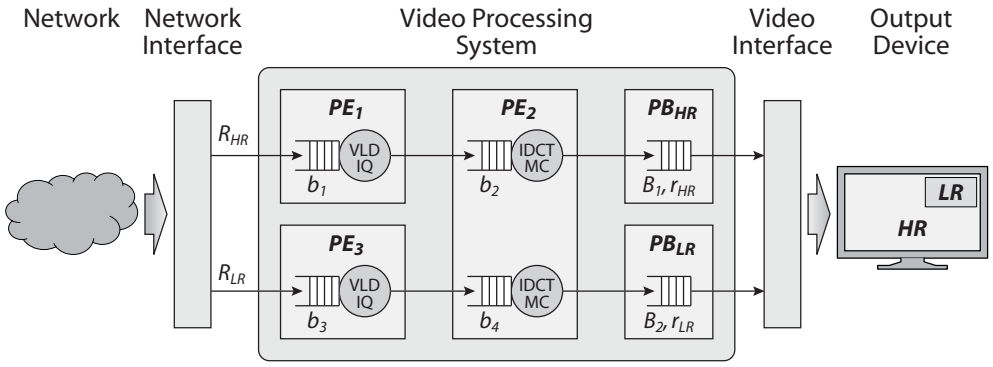

Fig. 3.1: A picture-in-picture (PiP) application decoding two MPEG-2 video streams

which then serve as input to another component. It may be noted here, that the main complexity of Rate Interfaces stems from the manner in which input and output data rates are required to be specified (which we explain later in this chapter).

3.3 Illustrative Example

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

Figure 3.1 shows an architecture consisting of three processing elements PE_1, PE_2, and PE_3 onto which an MPEG-2 decoder has been partitioned and mapped. This architecture implements a picture-in-picture (PiP) application where two concurrent video streams are being decoded and displayed on the same output device. The variable length decoding (VLD) and inverse quantization (IQ) tasks of the decoder have been mapped onto PE_1 and also replicated on PE_3. Each of these two PEs process a different video stream. PE_2, on the other hand, implements the inverse discrete cosine transform (IDCT) and the motion compensation (MC) tasks and processes both of the streams. A scheduler implemented on PE_2 schedules these streams, typically using different QoS parameters for each stream. The stream corresponding to the main window in the output display device is typically associated with a higher frame resolution (indicated as "HR" in Fig. 3.1) and generates a higher workload

on PE_2, compared to the lower resolution video ("LR"in the figure) associated with the secondary window.

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

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

In this chapter we show that following a component-based design approach, each processing element and playout buffer may be considered to be an independent component with well-defined interfaces. To connect these components together in order to realize an architecture like the one shown in Fig. 3.1, a designer only needs to check if their interfaces are compatible. If these interfaces are well-designed, then such compatibility would guarantee that the buffers inside the components would not overflow and the playout buffers would not overflow and underflow. Our goal in this chapter is to design such interfaces. By designing appropriate interfaces, it is also possible to satisfy other non-functional or performance constraints such as the utilization of the components should be above a certain threshold or that the data streams being processed should meet their maximum WCTT constraints.

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

We will follow the general approach proposed in Chapter 2 that yields a component system suitable for interface-based design of real-

time systems. The approach can be summarized in the following three steps:

1. First, we need to define an abstract component that describes the real-time properties of a concrete hardware/software system component. This entails defining proper abstractions for component inputs and outputs, and internal component relations that meaningfully relate abstract inputs to abstract outputs.

2. To derive the interface of an abstract component we need to define interface variables as well as input and output predicates on these interface variables.

3. Finally, we need to establish the internal interface relations that relate incoming guarantees and assumptions to outgoing guarantees and assumptions of the interface.

3.4 Modular Performance Analysis

In this section, we will describe a component-based method suitable for performance analysis of stream-processing distributed real-time systems based on Real-Time Calculus. The abstract component model for the picture-in-picture application of Fig. 3.1 is depicted in Fig. 3.2.

In real-time systems, as we consider them in this chapter, event streams are processed on a sequence of hardware/software (HW/SW) components that we will interpret as tasks executing on possibly different hardware resources. Figure 3.3(a) depicts such a component. An event or data stream described by the cumulative function $R(t)$ enters the input buffer of the component and is eventually processed by the component that is executed on a hardware resource whose availability is described by the cumulative function $C(t)$. The domain of these functions is extended to the real numbers as components may only partially process events or data items.

Definition 3.1: (Cumulative Functions) *The cumulative function* $R(t) \in \mathbb{R}^{\geq 0}$ *(*$C(t) \in \mathbb{R}^{\geq 0}$*) for* $t \geq 0$ *denotes the number of events/data items that have been received (could be processed) within the time interval* $[0, t)$*.*

After being processed, events are emitted on the component's output, resulting in an outgoing event stream $R'(t)$, and the remaining resources that were not consumed are available to be used by other components and are described by an outgoing resource availability trace $C'(t)$. The relations between $R(t)$, $C(t)$, $R'(t)$ and $C'(t)$ depend on the component's

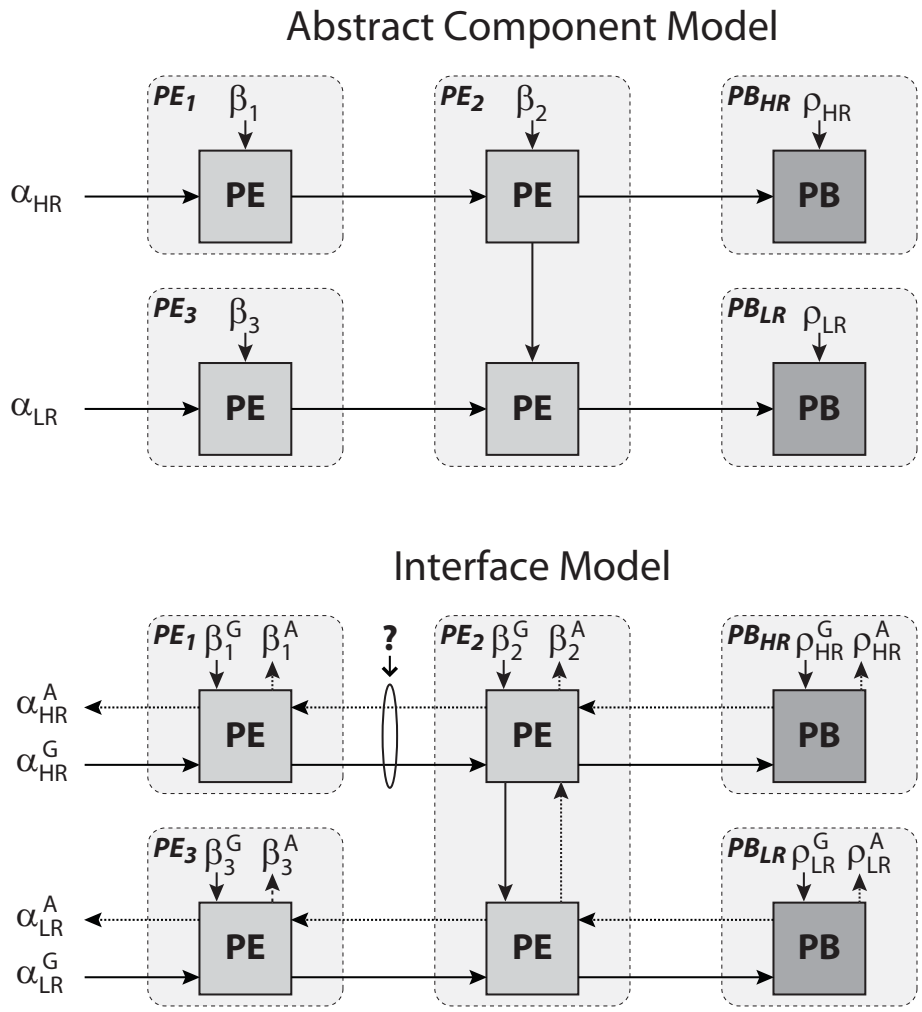

Fig. 3.2: A component model (top) and the corresponding interface model (bottom) of the architecture in Fig. 3.1

processing semantics. For example, Greedy Processing (GP), also called work-conserving, 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.

For the purpose of analysis, we model such a HW/SW component as an abstract component as depicted in Fig. 3.3(b). While cumulative functions such as $R(t)$ or $C(t)$ describe one concrete trace of an event stream or a resource availability, variability characterization curves (VCC) provide means to capture all possible traces within an event stream or of a resource availability, see for example [MZCW04]. In the context of this chapter, we use the notion of arrival curves and service curves [LBT01],

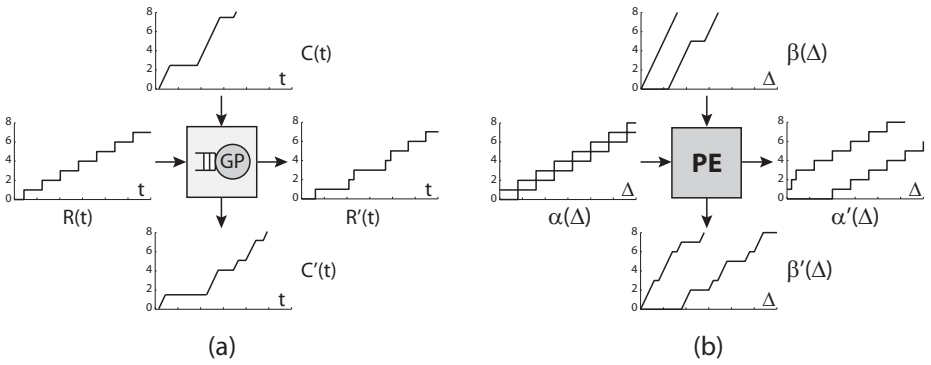

Fig. 3.3: **(a)** A concrete component, processing an event stream on a resource. **(b)** An abstract component, processing an abstract event stream on an abstract resource. The label GP (greedy processing) denotes that events are processed whenever resources are available

which are special cases of VCCs. They are considerably more expressive than, for example, traditional event stream models. While these typically only specify a period and a jitter to model an event stream, arrival curves and service curves can accurately characterize the detailed burstiness and variability of event streams and resource availabilities. We further use workload curves [MKT04] to model the variable execution demands of tasks, and readout curves to model the removing of data items from a playout buffer.

Arrival curves are used to characterize the burstiness in the arrival pattern of events or data items. They have the following formal definition.

Definition 3.2: (Arrival Curves) *Let $R(t)$ denote the total number of events that arrived in the time interval $[0, t)$. Then, the arrival curves $\alpha^l(\Delta)$ and $\alpha^u(\Delta)$ denote the minimum and the maximum number of events that can arrive in* any *time interval of length Δ, i.e. $\alpha^l(t-s) \leq R(t) - R(s) \leq \alpha^u(t-s)$ for all $t > s \geq 0$. In addition, $\alpha^l(0) = \alpha^u(0) = 0$. We also denote $\alpha = (\alpha^l, \alpha^u)$.*

Service curves are used to characterize the variability in the service provided by a resource. Service curves have the following formal definition.

Definition 3.3: (Service Curves) *Let $C(t)$ denote the total number of events that could be processed in the time interval $[0, t)$. Then, the service curves $\beta^l(\Delta)$ and $\beta^u(\Delta)$ denote the minimum and the maximum number of events that could be processed within* any *time interval of length Δ, i.e. $\beta^l(t-s) \leq C(t) - C(s) \leq \beta^u(t-s)$ for all $t > s \geq 0$. In addition, $\beta^l(0) = \beta^u(0) = 0$. We also denote $\beta = (\beta^l, \beta^u)$.*

Workload curves are similar to arrival and service curves however, they describe the variability in the execution requirements (for example in terms of processing cycles or time) necessary to process an event or communicate a data item. Therefore, they relate physical to abstract quantities in the analysis. They have the following formal definition.

Definition 3.4: (Workload Curves) *The workload curves $\gamma^l(v)$ and $\gamma^u(v)$ denote the minimum and the maximum execution requirement that is necessary to process* any *v consecutive stream objects or events within a given sequence. We also denote $\gamma = (\gamma^l, \gamma^u)$.*

In Fig. 3.3(b), an abstract event stream $\alpha(\Delta)$ enters the abstract component and is processed using an abstract resource $\beta^l(\Delta)$. The output is again an abstract event stream $\alpha'(\Delta)$, and the remaining resources are expressed again as an abstract resource $\beta'^l(\Delta)$. Later, we will also use the concept of VCCs to specify the process of reading out a playout buffer (readout VCC). Note that the domain of the arrival and service curves used in Fig. 3.3(b) are events, i.e. they describe the number of arriving events and the capability to process a certain number of events. The generalization towards physical quantities such as processing cycles or computation time by means of workload curves as defined above will be done in Section 3.8.1.

3.4.1 Abstract Components

The internal relations of an abstract component depend on the processing semantics and scheduling policy of the modeled HW/SW component. For modeling distributed stream-processing systems, we consider three basic types of components:

- **Processing Element (PE):** It can be used to model a single *processing element* which processes one input stream. However, it can also be used to represent a composition with other components of the same type and model processing elements that process more than one input stream with a fixed priority (FP) scheduling as illustrated in component PE_2 in Fig. 3.2 and described in Section 3.7.2.

- **Playout Buffer (PB):** It models a component that receives data which is stored in a buffer, and the buffer is read at a constant (usually periodic) rate. An example is the PB_{HR} component shown in Fig. 3.2.

- **EDF Component:** This component is similar to PE but it processes several data streams by using the earliest deadline first (EDF) scheduling policy. Such a component can be used, for example, to replace component PE_2 in Fig. 3.2.

3.4.1.1 Processing Element

Let us first derive the abstract component model **PE**. Consider the *greedy processing component* from Fig. 3.3(a), 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 (or processor capacity). 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. Using Definition 3.1 and the above semantics, we define the input-output relations of a greedy processing component with the following theorem.

Theorem 3.5: (PE Relations in Time Domain) *A greedy processing component with an empty input buffer at $t = 0$ satisfies the following input-output relations with $R(t)$ defined as the number of events that have been received, $C(t)$ as the number of events that could be processed, $R'(t) \in \mathbb{R}^{\geq 0}$ as the number of events that have been processed, $C'(t) \in \mathbb{R}^{\geq 0}$ as the number of events of other event streams that could be processed, and $b(t) \in \mathbb{R}^{\geq 0}$ as the number of events stored in the input buffer at time t:*

$$b(t) = R(t) - R'(t) \ , \tag{3.1}$$

$$C(t) = C'(t) + R'(t) \ , \tag{3.2}$$

$$R'(t) = \inf_{0 \leq u \leq t} \{R(u) + C(t) - C(u)\} \ . \tag{3.3}$$

Proof. The first two relations simply describe the conservation laws of a greedy processing element. As no events are lost, all events that arrived until t either left the processing element or they are still stored in the buffer, see (3.1). In addition, all the available computing capability of the processing element until t ($C(t)$), has either been used to process events ($R'(t)$), or has been available to process other events from lower priority streams ($C'(t)$), see (3.2). In order to prove (3.3), first note that for all $u \geq 0$ we have $R'(u) \leq R(u)$ as $b(u) \geq 0$. We also find $R'(t) - R'(u) \leq C(t) - C(u)$ for $u \leq t$ as the number of output events in the time interval $[u, t)$ can not be larger than the available processing ability in $[u, t)$. As a result, we obtain $R'(t) \leq R(u) + C(t) - C(u)$. In order to show (3.3), we need to prove that there exists a $u \leq t$ such that $R'(t) = R(u) + C(t) - C(u)$. To this end, let us suppose that u^* is the last time instance before t with an empty input buffer. Such a time exists as the buffer was empty at $t = 0$. We have $R(u^*) = R'(u^*)$ at time u^* and also $R'(t) - R'(u^*) = C(t) - C(u^*)$ as all available resources in $[u^*, t)$ have been used to produce output because of the greediness of the processing element. Therefore, we find $R'(t) = R(u^*) + C(t) - C(u^*)$.

■

Such a component can be modeled by an abstract component PE as depicted in Fig. 3.3(b) and in Fig. 3.2 with the following internal component relations in the time interval domain[1]:

$$\alpha'^u = \alpha^u \oslash \beta^l \ , \tag{3.4}$$

$$\alpha'^l = \alpha^l \otimes \beta^l \ , \tag{3.5}$$

$$\beta'^l(\Delta) = \sup_{0 \leq \lambda \leq \Delta} \left\{ \beta^l(\lambda) - \alpha^u(\lambda) \right\} \ , \tag{3.6}$$

where the backlog of the input FIFO buffer can be bounded by:

$$b(t) \leq \sup_{0 \leq \lambda \leq \Delta} \left\{ \alpha^u(\lambda) - \beta^l(\lambda) \right\} \ . \tag{3.7}$$

These relations will not be proven here as they follow directly from Network Calculus [LBT01] and Real-Time Calculus [CKT03b].

If the maximum available buffer space in the input buffer is constrained by $b_{\max}$, the backlog should never become bigger than the buffer size, $b(t) \leq b_{\max}$, i.e. we have a *buffer overflow constraint*. In this case, we can obtain the following component-based constraint on the admissible arrival and service curves:

$$\alpha^u(\Delta) \leq \beta^l(\Delta) + b_{\max} \qquad \forall \Delta \in \mathbb{R}^{\geq 0} \ . \tag{3.8}$$

If the input arrival and service curves satisfy the above constraint, the backlog will never be bigger than $b_{\max}$.

This abstract PE component is one of the basic building blocks in Real-Time Calculus as it can be used to model a single processing element which processes a single input stream or a composition with other components of the same type and model components processing more than one input stream with a fixed priority (FP) scheduling.

3.4.1.2 Playout Buffer

Now, let us derive the abstract component model **PB** of the *playout buffer* as used in the picture-in-picture application in Fig. 3.1, and as shown in Fig. 3.2. It receives data at a rate bounded by the arrival curve $\alpha(\Delta)$. The incoming data is stored in a buffer of maximum size $B_{\max}$. We make the assumption that at time $t = 0$, there are already $B(0) = B_0$ data items in the playout buffer, e.g. due to a playback delay.

Data items in the playout buffer are removed by the video interface, see Fig. 3.1. In particular, $D(t)$ data items are removed within the time interval $[0, t)$. This behavior can be described by the readout VCC $\rho(\Delta) =$

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

$(\rho^l(\Delta), \rho^u(\Delta))$, i.e. $\rho^l(t-s) \leq D(t) - D(s) \leq \rho^u(t-s)$ for all $t > s \geq 0$. What needs to be guaranteed is that the playout buffer PB never *overflows or underflows*.

Theorem 3.6: (PB Relations in Time Interval Domain) *Let be given a playout buffer (PB) component with input and readout event streams $R(t)$ and $D(t)$ which are characterized by the VCCs α and ρ, respectively. Further suppose that there are $B(0) = B_0$ events in the playout buffer at time $t = 0$. Then the playout buffer size is constrained by $0 \leq B(t) \leq B_{\max}$ if the following constraints are satisfied:*

$$\alpha^l(\Delta) \geq \rho^u(\Delta) - B_0 \qquad \forall \Delta \in \mathbb{R}^{\geq 0}\,, \tag{3.9}$$

$$\alpha^u(\Delta) \leq \rho^l(\Delta) + B_{\max} - B_0 \qquad \forall \Delta \in \mathbb{R}^{\geq 0}\,. \tag{3.10}$$

Proof. Assume that the total number of data items written into the playout buffer PB till time t is given by $R(t)$ and the total number of data items read and removed from PB till time t is given by $D(t)$ (readout stream). Then the number of data items $B(t)$ in the buffer at time t can be expressed as $B(t) = R(t) - R(0) - [D(t) - D(0)] + B_0$. Using the definitions of the associated VCCs we simply get the inequalities $B(t) \geq \alpha^l(\Delta) - \rho^u(\Delta) + B_0$ for all $\Delta \geq 0$ and $B(t) \leq \alpha^u(\Delta) - \rho^l(\Delta) + B_0$ for all $\Delta \geq 0$. In other words, in order for the PB not to underflow and overflow, its input arrival curve α and readout curve ρ need to obey (3.9) and (3.10).

■

3.4.1.3 Earliest Deadline First Component

In order to model a component processing several input streams with an *earliest deadline first* (**EDF**) scheduling, a new abstract component is needed with different internal relations [WT06a], as the greedy processing component PE defined above can not handle a deadline-based scheduling policy. Such an EDF 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 buffer with maximum size $b_{i\,\max}$ where events are backlogged. Tasks associated with the input event streams process the head events in these buffers and are scheduled in an EDF order. Each task has a best-case execution time BCET_i, a worst-case execution time WCET_i, and a relative deadline D_i where $\mathrm{BCET}_i \leq \mathrm{WCET}_i \leq D_i$. 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. The following theorem describes the properties of the EDF component. It

uses the following shift function which simply shifts a given curve $\alpha(\Delta)$ by the amount c to 'the right':

$$r(\alpha, c, \Delta) = \begin{cases} \alpha(\Delta - c) & \text{if } (\Delta > c) \wedge (\Delta \neq 0) \\ 0 & \text{if } (\Delta \leq c) \vee (\Delta = 0) \end{cases} \tag{3.11}$$

Theorem 3.7: (EDF Component Relations in Time Interval Domain) *Let be given an EDF component as described above with an input service curve β. The event streams i are characterized by arrival curves α_i and have associated tasks with relative deadlines, worst-case and best-case execution times given as $D_i \geq WCET_i \geq BCET_i$, respectively. All tasks are processed within their deadlines if and only if:*

$$\sum_{i=1}^{N} r(\alpha_i^u, D_i, \Delta) \leq \beta^l(\Delta) \qquad \forall \Delta \in \mathbb{R}^{\geq 0} , \tag{3.12}$$

using the shift function r from (3.11). The output streams can be characterized by arrival curves computed as follows:

$$\alpha_i'^u(\Delta) = r(\alpha_i^u, -(D_i - BCET_i), \Delta) \qquad \forall i , \tag{3.13}$$

$$\alpha_i'^l(\Delta) = r(\alpha_i^l, (D_i - BCET_i), \Delta) \qquad \forall i . \tag{3.14}$$

The number of events in the input buffers do not exceed their capacity $b_{i\,\max}$ if the following constraints are satisfied:

$$\alpha_i^u(D_i) \leq b_{i\,\max} \qquad \forall i . \tag{3.15}$$

Proof. Let us first prove (3.12). Function $r(\alpha_i, D_i, \Delta)$, as defined in (3.11), characterizes an event stream α_i where all events need to be processed within the same relative deadline D_i. In particular, it returns the maximum number of events that arrive in any time interval of length Δ *and* have their deadlines in the same time interval. The function is defined similarly to a *demand bound function* as introduced by Baruah et al. in [BCGM99]. We now would like to prove that if a component is processing N independent streams given a minimum resource availability β^l using an EDF resource sharing policy, all events will be processed within their deadlines, if and only if (3.12) holds.
If: Suppose that $r(\alpha_i, D_i, \Delta)$ is tight, i.e. there is a time interval of length $\Delta = t_i^u - t_i^l$, with $t_i^u > t_i^l \geq 0$, for which the number of events that have their arrival and absolute deadline times within $[t_i^l, t_i^u)$ is equal to $r(\alpha_i, D_i, \Delta)$. Since the streams are independent, we can construct a critical instant by shifting the time intervals of all of the N input streams such that $t^l = t_i^l = t_j^l$, $t^u = t_i^u = t_j^u$, $\forall i, j \in \{1, \ldots, N\}$. Then, the cumulative number of events that have their arrival and deadline times within the interval $[t^l, t^u)$

is given by $\sum_{i=1}^{N} r(\alpha_i, D_i, \Delta)$. Suppose that, we also shift the time axis of the component service in such a way that the number of events that can be processed in the interval $[t^l, t^u)$ is equal to $\beta^l(\Delta)$. If $\sum_{i=1}^{N} r(\alpha_i, D_i, \Delta) > \beta^l(\Delta)$, then clearly not all events can be processed within their deadlines.
Only if: Suppose that there is a deadline violation at some time instance t^u. Let us define t^l to be the earliest time such that for the interval $[t^l, t^u)$, the processor has been operating continuously on events with absolute deadlines smaller than t^u. Because of the greediness of EDF, all resources available in the interval $[t^l, t^u)$ have been used which is at least equal to $\beta^l(t^u - t^l)$, i.e. the processor has not been idling. Because of the definition of EDF, the processor has only been operating on events that have their arrival and deadline times within the interval $[t^l, t^u)$. Therefore, it must be the case that $\sum_{i=1}^{N} r(\alpha_i, D_i, \Delta) > \beta^l(\Delta)$, $\Delta = t^u - t^l$, i.e. the maximum number of events that *must* be processed is greater than the number of events that can be processed in the interval $[t^l, t^u)$.

Now, let us prove (3.13) and (3.14). Because of the deadline-based scheduling, an event arriving at an input buffer i takes at least BCET_i and at most D_i time units until it appears at the output stream. Therefore, the additional jitter it observers is bounded by $(D_i - \text{BCET}_i)$. As a result, the events that arrived in some interval $\Delta + (D_i - \text{BCET}_i)$ can now appear at the output in an interval of length Δ.

Finally, we show (3.15). As any event will leave the associated queue after at most its relative deadline D_i time units, the backlog of each buffer associated with an input event stream can be bounded by $b_i(t) \leq \alpha_i^u(D_i)$ which directly leads to (3.15).

■

3.5 Assume/Guarantee Interfaces in Our Setup

In this section we outline the basic principles of assume/guarantee (A/G) interfaces as proposed in [dAH01b], and briefly discuss how we borrow ideas from them and adapt them in the context of our setup, where components communicate via continuous data streams, have limited buffers inside them, and the interfaces implement a constraint propagation scheme. This section outlines briefly many concepts that were developed in more detail in Chapter 2.

Let us first consider a single component. Its A/G interface consists of two disjoint sets of input and output variables X^I and X^O, as well as two associated predicates ϕ^I and ϕ^O, respectively. The interface makes certain *assumptions* on X^I, which are specified using the predicate $\phi^I(X^I)$. Provided this predicate is satisfied, the interface *guarantees* that the

component works correctly and its output variables will satisfy a predicate $\phi^O(X^O)$. Hence, ϕ^O is the guarantee that the component provides to the environment assuming the precondition ϕ^I. In other words, for each component $\phi^I \Rightarrow \phi^O$ is true for each valuation of input variables X^I. Clearly, the predicate ϕ^I need not be true for *all* possible valuations of X^I, i.e. there might exist environments where this component cannot be used or cannot provide the output guarantee. Environments which satisfy ϕ^I are environments where this component can be used. If there exists at least one environment, i.e. one valuation of variables X^I, such that $\phi^I(X^I)$ is true, then we call the corresponding component satisfiable.

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

Two interfaces can be syntactically composed if their output variables are disjoint. If F and G form a closed system, i.e. all outputs of F are connected to the inputs of G and vice-versa, then for F and G to be *composable*, the following closed formula should be true for all possible valuations of variables:

$$\phi^O_F \wedge \phi^O_G \;\Rightarrow\; \phi^I_F \wedge \phi^I_G \,. \tag{3.16}$$

In other words, what one component guarantees at its output satisfies the predicates of the connected inputs.

If $F||G$ is open, i.e. it has free input variables then (3.16) should be *satisfiable*. In other words, there must exist *some* environment in which F and G can be composed. Expression (3.16) is hence the weakest precondition on the environment of $F||G$ for the two interfaces to be composable. Therefore, the assumption $\phi^I_{F||G}$ on the input variables of the composed interface is given by (3.16).

A component H can now provide inputs to $F||G$ if $\phi^O_H \Rightarrow \phi^I_{F||G}$ is satisfiable. In other words, there should exist a valuation of X^I_H and a valuation of $X^I_{F||G}$ under which $\phi^I_H(X^I_H) \Rightarrow \phi^O_H(X^O_H)$ is true and $\phi^O_H(X^O_H) \Rightarrow \phi^I_{F||G}(X^I_{F||G})$ is also true. This way, a system can be incrementally designed by adding one component at a time and verifying if the newly added component is compatible with the existing partially designed system.

In our setup, each component receives as input one or more data streams, processes them, and the processed data streams enter other components for further processing, see Fig. 3.1). Each component has a certain processing capability and allocates this capability among

the incoming streams according to a predefined scheduling policy. Components also have a specified amount of buffer to hold the incoming streams. As mentioned in Section 3.1, the interface variables X^I and X^O represent rates or timing properties of the incoming and outgoing streams. Hence, the component composability conditions translate into predicates involving these data rates. The predicate ϕ^I specifies constraints on the rates of the incoming streams and ϕ^O specifies guarantees that a component provides on the rates at which the processed streams may be consumed by other components. We will show how to derive the interface variables and predicates in such a way that the composability of two components will implicitly guarantee that the buffers inside them do not overflow, and for certain components they also do not underflow.

By extending the interface theory provided in [dAH01b], we also propagate information about the predicates between the interfaces, see also Chapter 2. This way, we combine interface theory with constraint propagation, which enables parameterized design of component-based systems. To this end, as we are dealing with stateless interfaces, X^I and X^O can be related by a transfer function, e.g. $X^O = F(X^I)$ where the actual function depends on the semantics of the modeled component.

In the context of embedded multimedia systems — which is the application domain we focus on — the proposed framework can be used to determine the maximal input rates of multimedia streams, processor speeds, and scheduling policy parameters of the different processing elements and various buffer sizes, such that all components work as intended and the system-wide buffer and timing constraints are satisfied.

3.6 Basis of an Interface-based Design Theory for Real-Time Systems

In Section 3.4, we introduced abstract components which work with abstract representations of the input event streams and the resource availabilities. In this section we need to relate them to the interface-theoretic concepts as introduced in Section 3.5, and develop the basis for an interface theory that will be used in Rate Interfaces similarly to Real-Time Interfaces [WT05b, WT06a, WT06b]. In order to do this, first we need to relate input and output interface variables to inputs and outputs of components, i.e. arrival, service, and readout curves. Secondly, we need to define the input and output predicates in terms of the VCC curves. Lastly, we will extend the interface theory with a method for constraint propagation.

In our setup, the input and output variables of the interface of an abstract component are VCC curves. They can be related to event streams (arrival curves α), resource availabilities (service curves β), or readout streams (ρ). In our presentation we will use the terms input and output ports. Each component has a set of input ports I and a set of output ports O. A connection from output j of one component to the input i of some other component will be denoted by (j, i).

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

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

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

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

Following the notation introduced in Section 3.5, we can now define the input and output predicates for some component input i and output j as:

$$\phi_i^I(\alpha_i) = (\alpha_i \vdash \alpha_i^A) \ , \qquad \phi_j^O(\alpha_j) = (\alpha_j \vdash \alpha_j^G) \ , \tag{3.17}$$

where α^A and α^G are assume and guarantee curves provided by the component interface. Now similarly as for A/G interfaces, we can define that the interfaces of every component must satisfy the following condition:

$$\bigwedge_{\forall i \in I} \phi_i^I(\alpha_i) \Rightarrow \bigwedge_{\forall j \in O} \phi_j^O(\alpha_j) \tag{3.18}$$

for all input curves α_i. In other words, if the input predicates of a component are all satisfied, then it works correctly and all output predicates are satisfied too. Note that the output curves α_j of a component are a function of its input curves α_i, see for example (3.4, 3.5, 3.6) and (3.13, 3.14).

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. Following Section 3.5, this can be done by testing whether the following relation is satisfiable:

$$\bigwedge_{\forall (j,i)} \phi_j^O(\alpha_j) \Rightarrow \bigwedge_{\forall (j,i)} \phi_i^I(\alpha_i) \ . \tag{3.19}$$

3.6.1 Constraint Propagation

The input and output predicates (3.17) are characterized by the assume and guarantee curves α^A and α^G, respectively. If they are static,

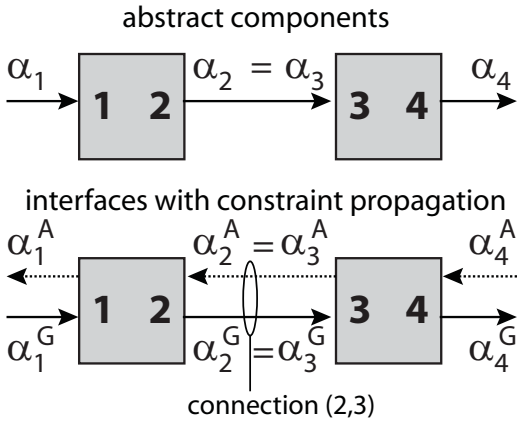

Fig. 3.4: Relation between input/output variables of abstract components and assume/guarantee quantities in real-time interfaces

i.e. independent of the system composition and the properties of the connected components, then we are severely limited in the applicability of the interface concept. For example, let us suppose that a component F needs to guarantee that some buffer never overflows and this fact can be guaranteed by limiting the input rate, i.e. by determining an appropriate input predicate ϕ_F^I. If a preceding component G, i.e. a component that transmits this input stream, is not aware of this predicate, then (3.19) will not be satisfied in general. We need a method to propagate ϕ_F^I to component G and combine it with the input predicate ϕ_G^I, i.e. the input to G should satisfy constraints from both components, F and G, respectively. This way, if another component is to be connected to the input of G, we only need to check at the interface connection if the composition will be successful, i.e. constraints from F and G will be satisfied. In other words, the constraints of individual components should propagate through the interfaces.

To this end, we propagate the assume and guarantee curves of the input and output predicates through the interfaces as shown in Fig. 3.4. Considering connections (j, i), we connect the interfaces, i.e. the corresponding guarantee and assume curves, as follows:

$$\forall\,(j,i):\quad (\alpha^G(i) = \alpha^G(j)) \wedge (\alpha^A(j) = \alpha^A(i))\,. \tag{3.20}$$

If we now determine the transfer function of a component interface in a specific manner, i.e. how output assume and guarantee curves are determined based on input curves, we can (a) propagate constraints and (b) check the compatibility of components in an integrated method.

Theorem 3.8: (Relations between Input and Output Assumes and Guarantees) *Let us suppose that the assume and guarantee quantities of an interface*

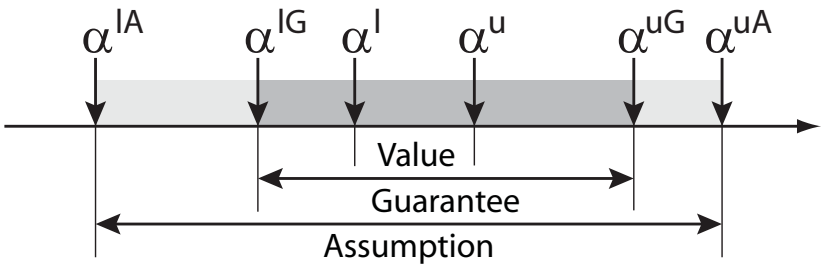

Fig. 3.5: Relation between interface assumptions, interface guarantees, and values

of any component and their relation to the input and output curves of the corresponding abstract component satisfy:

$$(\forall i \in I:\ \alpha_i \vdash \alpha_i^G \vdash \alpha_i^A) \Rightarrow (\forall j \in O:\ \alpha_j \vdash \alpha_j^G \vdash \alpha_j^A)\,, \tag{3.21}$$

where the component has input and output ports, I and O, respectively.

If we have a network of components and the relations:

$$\forall \textit{ inputs } i:\quad \alpha_i^G \vdash \alpha_i^A \tag{3.22}$$

are satisfied, then the system works correctly, i.e. (3.18) and (3.19) hold.

Proof. At first note that the 'complies to' operator $\vdash$ is transitive and therefore, $\alpha \vdash \alpha^G \vdash \alpha^A \Rightarrow \alpha \vdash \alpha^A$. From (3.21), we find $(\forall i \in I : \alpha_i \vdash \alpha_i^A) \Rightarrow (\forall j \in O : \alpha_j \vdash \alpha_j^G)$ which directly leads to (3.18). If we have a connection (j, i), then from (3.20) we have $\alpha_i = \alpha_j$, $\alpha_i^A = \alpha_j^A$ and $\alpha_i^G = \alpha_j^G$. From (3.22) we find that as well $\alpha_j^G \vdash \alpha_j^A$ as $\alpha_i^G \vdash \alpha_i^A$ hold for any connection (j, i). As $(\alpha_j \vdash \alpha_j^G) = (\alpha_i \vdash \alpha_i^G)$, we can conclude that for any connection (j, i), we have $(\alpha_j \vdash \alpha_j^G \vdash \alpha_j^A) = (\alpha_i \vdash \alpha_i^G \vdash \alpha_i^A)$ and therefore $(\alpha_j \vdash \alpha_j^G) = (\alpha_i \vdash \alpha_i^A)$. Expression (3.19) follows directly using the definitions (3.17).

■

The relations between the values, the assume and the guarantee curves of a single connection are shown in Fig. 3.5. As a result of the above Theorem 3.8, we can determine whether two abstract components are compatible by checking the compatibility of their interfaces. We can then generalize that two interfaces are compatible if (3.22) is true for all internal rate, readout and service connections, respectively. Note, that in our setup this also implies that all open input variables are satisfiable.

In the next section we develop a method to determine the internal interface relations for the abstract real-time streaming components introduced in Section 3.4.

3.7 Rate Interfaces

Now we need to develop the relations between guarantees and assumptions in order to satisfy (3.21) for every component. We will first describe a general method how these relations can be determined and then apply it to the real-time components described in Section 3.4.

To this end, we first 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 3.9: (Monotone Abstract Component) *An abstract component with a set of input and output ports I and O, respectively, and a transfer function F that maps input rates to output rates, is monotone if:*

$$((\widehat{\alpha_i} : i \in I) \vdash (\alpha_i : i \in I)) \Rightarrow ((\widehat{\alpha_j} : j \in O) \vdash (\alpha_j : j \in O)) \ ,$$

where $(\alpha_j : j \in O) = F(\alpha_i : i \in I)$ *and* $(\widehat{\alpha_j} : j \in O) = F(\widehat{\alpha_i} : i \in I)$.

In other words, if we replace the input rates of an abstract component with rates that are compliant, then the new output rates are also compliant to the previous ones. Note that all components we look at in this chapter satisfy this monotonicity condition, see for example (3.4, 3.5, 3.6) and (3.13, 3.14).

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 $j \in O$ where I and O denote the input and output ports of the corresponding abstract component, respectively.

Theorem 3.10: (Computation of Output Guarantees and Input Assumes) *Given a monotone component with input ports I, output ports O, and a transfer function F that maps input curves to output curves, i.e.* $(\alpha_j : j \in O) = F(\alpha_i : i \in I)$.

Let us suppose that we determine the output guarantees using:

$$\alpha_j^G = F_j(\alpha_i^G : i \in I) \quad \forall j \in O \ . \tag{3.23}$$

Furthermore, suppose that the input assumes are computed such that:

$$\forall j \in O \ \ \exists i^* \in I : \left(F_j(\alpha_i^G : i \in I)\Big|_{\alpha_{i^*}^G \leftarrow \alpha_{i^*}^A} \ \vdash \ \alpha_j^A \right) , \tag{3.24}$$

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

Then (3.21) holds.

Proof. Let us assume that for all input ports $i \in I$ we have $\alpha_i \vdash \alpha_i^G$, see (3.21). Using the monotonicity of F, we can now see that $(\forall i \in I : \alpha_i \vdash \alpha_i^G) \Rightarrow F(\alpha_i : i \in I) \vdash F(\alpha_i^G : i \in I) \Rightarrow (\forall j \in O : \alpha_j \vdash \alpha_j^G)$.

We still need to show that $(\forall i \in I : \alpha_i^G \vdash \alpha_i^A) \Rightarrow (\forall j \in O : \alpha_j^G \vdash \alpha_j^A)$ using the construction in (3.23). At first note that this expression is equivalent to

$$\forall j \in O \;\; \exists i^* \in I : \left((\alpha_{i^*}^G \vdash \alpha_{i^*}^A) \Rightarrow (\alpha_j^G \vdash \alpha_j^A)\right) .$$

We also know that for any $i^* \in I$ we have:

$$(\alpha_{i^*}^G \vdash \alpha_{i^*}^A) \Rightarrow \left((\alpha_i^G : i \in I) \;\; \vdash \;\; (\alpha_i^G : i \in I)\Big|_{\alpha_{i^*}^G \leftarrow \alpha_{i^*}^A}\right) .$$

Because of the monotonicity of F we can derive that for any $i^* \in I$ we have:

$$(\alpha_{i^*}^G \vdash \alpha_{i^*}^A) \Rightarrow \left(F(\alpha_i^G : i \in I) \;\; \vdash \;\; F(\alpha_i^G : i \in I)\Big|_{\alpha_{i^*}^G \leftarrow \alpha_{i^*}^A}\right) ,$$

and using (3.23) we find:

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

■

The theorem establishes that we can simply determine the output guarantees using the components of a given transfer function of the abstract component, see (3.23). For the input assumes we need to determine some kind of inverses of the transfer functions F_j with respect to at least one of its arguments. Following (3.24), all arguments of some F_j are determined by the input guarantees but one, say for example $\alpha_{i^*}^G$. This one we replace by $\alpha_{i^*}^A$ and try to determine this curve such that the result of the transfer function still complies to the given output assumes. If we choose the same i^* for several components of the output function, then the resulting $\alpha_{i^*}^A$ needs to comply to all partial 'inverses'. As we will see later on, we may determine the inverses of the transfer functions F_j with respect to more than one of its arguments. This way, a violation at some output port will appear as violations at more than one input port. Later on, we will attempt 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.

3.7.1 Abstract Components: Interface Relations

We will now illustrate the concept of rate interfaces using a system with a processing element (PE) containing one greedy processing abstract component and one playout buffer (PB) as shown in Fig. 3.6. Due to the modularity of our framework, the results derived here can be used for building more complex systems.

In this system, there is one input stream, which is described by the arrival curve $\alpha_x = (\alpha_x^l, \alpha_x^u)$. The processing element PE provides a service described by the lower service curve β^l. The processed output stream is described by the arrival curve $\alpha_y = (\alpha_y^l, \alpha_y^u)$. Note that the input stream α_x could be a stream coming from the environment or it could have been processed by other processing elements. The processed stream α_y is the input for the playout buffer. Data items from the PB are read at a rate described by the readout VCC $\rho = (\rho^l, \rho^u)$. It specifies the minimum and maximum number of data items being read from the PB over time intervals of any specified length. For the system shown in Fig. 3.6, we will include into the interface descriptions of the components the conditions under which the buffer b associated with the component in the PE never overflows, and the buffer B in the PB does not overflow or underflow. Fig. 3.7 represents the abstract components of Fig. 3.6 in terms of their interfaces.

3.7.1.1 Processing Element

Now, using the relation between interface values, assumptions and guarantees from Theorem 3.8, and following the results from Theorem 3.10, we can deduce that the equations describing the output guarantees are equivalent to those for the abstract component, i.e. (3.4) and (3.5), just using interface guarantees instead of values. Therefore, we have:

$$\alpha_y^{lG} = \alpha_x^{lG} \otimes \beta^{lG} , \tag{3.25}$$

$$\alpha_y^{uG} = \alpha_x^{uG} \oslash \beta^{lG} . \tag{3.26}$$

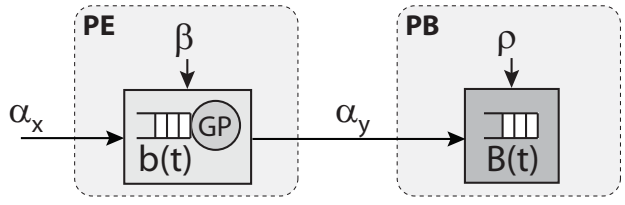

Fig. 3.6: Abstract components of a system with one processing element and one playout buffer

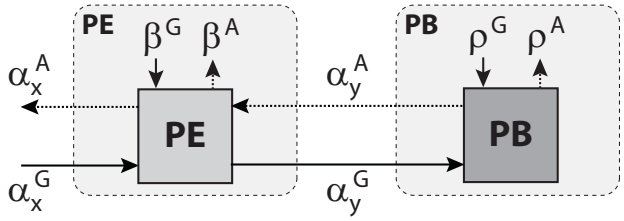

Fig. 3.7: Interface description of the abstract components shown in Fig. 3.6

In order to calculate the input assumptions of the abstract component in the processing element, we need to determine inverse relations corresponding to (3.4), (3.5) and the constraint (3.8). In particular, we need to compute the largest upper curve and smallest lower curve for which the respective relations still hold. This leads to the weakest possible input assumption. Following results from Network Calculus [LBT01], 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 equation (3.5), we get the inequalities $\alpha_x^{lA} \geq \alpha_y^{lA} \oslash \beta^{lG}$ and $\beta^{lA} \geq \alpha_y^{lA} \oslash \alpha_x^{lG}$. Here we use the duality relation between the $\oslash$ and $\otimes$ operators as defined in Appendix A. In a similar way, from the pseudo-inverses of equation (3.4), we get the inequalities $\beta^{lA} \geq \alpha_x^{uG} \oslash \alpha_y^{uA}$ and $\alpha_x^{uA} \leq \beta^{lG} \otimes \alpha_y^{uA}$. Inverting the buffer overflow constraint (3.8) is trivial and we get the inequalities $\alpha_x^{uA} \leq \beta^{lG} + b_{\max}$ and $\beta^{lA} \geq \alpha_x^{uG} - b_{\max}$.

In summary, after combining the previous inequalities, the assumptions related to the abstract component PE can be determined as follows:

$$\alpha_x^{lA} = \alpha_y^{lA} \oslash \beta^{lG} \ , \tag{3.27}$$

$$\alpha_x^{uA} = \min\{\beta^{lG} \otimes \alpha_y^{uA}, \beta^{lG} + b_{\max}\} \ , \tag{3.28}$$

$$\beta^{lA} = \max\{\alpha_y^{lA} \oslash \alpha_x^{lG}, \alpha_x^{uG} \oslash \alpha_y^{uA}, \alpha_x^{uG} - b_{\max}\} \ . \tag{3.29}$$

3.7.1.2 Playout Buffer

For a playout buffer PB, the relations are simpler. We only need to determine the inverse relations for the buffer constraints (3.9) and (3.10),

which directly yield the following relations:

$$\alpha_y^{uA} = \rho^{lG} + B_{\max} - B_0 \ , \tag{3.30}$$
$$\alpha_y^{lA} = \rho^{uG} - B_0 \ , \tag{3.31}$$
$$\rho^{uA} = \alpha_y^{lG} + B_0 \ , \tag{3.32}$$
$$\rho^{lA} = \alpha_y^{uG} - (B_{\max} - B_0) \ . \tag{3.33}$$

We can now also combine the above interface relations to construct a *single* interface that describes the composed system $PE||PB$ in Fig. 3.7. The new interface states the input assumptions of the system only in terms of the output guarantees that the system expects from its environment:

$$\alpha_x^{lA} = (\rho^{uG} - B_0) \oslash \beta^{lG} \ , \tag{3.34}$$
$$\alpha_x^{uA} = \min\{\beta^{lG} \otimes (\rho^{lG} + B_{\max} - B_0), \beta^{lG} + b_{\max}\} \ , \tag{3.35}$$
$$\beta^{lA} = \max\{(\rho^{uG} - B_0) \oslash \alpha_x^{lG}, \alpha_x^{uG} \oslash (\rho^{lG} + B_{\max} - B_0), \alpha_x^{uG} - b_{\max}\} \ , \tag{3.36}$$
$$\rho^{lA} = (\alpha_x^{uG} \oslash \beta^{lG}) - (B_{\max} - B_0) \ , \tag{3.37}$$
$$\rho^{uA} = (\alpha_x^{lG} \otimes \beta^{lG}) + B_0 \ . \tag{3.38}$$

3.7.2 Composition Using Fixed Priority Schedulers

The system shown in Fig. 3.7 can be composed with other processing elements which, for example, have more than one task that needs to be executed for each single data item. It is also possible to have systems with processing elements that process more than one data stream. This is the case for the application shown in Fig. 3.1 where in PE_2 there are two components which share the service provided by PE_2 using fixed priority scheduling. The corresponding abstract components and interface connections are depicted in Fig. 3.2.

If a component shares the service it receives from a processing element, with another lower-priority component, this remaining service is bounded by (3.6). In terms of output guaranteed values, this can be expressed as:

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

In order to obtain the input assumptions of a component that shares its provided service with lower priority components, we will need to use the inverses of the RT operator[2]. Most of the relations for the input assumptions are the same as the ones for a single PE. However,

[2]See Appendix B for the definitions of the pseudo-inverses of the RT operator, and for more details refer to [WT05b].

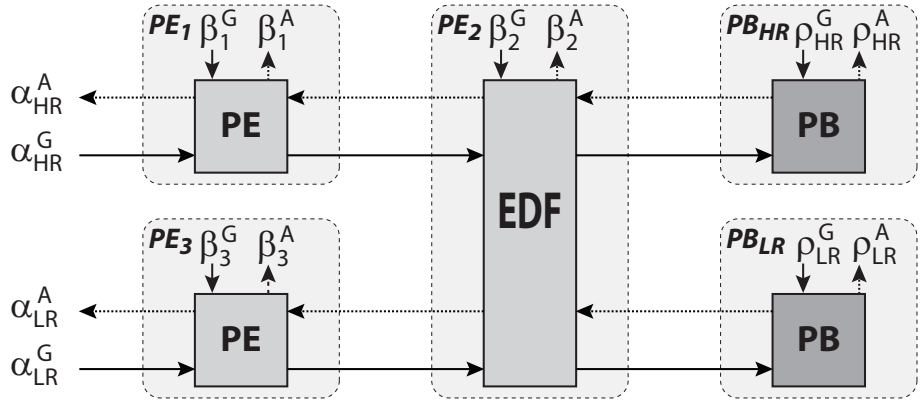

Fig. 3.8: Interface model for the architecture in Fig. 3.1 with EDF scheduling in PE_2

equations (3.28) and (3.29) change because of taking into account the pseudo-inverse of (3.6). This is written as follows:

$$\begin{aligned} \alpha_x^{uA} &= \min\{\beta^{lG} \otimes \alpha_y^{uA}, \beta^{lG} + b_{\max}, RT^{-\alpha}(\beta'^{lA}, \beta^{lG})\} \,, \\ \beta^{lA} &= \max\{\alpha_y^{lA} \oslash \alpha_x^{lG}, \alpha_x^{uG} \oslash \alpha_y^{uA}, \alpha_x^{uG} - b_{\max}, RT^{-\beta}(\beta'^{lA}, \alpha_x^{uG})\} \,. \end{aligned} \tag{3.39}$$

3.7.3 Earliest Deadline First Scheduling

It was shown in Section 3.4 that it is possible to model abstract components which process several data streams using an earliest deadline first scheduling policy. For example, the application shown in Fig. 3.1 could be changed by substituting the fixed priority scheduler in PE_2 with an EDF one. The corresponding interface model is shown in Fig. 3.8.

Again, what we need to do is determine the relations between all interface variables. This is similar to what was done for the processing element PE and the playout buffer PB in Sections 3.7.1.1 and 3.7.1.2, respectively. The interfaces corresponding to a generic EDF component which processes N input data streams are shown in Fig. 3.9.

Equations describing the output guarantees are again equivalent to those for the abstract component, i.e. (3.13) and (3.14). They only need to be expressed in terms of interface guarantees instead of values as follows:

$$\alpha_{yi}^{uG}(\Delta) = r(\alpha_{xi}^{uG}, -(D_i - \mathrm{BCET}_i), \Delta) \qquad \forall i \,, \tag{3.40}$$

$$\alpha_{yi}^{lG}(\Delta) = r(\alpha_{xi}^{lG}, (D_i - \mathrm{BCET}_i), \Delta) \qquad \forall i \,, \tag{3.41}$$

using the definition of the shift function r in (3.11).

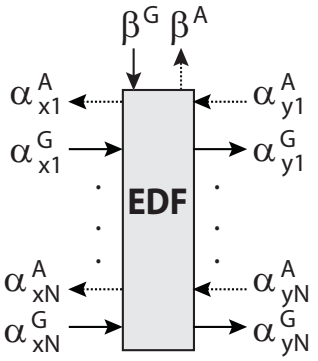

Fig. 3.9: Interface model for a generic EDF component processing N data streams

Similarly, for the resource and buffer constraints, (3.12) and (3.15), we obtain:

$$\sum_{i=1}^{N} r(\alpha_{xi}^{uG}, D_i, \Delta) \leq \beta^{lG}(\Delta) \qquad \forall \Delta \in \mathbb{R}^{\geq 0}\ , \tag{3.42}$$

$$\alpha_{xi}^{uG}(D_i) \leq b_{i\,\max} \qquad \forall i\ . \tag{3.43}$$

Determining the input assumptions of the EDF component also involves finding the pseudo-inverse functions of the relations. Finding the input assumes for the upper arrival curves involves inverting relations (3.12), (3.13), and (3.15). Again, we need to compute the largest upper curves for which the relations still hold. Using (3.13), we can determine for the upper curve of each input data stream the following inequality:

$$\alpha_{xi}^{uA}(\Delta) \leq s(\alpha_{yi}^{uA}, (D_i - \mathrm{BCET}_i), \Delta) \qquad \forall \Delta \in \mathbb{R}^{\geq 0} \quad \forall i\ , \tag{3.44}$$

where $s(\alpha, c, \Delta)$ is the function:

$$s(\alpha, c, \Delta) = \begin{cases} \alpha(\Delta - c) & \text{if } \Delta > c \\ \alpha(0^+) & \text{if } 0 < \Delta \leq c \\ 0 & \text{if } \Delta = 0 \end{cases}$$

with $\alpha(0^+) = \lim_{\epsilon \to 0}\{\alpha(\epsilon)\}$.

Similarly, considering the buffer constraint (3.15) for each input data stream, we find:

$$\alpha_{xi}^{uA}(\Delta) \leq t(D_i, b_{i\,\max}, \Delta) \qquad \forall \Delta \in \mathbb{R}^{\geq 0} \quad \forall i\ , \tag{3.45}$$

where $t(d, b, \Delta)$ is the function:

$$t(d, b, \Delta) = \begin{cases} \infty & \text{if } \Delta > d \\ b & \text{if } 0 < \Delta \leq d \\ 0 & \text{if } \Delta = 0 \end{cases}$$

Finally, inverting the resource constraint (3.12) yields:

$$\alpha_{xi}^{uA}(\Delta) \leq \beta^{lG}(\Delta + D_i) - \sum_{\substack{j=1 \\ j \neq i}}^{N} r(\alpha_{xj}^{uG}, (D_j - D_i), \Delta) \qquad \forall \Delta \in \mathbb{R}^{\geq 0} \quad \forall i \ , \quad (3.46)$$

using the definition of the shift function r in (3.11).

Combining inequalities (3.44), (3.45), and (3.46), for the input assume of the upper curve of each input data stream, we finally find for all i:

$$\alpha_{xi}^{uA}(\Delta) = \min \left\{ s(\alpha_{yi}^{uA}, (D_i - \mathrm{BCET}_i), \Delta),\ t(D_i, b_{i\,\max}, \Delta), \right.$$
$$\left. \beta^{lG}(\Delta + D_i) - \sum_{\substack{j=1 \\ j \neq i}}^{N} r(\alpha_{xj}^{uG}, (D_j - D_i), \Delta) \right\} \qquad \forall i \ . \quad (3.47)$$

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 relation (3.14) or $\alpha_{xi}^{lA}(\Delta) \geq \alpha_{yi}^{lA}(\Delta + (D_i - \mathrm{BCET}_i))$ for all i. Therefore, we can determine the following assume interface function for the lower curve of each input data stream:

$$\alpha_{xi}^{lA}(\Delta) = r(\alpha_{yi}^{lA}, -(D_i - \mathrm{BCET}_i), \Delta) \qquad \forall i \ , \quad (3.48)$$

using the shift function r as defined in (3.11).

Similarly, for the assume of the lower service curve we invert relation (3.12) which yields the inequality $\beta^{lA}(\Delta) \geq \sum_{i=1}^{N} r(\alpha_{xi}^{uG}, D_i, \Delta)$. Therefore, the input assume for the lower service curve of an EDF component can be determined as:

$$\beta^{lA}(\Delta) = \sum_{i=1}^{N} r(\alpha_{xi}^{uG}, D_i, \Delta) \ . \quad (3.49)$$

It is also possible to combine the relations for an EDF component with those for a playout buffer PB in the same way it was done for a single processing component PE and a PB component. Note that the component-based approach described in this chapter is suitable for design and analysis of any complex hierarchial scheduling method in a distributed implementation, see for example [WT06a].

3.8 Generalizing the Interface Model

The theory of Rate Interfaces is not confined to the interface models presented in the previous sections, but can be generalized into various

directions. In the following sections we only show some of the possible generalizations.

3.8.1 Workload Characterization

So far, the main characterization of streams and the service provided by a resource have been the arrival and service curves α and β. As explained in Section 3.4, they map the domain of time intervals to that of events. On the other hand, computing resources are usually characterized by physical quantities like processing cycles per second. The relation between these physical quantities and the abstract notion of events can be given by workload curves as defined in Section 3.4. Here, $\gamma(v) = (\gamma^l(v), \gamma^u(v))$ denotes the minimal and maximal amount of resource units (workload) which is needed to process any sequence of v consecutive events.

We denote as $\overline{\beta}(\Delta) = (\overline{\beta^l}(\Delta), \overline{\beta^u}(\Delta))$ the minimum and the maximum available resource units in any time interval of length Δ. If event-based service curves are given, then valid resource-based service curves can be determined as:

$$\overline{\beta^u}(\Delta) = \gamma^u(\beta^u(\Delta)) , \qquad \overline{\beta^l}(\Delta) = \gamma^l(\beta^l(\Delta)) . \tag{3.50}$$

Conversely, if resource-based service curves are given, then valid event-based curves are:

$$\beta^u(\Delta) = \gamma^{-l}(\overline{\beta^u}(\Delta)) , \qquad \beta^l(\Delta) = \gamma^{-u}(\overline{\beta^l}(\Delta)) , \tag{3.51}$$

where

$$\gamma^{-u}(r) = \sup\{v \mid \gamma^u(v) \leq r\} , \qquad \gamma^{-l}(r) = \inf\{v \mid \gamma^l(v) \geq r\} .$$

These inverse workload curves $\gamma^{-u}(r)$ and $\gamma^{-l}(r)$ represent the minimal and the maximal number of events that can be processed with r resource units.

In a similar way, we denote as $\overline{\alpha}(\Delta) = (\overline{\alpha^l}(\Delta), \overline{\alpha^u}(\Delta))$ the minimum and the maximum workload arriving on an event stream in any time interval of length Δ. Then, the conversion between the event-based and resource-based arrival curves can be given as follows:

$$\begin{aligned} \overline{\alpha^u}(\Delta) &= \gamma^u(\alpha^u(\Delta)) , & \overline{\alpha^l}(\Delta) &= \gamma^l(\alpha^l(\Delta)) , \\ \alpha^u(\Delta) &= \gamma^{-l}(\overline{\alpha^u}(\Delta)) , & \alpha^l(\Delta) &= \gamma^{-u}(\overline{\alpha^l}(\Delta)) . \end{aligned}$$

The workload curves can be obtained from the system specification using various methods, see also [MKT04]:

- Let us suppose that the resource unit we are considering is the execution demand of a task, e.g. in terms of processing cycles. In this case we can determine the workload curves as a function of the worst-case execution demand (WCED) and the best-case execution demand (BCED) of the task associated with each stream:

$$\gamma^u(v) = v \cdot \text{WCED} , \qquad \gamma^l(v) = v \cdot \text{BCED} ,$$
$$\gamma^{-u}(r) = \lfloor r/\text{WCED} \rfloor , \qquad \gamma^{-l}(r) = \lceil r/\text{BCED} \rceil .$$

- If a more detailed model of the task execution is known, e.g. in form of a finite state machine that executes a subfunction of the task at each transition, then one can analytically determine the workload functions. In this case, one can take advantage of the fact that not for every event the worst-case execution demand or the best-case execution demand may occur, i.e. for v successive events we have $\gamma^u(v) < v \cdot \text{WCED}$ and $\gamma^l(v) > v \cdot \text{BCED}$.

- Finally, one may determine workload curves by simulation. In this case, the execution demand of event i, $i \geq 0$, in a sample input stream is denoted as ED_i. From the definition of the workload curves, we obtain $\gamma^u(v) = \sup\{\sum_{i=k}^{i<k+v} \text{ED}_i \mid k \geq 0\}$ and $\gamma^l(v) = \inf\{\sum_{i=k}^{i<k+v} \text{ED}_i \mid k \geq 0\}$. Because of the simulation-based approach, we do not obtain worst-case results that hold for all executions of the system. They are only valid for a certain class of input streams.

Now, the rate interface equations as derived in this chapter can be rewritten such that they involve for example the resource-based service curve. Here, we only describe how to derive the relations given in Sections 3.7.1.1 and 3.7.1.2 for the simple scenario shown in Fig. 3.7. The other scenarios described in the chapter can be adapted in a similar way.

The basic equations that describe the abstract greedy processing component PE in (3.4), (3.5), (3.6) and (3.8) change to:

$$\alpha'^u = \alpha^u \oslash \gamma^{-u}(\overline{\beta^l}) , \qquad \alpha'^l = \alpha^l \otimes \gamma^{-u}(\overline{\beta^l}) ,$$
$$\overline{\beta'^l}(\Delta) = \sup_{0 \leq \lambda \leq \Delta} \left\{ \overline{\beta^l}(\lambda) - (\gamma^u(\alpha^u))(\lambda) \right\} , \qquad \alpha^u(\Delta) \leq (\gamma^{-u}(\overline{\beta^l}))(\Delta) + b_{\max} .$$

In a similar way, also the relations that describe the abstract EDF component can be adapted to the resource-based representation. Constraints (3.9) and (3.10) for the playout buffer component PB remain unchanged.

As a result, (3.25) and (3.26) are adapted to:

$$\alpha_y^{uG} = \alpha_x^{uG} \oslash \gamma^{-u}(\overline{\beta^{lG}}) , \qquad \alpha_y^{lG} = \alpha_x^{lG} \otimes \gamma^{-u}(\overline{\beta^{lG}}) .$$

By a simple replacement of the event-based service curve, we obtain the following for (3.27) and (3.28):

$$\alpha_x^{lA} = \alpha_y^{lA} \oslash \gamma^{-u}(\overline{\beta^{lG}}) ,$$
$$\alpha_x^{uA} = \min\{\gamma^{-u}(\overline{\beta^{lG}}) \otimes \alpha_y^{uA}, \gamma^{-u}(\overline{\beta^{lG}}) + b_{\max}\} .$$

Finally, the workload relations lead to:

$$\overline{\beta^{lA}}(\Delta) = \gamma^{l}(\beta^{lA}(\Delta)) , \qquad \beta^{lA} = \max\{\alpha_y^{lA} \oslash \alpha_x^{lG}, \alpha_x^{uG} \oslash \alpha_y^{uA}, \alpha_x^{uG} - b_{\max}\} .$$

The playout buffer relations (3.30–3.33) remain unchanged as only event-based quantities are involved.

As a result of this discussion, we can state that the interface rate algebra derived in this chapter can be extended towards resource-based quantities like computation time, number of execution cycles or communication bandwidth.

3.8.2 Large Interface Models

In Section 3.7 we derived the interface relations for a PE with one abstract component in (3.25)–(3.29), as well as for a PB in (3.30)–(3.33). In (3.34)–(3.38) we combine these relations to form the interface of a system consisting of a single PE and a single PB, as depicted in Fig. 3.7. Analogously one could combine the interfaces of several single PEs and PBs to model larger systems that are composed of many PEs and PBs, as for example to form the interface of the system depicted in the bottom of Fig. 3.2. By concatenating the relations of the single component interfaces we can directly obtain the interface of the larger system.

3.8.3 Scheduling Policies

As was shown in Section 3.7, the theory of Rate Interfaces allows us to express the interface of a processing element processing more then one input stream. In Sections 3.7.2 and 3.7.3, we presented the interface relations for abstract components that share the available service using fixed priority and earliest deadline first scheduling policies. It is also possible to derive the internal interface relations for various other scheduling policies such as time division multiple access (TDMA), first-in first-out (FIFO), generalized processor sharing (GPS), polling servers, and any hierarchical combination of them. It is feasible to obtain interface models of PEs that implement any of these scheduling policies, see e.g. [WT06a].

3.8.4 Rich Interfaces

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

However, we could also expose the internal buffers of abstract components at their interfaces. Analogous to (3.25)–(3.29) and (3.30)–(3.33), we could then derive the interface relations for the buffer guarantees and assumptions, using the various buffer constraints described in Section 3.7. With such richer interfaces, one could solve additional design problems. For example, given the service guarantees and assumptions as well as the arrival guarantees and assumptions of an abstract component, what is the minimum assumption on its internal buffer such that the buffer does not overflow? Moreover, if we would also expose the initial buffer fill level in the interface of a playout buffer, one could even derive the assumptions on the minimum and maximum initial buffer fill level such that the playout buffer does not overflow or underflow, given its arrival and consumption guarantees.

For a component with EDF scheduling it could be interesting to determine the deadlines that can be associated with the event streams such that all streams remain schedulable. By exposing the deadlines at the component interface, we could easily derive the minimum assumptions on the deadlines that must be associated to an event stream such that the system remains schedulable.

3.8.5 Interface for Worst-Case Traversal Times Constraints in Component Networks

Until now we have considered only component-wise constraints such as a certain buffer inside of a component should never underflow or overflow. However, we may have constraints that span a network of components such as end-to-end delay constraints or worst-case traversal time (WCTT) constraints.

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" [LBT01].

Before we continue, we need to define the computation of the WCTT for the abstract components described in Section 3.4. Following results from Network Calculus, the WCTT experienced by an event in a component of type PE, defined as its finishing time minus its arrival time, can be computed as:

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

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

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

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

For the PB component, the WCTT experienced by an event can be computed as $Del(\alpha^u, \rho^l_\tau)$ where

$$\rho^l_\tau(\Delta) = r(\rho^l, \tau, \Delta) \tag{3.52}$$

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 the PE component, meeting a delay constraint of D in the PB component would require that we have the following constraint satisfied:

$$\rho^l_\tau(\Delta) \geq r(\alpha^u, D, \Delta) \qquad \forall \Delta \in \mathbb{R}^{\geq 0} \ ,$$

where r is the shift function defined in (3.11).

For the EDF component, the WCTT experienced by an event from a stream can be computed as $Del(\alpha^u_i, \beta^l_{D_i})$ where $\beta^l_{D_i}$ is the service curve provided to each stream when the schedulability condition (3.12) for the EDF component is satisfied. The service curve is modeled with a burst-delay function defined in [LBT01] as:

$$\beta^l_{D_i}(\Delta) = \begin{cases} +\infty & \text{if } \Delta > D_i \\ 0 & \text{otherwise} \end{cases} \tag{3.53}$$

Given the above computations of the WCTT in individual abstract components, we can generalize them towards networks of components. For an input event stream bounded by α^u traversing a sequence of components which consists of a set of PEs, a set of PBs, and a set of

EDF components denoted as $\mathcal{PE}$, $\mathcal{PB}$ and $\mathcal{EDF}$, respectively, the worst-case traversal time that an event can experience can be computed as $Del(\alpha^u, \beta_{\mathcal{PE}} \otimes \rho_{\mathcal{PB}} \otimes \beta_{\mathcal{EDF}})$ with $\beta_{\mathcal{PE}} = \bigotimes_{c \in \mathcal{PE}} \beta^l_c$, $\rho_{\mathcal{PB}} = \bigotimes_{c \in \mathcal{PB}} \rho^l_{\tau\, c}$, and $\beta_{\mathcal{EDF}} = \bigotimes_{c \in \mathcal{EDF}} \beta^l_{D_i\, c}$, where β^l_c is the service availability of PE component c, $\rho^l_{\tau\, c}$ is the lower readout curve for PB component c as defined with (3.52), and $\beta^l_{D_i\, c}$ is the service availability provided to the stream served with relative deadline D_i by EDF component c as defined with (3.53).

A WCTT constraint on the sequence of components $Del(\alpha^u, \beta_{\mathcal{PE}} \otimes \rho_{\mathcal{PB}} \otimes \beta_{\mathcal{EDF}}) \leq D$ can be written as follows:

$$\beta_{\mathcal{PE}} \otimes \rho_{\mathcal{PB}} \otimes \beta_{\mathcal{EDF}} \geq r(\alpha^u, D, \Delta) \qquad \forall \Delta \in \mathbb{R}^{\geq 0}\,, \tag{3.54}$$

using the shift function r from (3.11).

Besides computing WCTT, we can develop an additional type of interface to alleviate design of systems with WCTT constraints that span networks of components. It is an interface-based interpretation of meeting a WCTT requirement with (3.54).

The 'complies to' relation $\vdash$ for this interface connection is defined as $\Pi^G(\Delta) \vdash \Pi^A(\Delta) = (\forall \Delta : \Pi^G(\Delta) \geq \Pi^A(\Delta))$, where Π^A expresses the minimum service requested from all subsequent components such that a 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 (3.54) and can be done with $\Pi^G = \beta^G_{\mathcal{PE}} \otimes \rho^G_{\mathcal{PB}} \otimes \beta^G_{\mathcal{EDF}}$. Connecting a PE component to the sequence would change the combined service to $\Pi'^G = \beta^{lG} \otimes \Pi^G$ where β^{lG} is the lower service guaranteed by the PE. Similarly, connecting a PB component we would have $\Pi'^G = \rho^{lG}_\tau \otimes \Pi^G$, where $\rho^l_\tau(\Delta)$ is the lower guaranteed shifted readout curve as defined with (3.52). For an EDF component, we have $\Pi'^G = \beta^{lG}_{D_i} \otimes \Pi^G$ where $\beta^{lG}_{D_i}$ is the service curve provided to the stream when processed with a relative deadline D_i as defined in (3.53).

From (3.54), we can also compute the assume on the combined service of a sequence of components as $\Pi^A = r(\alpha^{uG}, D, \Delta)$ which expresses the minimum necessary service in order to meet a WCTT constraint of D for the input α^{uG}. Propagating the assume value through a sequence of components can be done for the three types of components by inverting (3.54) as follows:

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

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

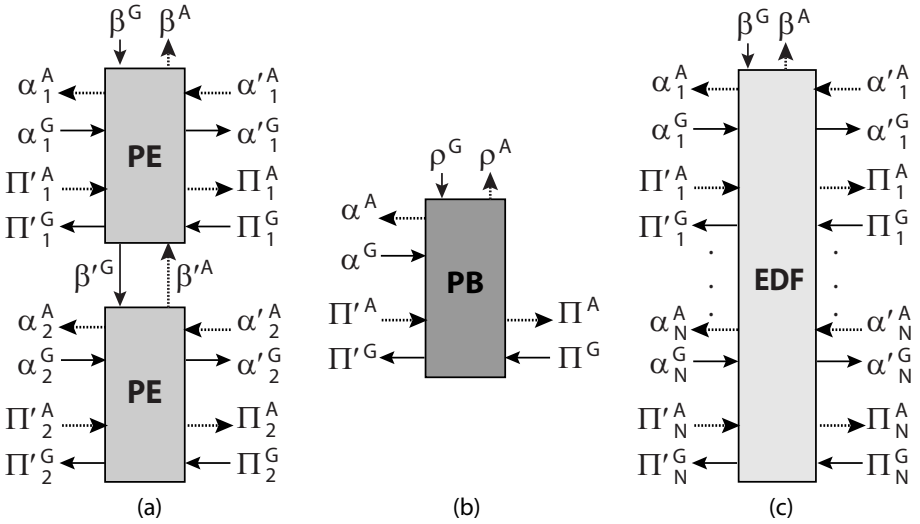

Fig. 3.10: Interface models for: **(a)** two PE components processing two streams with FP scheduling, **(b)** PB component, and **(c)** EDF component processing N streams

components Π^G. We can do this for the three types of components as follows:

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

The above constraints can be combined with the previously computed input assumes for the resources of the three components with (3.39), (3.33), and (3.49). By doing this, satisfying all interface relations of components composed in a sequence will guarantee that the WCTT constraint on the sequence of components is satisfied. The WCTT interfaces for the PE, PB, and EDF components are shown in Fig. 3.10(a)–(c).

3.9 Case Study

In this section we show how our proposed theory can be applied to the application scenario described in Section 3.3. Towards this, each processing element and playout buffer in Fig. 3.1 is considered to be an independent component and our objective is to connect them together to realize the architecture shown in the figure. In order to decide whether two components can be connected together, we would only inspect their interfaces. Two compatible interfaces implicitly guarantee that the buffers inside their respective components will never overflow, and in addition, the playout buffers will never underflow.

The main message in this section is an illustration of how the internal details of a component (e.g. its buffer size, scheduling policy,

processor frequency) are reflected (or summarized) through its rate interface. We show that if these internal details are changed then the component's rate interface also gets changed and two previously compatible components may now become incompatible (or vice versa). Further, we show how rate interfaces can help determine internal parameters of a component during design time by keeping all interface connections compatible. As mentioned earlier, this approach is in contrast to first designing/assembling the complete system and postponing the performance analysis step to the very end.

Figure 3.2 shows the component model of the architecture in Fig. 3.1 and also its interface model. Whereas the arrows in the component model (top) represent the flow of the video streams, those in the interface model (bottom) represent the assumptions and the guarantees associated with the individual components.

3.9.1 Experimental Setup

We use two different sets of video clips; the first set being made up of regular clips with moderate to high motion content and the second set being made up of clips displaying still images. The former set characterizes video streams to be viewed through the main window of the PiP application. The latter set represents content like stock quotes and upcoming program schedules, which will be viewed in the smaller secondary window (see Fig. 3.1). These two sets characterize the two streams *HR* and *LR* shown in Fig. 3.1. In our experiments, both the incoming streams have the same frame resolution of 704×576 pixels and we assume that the down-scaling for the secondary window is being done at the output device. However, for simplicity we slightly abuse the notation and refer to the stream for the main window as the high-resolution (or *HR*) stream and that for the secondary window as the low-resolution (or *LR*) stream. Both the streams arrive at the system at a constant bitrate of 8 Mbps and the playout buffers are read at a constant rate of 25 frames/second. Further, in the architecture shown in Fig. 3.1, PE_1 is set to run at a processor frequency of 1.3 GHz, PE_2 at 3 GHz and PE_3 at 1.25 GHz.

We simulated the execution of the VLD, IQ, IDCT and MC tasks for these two sets of video clips using a SimpleScalar model [ALE02] of the processing elements (with the *sim-profile* configuration and the PISA instruction set). Both of the video streams were modeled at the macroblock granularity. Note that the number of bits constituting each compressed macroblock in the input stream is variable. Hence, at the macroblock granularity, a constant bitrate stream translates into a bursty arrival pattern. Similarly, the number of processor cycles required to

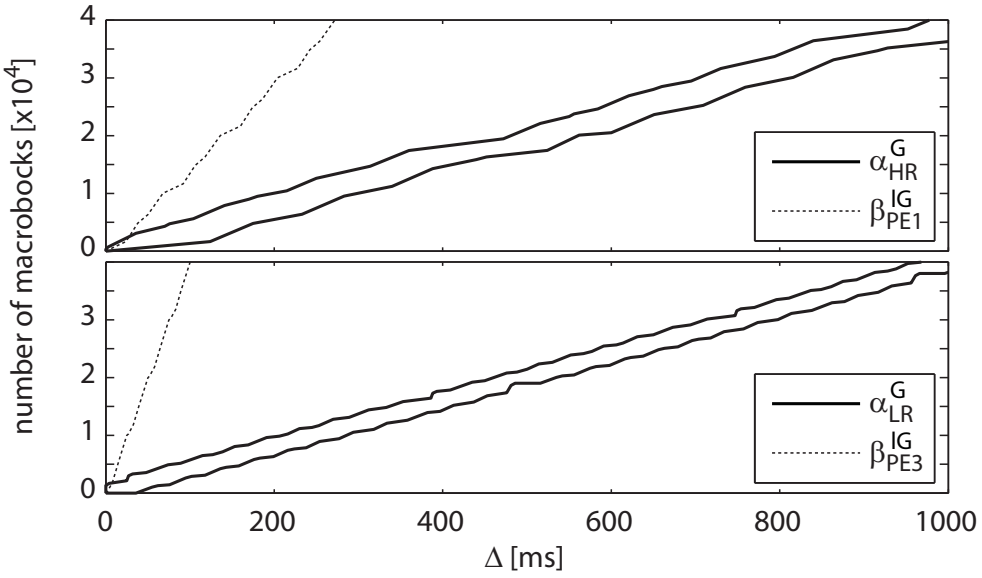

Fig. 3.11: Upper and lower bounds on the arrival process of the video streams at PE_1 and PE_3, and the guaranteed service offered by these PEs

process each macroblock is also highly variable. Hence, the service offered by a PE running at a constant frequency translates into a variable service when expressed in terms of the number of macroblocks guaranteed to be processed within any time interval of a specified length. Figure 3.11 shows the bounds on these bursty arrival patterns at the input to the system for the *HR* and *LR* streams. It also shows the guaranteed service β^G offered by PE_1 and PE_3 to these two streams. The event-based (macroblock level) service curves have been computed with workload curves that come from data recorded for the two streams during simulation. It represents the maximum number of CPU cycles needed to process one, two, and more consecutive macroblocks. The conversion has been done with equations (3.51).

3.9.2 Results

We report four different cases, each of them involving different instances of the five components shown in Fig. 3.2. For the first three cases, these instances were obtained by changing the sizes and initial fill-levels of the playout buffer components, with all the other components remaining unchanged. We show that in the first case, the components turn out to be compatible, whereas in the following two cases they are incompatible. Our main result is that this compatibility is checked by only inspecting the components' interfaces. The fourth case involves changing the scheduler

in PE_2 from one with a fixed priority policy to one with an EDF policy. Bounds on the deadlines for the two tasks running in component PE_2 are determined by keeping all interface connections of the component compatible.

Before continuing further, we would like to point out that the buffer sizes b_2 and b_4 in PE_2 (see Fig. 3.1) can be chosen independently of the other system parameters. However, they influence the rates of the incoming streams *assumed* by the component PE_2. On the other hand, for given arrival rates of the input streams and given processor frequencies, the buffer sizes b_1 and b_3 inside PE_1 and PE_3 should be of a certain minimum size for them not to overflow. Given the parameters in our experimental setup, these buffer sizes turn out to be $b_1 = 576$ macroblocks and $b_3 = 1315$ macroblocks.

For the first three cases, the buffer sizes b_2 and b_4 inside PE_2 are set to 1 and 2 video frames respectively (each frame being made up of 1584 macroblocks). In these cases, we will only be concerned with checking if the component PE_1 (in Fig. 3.2) is compatible with a partially designed system with all the other components already connected together. This is done by checking if the output rate guaranteed by PE_1 is compatible with the input rate assumed by PE_2. This compatibility check is indicated in Fig. 3.2 with a "?" mark.

3.9.2.1 Case I

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

The problem is now to check whether the component PE_1 is compatible with this partially designed system. Figure 3.12 shows the *assumption* on the input rate α^A_{HR,PE_2} expressed by PE_2's component interface. The same figure also shows the *guarantee* on the output stream rate α^G_{HR,PE_1} expressed by PE_1's interface. Here, the output guarantee is fully "enclosed" by the input assumption (i.e. they match), thereby suggesting that the two components are compatible.

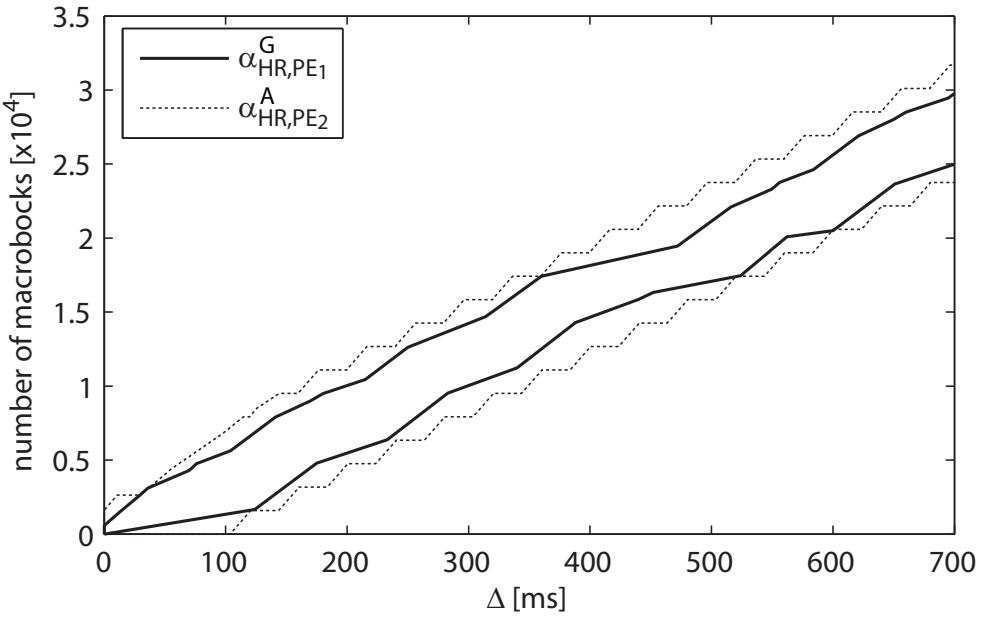

Fig. 3.12: Assumption on the input rate expressed by PE_2's interface matches the guarantee on the output rate provided by PE_1's interface. Hence, PE_1 is compatible with the partially designed system

3.9.2.2 Case II

Now we consider the same partially designed system but with the component PB_{HR}'s initial buffer fill-level reduced to 2 frames and everything else remaining unchanged. The assumptions and the guarantees on the input and output rates in this case are shown in Fig. 3.13. Note that since the component PE_1 remains unchanged, its output guarantee α^G_{HR,PE_1} is also unchanged. However, the input assumption from the partially designed system has changed and it does not satisfy the guarantee provided by PE_1 anymore. Hence, the two components are no longer compatible. If they are connected together, then the playout buffer in PB_{HR} might underflow.

3.9.2.3 Case III

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

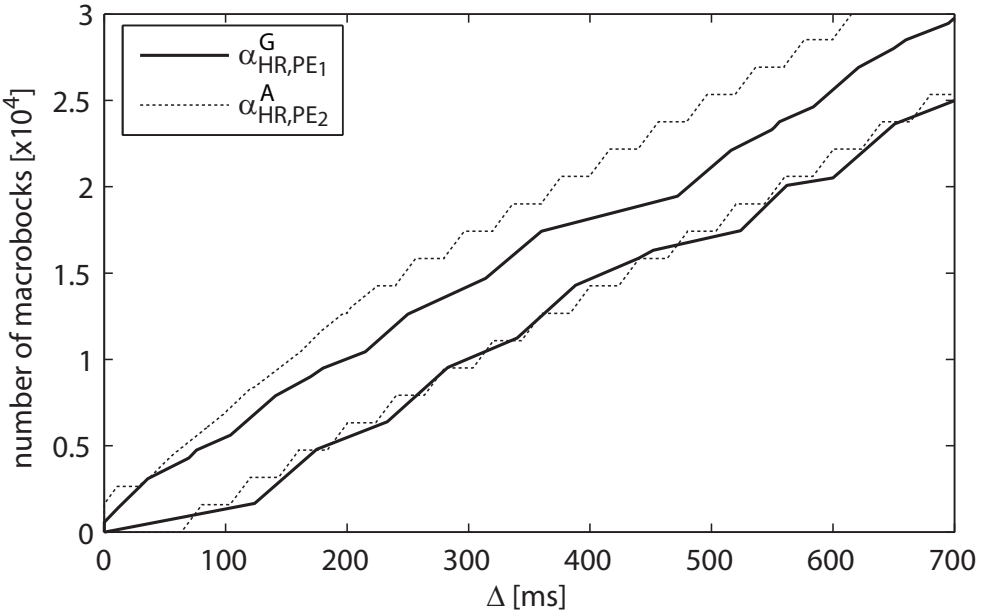

Fig. 3.13: Assumption on the input rate expressed by PE_2's interface does not match the guarantee on the output rate provided by PE_1's interface. Hence, PE_1 is *not* compatible with the partially designed system

component's interface once they have been composed together due to the constraint propagation feature of our interface theory as outlined in Section 3.6.1. Figure 3.14 shows the assumptions and the guarantees on the input and output rates in this case. Again, here they do not match. If PE_1 is connected to the system then this might potentially lead to an overflow of the buffer inside PB_{LR}.

3.9.2.4 Case IV

Here we consider a different design scenario. The buffer sizes of components PB_{HR} and PB_{LR} are both set to 8 frames and their initial fill-levels are set to 4 frames. The buffers b_2 and b_4 in component PE_2 are set respectively to 2 and 3 frames. Given the system shown in Fig. 3.2, we would like to change the scheduler in PE_2 from a fixed priority one to an EDF one. The respective system model is shown in Fig. 3.8. Substituting a component means that its internal configuration parameters need to be set-up in such a way that the interface connections of the component are compatible with the rest of the system. In this case, it would mean exploring the space of feasible relative deadlines, D_{HR} and D_{LR}, for the two tasks in PE_2 while keeping the interface connections of the component satisfied. Taking any values for the relative deadlines

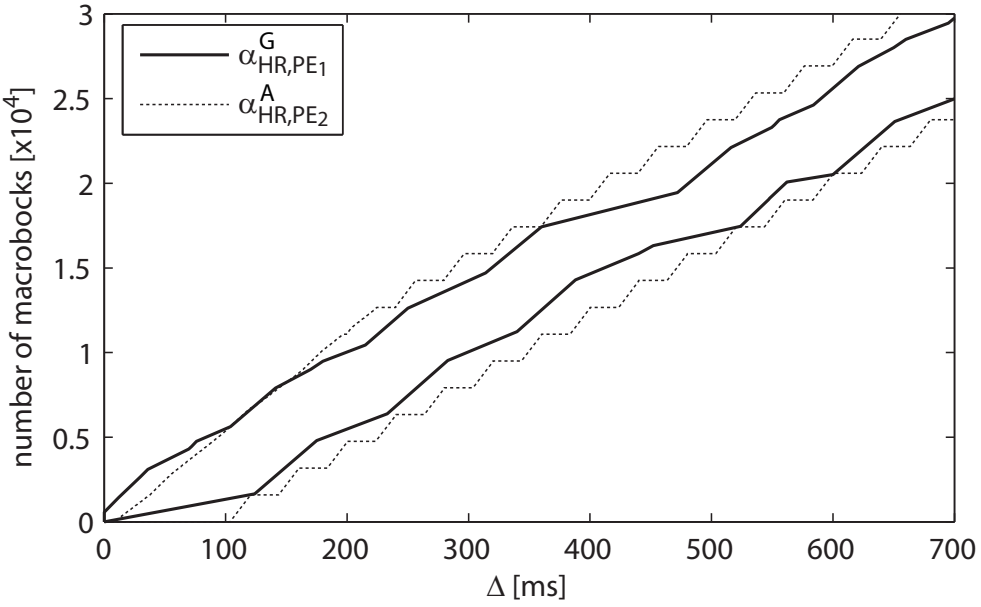

Fig. 3.14: The assumptions and guarantees do not match; hence, PE_1 is *not* compatible with the partially designed system. Here, the incompatibility is due to the component PB_{LR} being changed

from the computed intervals will guarantee that all buffers in the system meet their overflow and underflow constraints since all the interfaces of the new EDF component are still compatible with the ones of the other components.

First, we need to consider that the deadlines should be greater or equal to the worst-case execution times of the two tasks running on PE_2, i.e. $D_{HR} \geq \mathrm{WCET}_{HR}$, $D_{LR} \geq \mathrm{WCET}_{LR}$. They were found in simulation to be 0.013 ms for the high-resolution stream and 0.025 ms for the low-resolution one.

Secondly, we need to consider the interface relations of the EDF component where the deadline parameters are present. Upper and lower bounds on the deadlines are found using binary search such that the assumptions and guarantees on the interface connections of PE_2 always match, i.e. the inequalities where the deadlines are involved are satisfied.

If we consider the data stream connections between PE_2, PE_1, and PE_3 as seen in Fig. 3.8, considering inequality (3.44) separately for the high-resolution and low-resolution streams will give us, for example, lower bounds on the feasible deadlines derived from:

$$\alpha^{uG}_{HR,PE_1}(\Delta) \leq s(\alpha^{uA}_{HR,PB_{HR}}, (D_{HR} - \mathrm{BCET}_{HR}), \Delta) \qquad \forall \Delta \in \mathbb{R}^{\geq 0} ,$$
$$\alpha^{uG}_{LR,PE_1}(\Delta) \leq s(\alpha^{uA}_{LR,PB_{LR}}, (D_{LR} - \mathrm{BCET}_{LR}), \Delta) \qquad \forall \Delta \in \mathbb{R}^{\geq 0} .$$

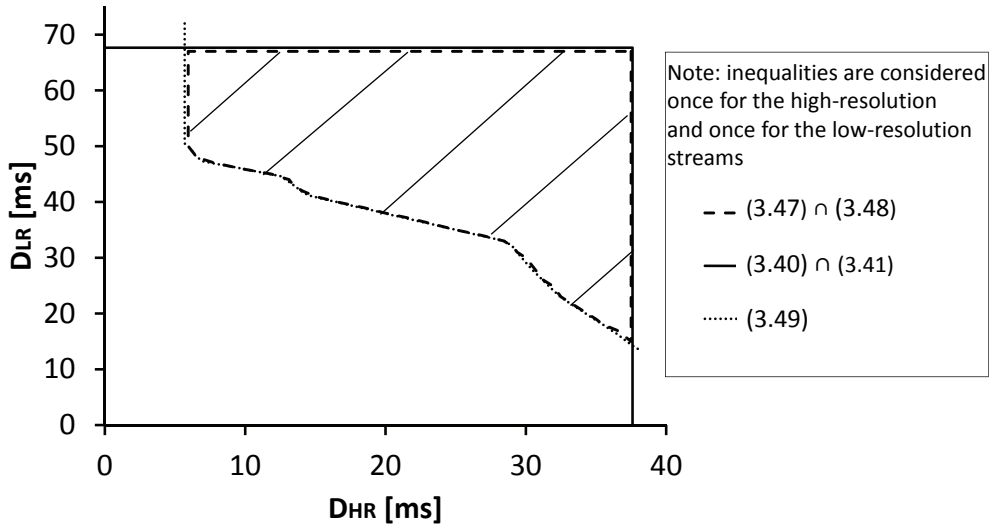

Fig. 3.15: Schedulability region for the EDF component in PE_2 in terms of the relative deadlines of the high-resolution and low-resolution tasks

Similarly, we can compute intervals for the deadlines from relations (3.45), (3.46), and (3.48) which we have to intersect in order to get the region of feasible deadlines as shown in Fig. 3.15. We need to do the same with the inequalities for the connections with the playout buffers PB_{HR} and PB_{LR}, (3.40) and (3.41), and the inequality for the service connection of PE_2 (3.49). Finally, we need to intersect all found intervals in order to get the region of feasible relative deadlines which will satisfy *all* interface connections of PE_2. This means that choosing any values for the relative deadlines from the computed region shown in Fig. 3.15 would guarantee that all constraints in the system are satisfied, e.g. no buffer in the system will underflow or overflow. The lower bounds on the deadlines which we constructed from the worst-case execution times are not drawn in the figure since they almost coincide with the x- and y-axes.

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

3.10 Discussion

In this chapter we proposed the theory of Rate Interfaces for compositional design of embedded systems whose components communicate via data streams. We showed how Rate Interfaces can be effectively checked to guarantee buffer overflow and underflow constraints. As mentioned earlier, the framework can be extended to handle other types of constraints as well, e.g. worst-case traversal time constraints.

This chapter extends previous work in two different ways. First, we cast the rate analysis problem studied in [LCM06, MKCT04] in an interface-theoretic setting and show how this leads to efficient component-based design of embedded systems. Second, it also extends our previous work on Real-Time Interfaces [WT05b, WT06a] by applying them to a distributed setup. Real-Time Interfaces are proposed for the design and analysis of real-time systems in an interface-theoretic setting. However, the previous results were restricted to uniprocessor systems. In this chapter we show that by using the Rate Interface algebra, they are extended towards the analysis and design of multiprocessor architectures.

The algebra presented in this chapter is purely stateless. It would be interesting to extend our work to setups where the processing in a component is state-dependent. Towards this, stateful interface languages like *interface automata* [dAH05] will be worth investigating.

Similarly to the results developed in Chapter 2, the Rate Interfaces algebra considers only networks of components free of directed cycles. An interface algebra for cyclic networks is a topic for future investigations.

4

Compositional Analysis for Real-Time Systems with Cyclic Dataflow

Chapters 2 and 3 deal with interface-based design frameworks for real-time systems. Such frameworks specify interfaces for system components that contain information about how the components can be used. With such frameworks a designer can decide whether two components can work together by only inspecting their interfaces without considering the internal implementation details of the components. One of the main restrictions of these frameworks is that they apply only to networks of components that are free of directed cycles. This chapter goes into the direction of developing a performance analysis method that can work with systems whose component networks contain directed cycles. Typical examples of such systems are parallel and distributed embedded systems which run control and signal processing applications. These are often specified as dataflow graphs with dependency cycles. Examples of the corresponding models of computation are marked graphs or synchronous data flow graphs.

Performance analysis is often used in the exploration of different implementation alternatives or in order to provide guarantees on the timing behavior. This chapter describes a new approach to the compositional performance analysis of cyclic dataflow graphs as existing component-based analysis methods are not able to faithfully deal with cycles in the event flows. The new method results in tight bounds on essential quantities such as buffer sizes, end-to-end delays, and throughput. Because of the generality of the approach, one can analyze not only systems that contain cyclic event flows, but also implementations that contain buffers with finite sizes that produce system-wide back-pressure caused by blocking-write semantics. The embedding of the novel approach into the Modular Performance Analysis (MPA) [CKT03b] method allows the analysis of distributed implementations that use

resource sharing mechanisms such as fixed priority (FP) scheduling and time division multiple access (TDMA).

4.1 Introduction

Applications that are implemented on distributed embedded systems can often be specified using dataflow graphs where nodes correspond to processes, and edges correspond to communication channels with first-in first-out (FIFO) buffer semantics. In particular, this observation holds if the underlying algorithms perform computations on streaming data which is common for control-, media-, signal-, image- and transceiver-applications. This model of computation has received a lot of interest in the past as it naturally fits to distributed implementations, for example heterogeneous multiprocessors, multiprocessor System-on-Chip (MPSoC), and large scale distributed systems in the automotive and avionics industries. There are several well-known subclasses of dataflow models such as Kahn Process Networks [Kah74], Marked Graphs [Mur89], and Synchronous Dataflow Graphs [LM87], for an overview see [LP95]. Many results are available concerning their deadlock behavior, schedulability, and mapping onto multiprocessor systems [BML99, SB00].

The performance analysis of applications that have been mapped onto distributed or parallel computation and communication platforms has received much attention recently, see e.g. [CKT03b, JE03, PEP02, TC94]. It enables the analysis of essential system characteristics such as end-to-end delays, upper bounds on buffer spaces, and throughput. It is based on information about the worst-case execution times, communication times, and the resource sharing strategies. The formal analysis can be used for design space exploration, e.g. binding of processes to computing resources, mapping of channels to communication paths, and selecting scheduling strategies, or for final verification of system properties after the design step.

In many of the above mentioned application domains we are faced with applications that contain cyclic dependence behavior where the result of a certain process output may depend on previous outputs of the same process, possibly transformed via a set of intermediate processes. Such applications exhibit iterative behavior that is combined with loop carried dependencies. Another prominent example is related to the use of finite buffers in the implementation of a given application which is usually modeled as a one which has infinite buffers but contains additional cyclic dependencies.

However, the analysis of cycles in the dataflow of applications poses tremendous difficulties for performance analysis, in particular for any

modular and component-based approach. The cycles in the information flow between the individual processes of an application lead to global, system-wide state dependencies. As a result, the timing behavior of a process (and as a result its use of the available resources) not only depends on predecessor processes that provide the data streams that are to be processed, but also on successor processes and the process itself. A typical special case is the use of finite buffers with blocking-write semantics which can be summarized as follows: if a buffer is full, it puts a back-pressure to preceding processes and may cause a system-wide slow-down or even blocking. Ignoring dependency cycles, for example by just cutting them or by replacing finite buffers by infinite ones, would lead to unsafe performance analysis results.

Following the above discussion, there is a need for extending the model of computation that can be handled efficiently by the modular component-based performance analysis methods towards systems with cyclic behavior in the event flows.

The problem statement of this chapter can be formulated as follows: Given is a set of marked graphs, as depicted in Fig. 4.1, that share a set of computation and communication devices by means of FP and TDMA scheduling policies, and show a complex interaction with the environment. Marked graph 1 (MG1) has finite buffers between processes $v_1 \rightarrow v_2$ and $v_2 \rightarrow v_3$ limited to maximal sizes of 1 and 2, respectively. Similarly, marked graph 2 (MG2) has finite buffers and shares the same execution platform with MG1. We would like to determine essential system characteristics for the two graphs such as end-to-end delays, buffer sizes, and throughput.

In summary, the chapter presents the following new results:

- Component-based performance analysis methods are extended to the class of marked graphs. Unlike other known methods, the approach takes into account a general model for resource interaction based on the concept of service curves that cover traditional resource models such as periodic and bounded delay as special cases, and a general data stream model based on arrival curves that cover traditional stream models such as periodic, sporadic, periodic with jitter, and periodic with bursts as special cases.

- Performance bounds obtained with the newly described method have higher accuracy than previously known methods.

- The analysis covers systems with cyclic data dependencies, finite buffer sizes, non-deterministic resource behavior, TDMA and FP scheduling policies. It can also be embedded into compositional frameworks such as the SymTA/S [JE03] or the MPA [CKT03b].

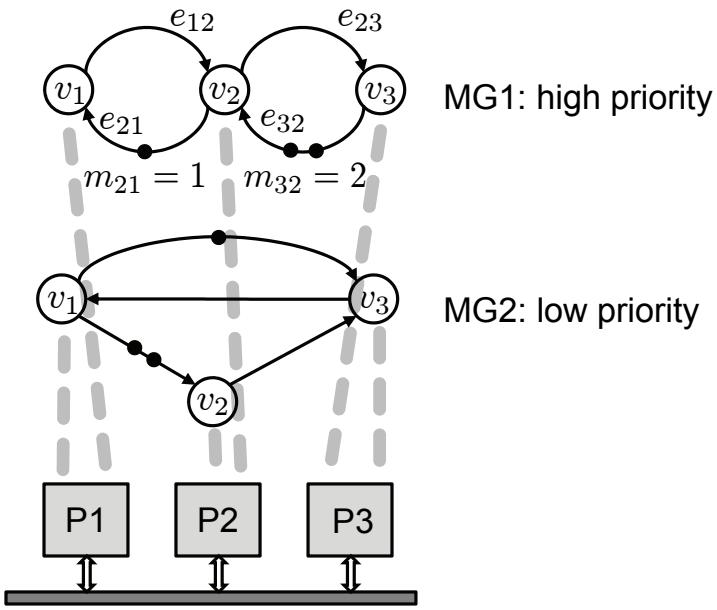

Fig. 4.1: Visualization of problem statement where two marked graphs are mapped to a distributed platform and the individual processes share the available resources using some scheduling scheme such as FP or TDMA

- Experimental results are provided that show the applicability of the new method to selected case studies and the advantages with respect to known approaches in terms of accuracy of the results.

The chapter describes a stepwise abstraction that leads from a characterization of a marked graph in time domain to an abstract representation in *time interval* domain which is then used to (a) determine essential performance indicators and to (b) embed the analysis into a compositional framework. Section 4.2 briefly discusses some of the related work. Section 4.3 contains the time domain characterization and introduces the essential notation of a service function to describe resources. Section 4.4 introduces an abstraction of the service function, i.e. it represents resource capabilities in the time interval domain using service curves, and analyzes marked graphs under this abstraction. Section 4.5 introduces the final abstraction, namely the representation of data streams in the time interval domain using arrival curves which is the main prerequisite for performance analysis and determining bounds on real-time characteristics such as buffer sizes, end-to-end delays, and throughput. Section 4.6 contains the experimental results that show the applicability and the tightness of the analysis. Finally, Section 4.7 concludes this chapter with a brief summary and a discussion.

4.2 Related Work

The present chapter specifically deals with a subclass of dataflow graphs called marked graphs, see e.g. [Mur89, Rei68]. They are characterized by the fact that each process can fire if there is at least one token in each input queue and the firing adds one output token to each output queue. For graphs where each process has a fixed delay (execution time), there are many results in the literature that characterize the timing behavior. They all suppose that there is a fixed deterministic processing time for tokens in each node. Early results in [Rei68] have been generalized and connected to eigenvalue problems in max-plus algebra, see [BOQC92, CDQV85] and more recently [GGS+06]. The results are not directly applicable to more complex interactions with the resources as envisioned in this chapter: non-deterministic delays, various resource sharing mechanisms such as FP and TDMA, and non-deterministic timing behavior of input streams.

The class of synchronous dataflow graphs (SDF) has been introduced in [LM87] as an untimed model. Unlike marked graphs, they are characterized by fixed token consumption and production rates that can be different than 1. Because of the practical importance of this model of computation, many results are available that describe properties of an implementation on single- and multi-processor systems. The processes in an SDF graph, also called actors, are annotated with fixed execution times or bounds on the execution times for the purpose of performance analysis [SB00]. The above mentioned restrictions on the scope of the performance analysis for marked graphs also hold here.

Very often, SDF graphs are converted to equivalent marked graphs, also denoted as homogeneous SDF graphs (HSDF) [SB00], for the purpose of performance analysis. The same method can be used by the analysis framework described in this chapter. Therefore, the new results can be generalized to the class of SDF graphs as well.

Systems with finite buffers can be modeled as marked graphs by adding to each edge with finite buffer an edge in the opposite direction with initial tokens that represent the available capacity of the buffer. Based on this concept, there have been several results based on the classical fixed delay models, e.g. computing buffer sizes under throughput constraints, see [WBJS06], and computing throughput while respecting sequence constraints by additional edges, see [PBB+03]. In all of these cases, resources are not explicitly modeled and therefore: (a) only limited resource sharing methods can be analyzed, and (b) modularity and composability is limited.

More complex interactions between resources and process executions can be faithfully modeled using the closely related concepts of dioids [BOQC92] and Network Calculus [Cha00, Cru91a, LBT01]. The concepts

of arrival and service curves allow a much more general modeling of the system environment and have been applied to model communication networks. In [BJLL06], results from [Cha00] have been applied to chains of processes with finite buffer sizes. Recently, very similar results have been described in [BPC09]. These results are restricted only to systems with finite buffer sizes, use course-grained approximations of resources as the upper bounds on processing and communication capabilities are set to infinity, and they are not compositional in terms of resources, i.e. they are pessimistic for systems with resource sharing.

The SymTA/S approach [JE03] has been extended towards cyclic data dependencies in [JRE05]. The analysis is based on classical real-time analysis, i.e. worst-case response times. By iterating relations for individual processes, the overall system behavior is obtained, see also [TC94]. The approach is limited in terms of traffic models, i.e. periodic with jitter and bursts, as well as in terms of resource models. Recently, the SymTA/S approach has been extended towards the class of SDF graphs, see [SSE07]. Here, an SDF application is encapsulated by providing input and output interfaces to it. The analysis of the dataflow graph itself is based on the classical results described above as well as simulation. Therefore, they are restricted to simple delay-based resource interactions with upper and lower bounds on the execution times.

Recently, the above models and methods have been generalized to distributed real-time systems, denoted as Real-Time Calculus (RTC) and Modular Performance Analysis (MPA) [CKT03b, WTVL06]. The method allows to consider complex communication and computation resource models, scheduling policies such as FP, TDMA, earliest deadline first (EDF), and hierarchical scheduling. On the other hand, complex state-dependent behavior such as cyclic data-dependencies as in marked graphs can not be modeled as well as implementations with finite buffer sizes. Recently, there have been extensions towards cyclic *resource* dependencies [JPTY08], which do not extend directly to cyclic dataflow dependencies as described in this chapter. In addition, the MPA framework has been extended towards AND/OR activations as described in [HT07] which are essential components in dataflow graph representations however, cycles in the dataflow have not been considered.

4.3 Model Definition

In this section, we will define the basic elements of the analysis framework. A system under analysis will be modeled as a *marked graph*, i.e. as a set of processes that (a) communicate via FIFO buffers with unlimited capacity, and (b) at the time of firing, a process consumes one

token from all of its inputs, and produces one token to all of its outputs. Finite size buffers will be modeled using cycles in the dataflow graph, see Section 4.6. Moreover, other forms of dataflow graphs (such as SDF) can be converted into marked graphs for analysis purposes, see [SB00].

4.3.1 Dataflow Graph

Let us first define a generic dataflow graph, i.e. the basic underlying model of the forthcoming analysis, see also Fig. 4.1.

Definition 4.1: (Dataflow Graph) *A dataflow graph* (V, E, M) *is defined as a set of processes* $v \in V$ *and a set of channels* $e \in E$ *where* $E \subseteq V \times V$*. To each channel there is associated a number of initial tokens* $M : E \to \mathbb{R}^{\geq 0}$*, i.e.* $m_{ij} \in \mathbb{R}^{\geq 0}$ *denotes the number of tokens associated with channel* $e_{ij} = (v_i, v_j)$ *that connects process* $v_i \in V$ *with process* $v_j \in V$*.*

The term 'token' is used in a very general sense. It should be interpreted as any amount of data, not necessarily integer. This way, we will be able to model systems in a flow-based as well as in a discrete-event setting.

It will be useful to assign input and output ports to each process $v_i \in V$. We denote the input port of v_i associated to channel $e_{ji} = (v_j, v_i)$ as (j, i), and the output port associated to $e_{ik} = (v_i, v_k)$ as (i, k).

4.3.2 Arrival Functions

The timing properties of an event stream can be described using the concept of an arrival function R: $R(t) \in \mathbb{R}^{\geq 0}$ which denotes the number of tokens that arrived in the time interval $[0, t)$, $t > 0$, and $R(0)$ denotes the initial number of tokens in the stream.

It will be useful for the analysis if we partially order the set of all arrival functions. In particular, we say that $R \geq R'$ if and only if $R(t) \geq R'(t)$ for all $t \geq 0$. If we are dealing with n-dimensional vectors of arrival functions $R = (R_i : i = 1, \ldots, n)$, then we say that $R \geq R'$ if and only if $R_i(t) \geq R'_i(t)$ for all $t \geq 0$, $i = 1, \ldots, n$.

It is known from lattice theory, see e.g. [DP02], page 63, that the set of arrival functions ordered by $\geq$ as defined above forms a complete lattice. The bottom $\bot$ and top $\top$ element of the set are defined as 0 and ∞ for all $t \geq 0$, respectively, where $\top \geq R \geq \bot$ for all arrival functions R. The 'complete' means that there exists a least upper bound and a greatest lower bound for each finite or infinite subset of arrival functions.

Example 4.2: *Figure 4.2 shows two examples of arrival functions. In Fig. 4.2(a)* R_1 *represents a periodic arrival pattern of discrete tokens with period* p*, and in*

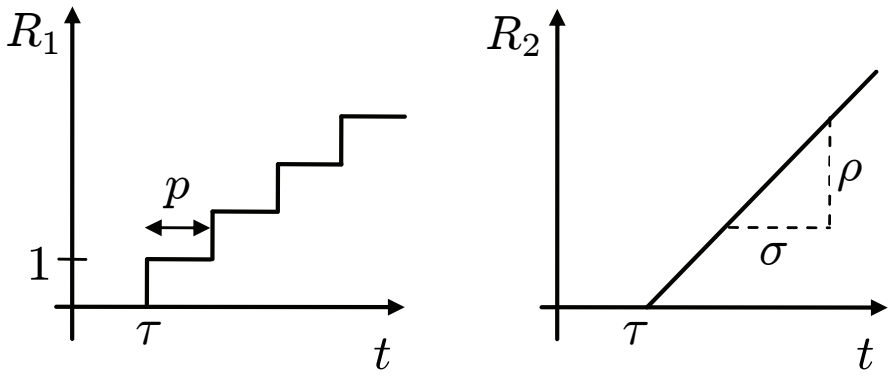

Fig. 4.2: Two simple arrival functions: **(a)** discrete periodic arrivals, **(b)** continuous flow arrival

Fig. 4.2(b) R_2 represents a continuous flow with rate ρ/σ. In both cases, the streams of tokens start at time τ.

4.3.3 Processes and Mappings

The operation of a single process v_i can be described as the mapping from a vector of input arrival functions to a vector of output arrival functions. The input arrival function R^{in}_{ji} is associated to an input port (j,i) of v_i and the output arrival function R^{out}_{ik} is associated to an output port (i,k).

Definition 4.3: (Process Mapping) *A process $v_i \in V$ with n input ports and m output ports maps an n-dimensional vector of input arrival functions R^{in} to an m-dimensional vector of output arrival functions R^{out} by means of a deterministic mapping Π_i, i.e. $R^{out} = \Pi_i \circ R^{in}$ where $R^{in} = (R^{in}_{ji} : e_{ji} \in E)$ and $R^{out} = (R^{out}_{ik} : e_{ik} \in E)$. We will also call the mapping Π_i, the transfer function of process v_i.*

In the following, we will restrict our discussion to the class of monotone processes. Loosely speaking, if we consider two distinct traces and we feed more tokens to a process in one of them ($\overline{R^{in}} \geq R^{in}$), than the process does produce at least as many output token as for the other one ($\overline{R^{out}} \geq R^{out}$). We say that such a process has a monotone mapping.

Definition 4.4: (Monotone Process Mapping) *The mapping Π of a monotone process satisfies: $\forall R, \overline{R} \; : \; \overline{R} \geq R \Rightarrow \Pi \circ \overline{R} \geq \Pi \circ R$.*

Note that not all possible processes satisfy this condition. Nevertheless, a large class of interesting processes are monotone, e.g. the considered class of marked graphs.

Example 4.5: *A simple process type is denoted as AND. It fires immediately when there are tokens available at all of its input ports. If we restrict it to two inputs R_1^{in} and R_2^{in}, we find its transfer function as:*

$$AND: \quad R^{out}(t) = \min\{R_1^{in}(t), R_2^{in}(t)\} \; . \tag{4.1}$$

Another process can be characterized as a 'tokenizer'. It receives fractional tokens at the input that may correspond to a partially transmitted packet or a partially executed function. But a discrete output token is only generated if the whole processing or communication of the predecessor process is finished, i.e. R^{out} is integer. We denote this process as TOK and define its transfer function as:

$$TOK: \quad R^{out}(t) = \lfloor R^{in}(t) \rfloor \; . \tag{4.2}$$

4.3.4 Service Function and Greedy Processing Components

The elementary processes described in Example 4.5 do not interact with available resources at all. On the other hand, it would be highly desirable to express the fact, that a process may need resources in order to operate on available input tokens. The concept of a service function C allows us to describe the availability of a resource (such as a processor or a communication device). $C(t) \in \mathbb{R}^{\geq 0}$ denotes the number of tokens that can be processed in the time interval $[0, t)$, $t > 0$, where $C(0) = 0$. In this chapter, the unit of the service function is the same as the one of the arrival function, more general concepts for characterization of these units are described in [MKT04] and Section 3.8.1 in Chapter 3.

Example 4.6: *Note that the concept of service functions allows us to model any complex resource behavior, i.e. the resource may be available with a resource rate of* 1 *token unit in* $[0, t_1)$ *and not available in* $[t_1, t_2)$ *which is the case when another task is running on the resource or other data are communicated, or the time slot allocated to the process has finished. This is expressed with* $C(t) = 1 \cdot t$, $0 \leq t < t_1$, *and* $C(t) = C(t_1)$, $t_1 \leq t < t_2$.

Now, let us consider a process with a single input which uses a resource. It consumes data which arrivals are described by the input arrival function $R^{in}(t)$, outputs processed data described by the output arrival function $R^{out}(t)$, and processes the data on a resource whose availability is described by means of the service function $C(t)$. It has greedy processing semantics (it is work-conserving) or in other words, input data tokens are processed always when there are resources available. Therefore, we call the corresponding process a Greedy Processing Component (GPC). It has the following formal definition.

Definition 4.7: (Greedy Processing Component) *A Greedy Processing Component (GPC) with single input arrival function R^{in}, output arrival function R^{out}, and service function C is defined by the following transfer function:*

$$GPC: \quad R^{out}(t) = \inf_{0 \le \lambda \le t} \{R^{in}(\lambda) + C(t) - C(\lambda)\} . \tag{4.3}$$

The remaining unused service from such a component is given by:

$$C'(t) = C(t) - R^{in}(t) . \tag{4.4}$$

The above definition can be related to the intuitive notion of a greedy (work-conserving) process as follows: The output between some time λ and t can not be larger than the available service: $C(t)-C(\lambda)$, and therefore, $R^{out}(t) \le R^{out}(\lambda)+C(t)-C(\lambda)$. As the component can not output more than what was available at the input, we have $R^{out}(\lambda) \le R^{in}(\lambda)$ and therefore, $R^{out}(t) \le R^{in}(\lambda) + C(t) - C(\lambda)$. There is some last time λ^* before t when the buffer was empty. At λ^*, we clearly have $R^{out}(\lambda^*) = R^{in}(\lambda^*)$. In the interval from λ^* to t, the buffer is never empty and all available resources are used to produce output tokens: $R^{out}(t) = R^{out}(\lambda^*) + C(t) - C(\lambda^*) = R^{in}(\lambda^*) + C(t) - C(\lambda^*)$. As a result, we obtain (4.3).

Obtaining (4.4) is much simpler as it expresses the semantics that resources available to a process until time t, $C(t)$, are either used to process input data, $R^{in}(t)$, or they are unused, $C'(t)$, and may be used by other lower priority processes.

Note that the above resource and timing semantics model almost all practically relevant processing and communication components, e.g. processors that operate on tasks and use queues to keep ready tasks, communication networks and buses, etc. As a result, we are not restricted to model processing time with a fixed delay. The service function can be chosen to represent a resource that is available only in certain time intervals (e.g. TDMA scheduling) or which is the remaining service after a resource has been used for other higher priority tasks (e.g. FP scheduling).

Following the above results, we can define the notion of a *Greedy Marked Graph Process*, and say that an activated Greedy Marked Graph Process simultaneously removes a token from each input channel and adds a token to each output channel with a rate that is determined by the available service. A Greedy Marked Graph Process is activated if there is a positive number of tokens in each input channel.

Using the above characterization, (4.1), and (4.3), it follows that a Greedy Marked Graph Process can be modeled as a concatenation of an AND and a GPC processes as shown in Fig. 4.3.

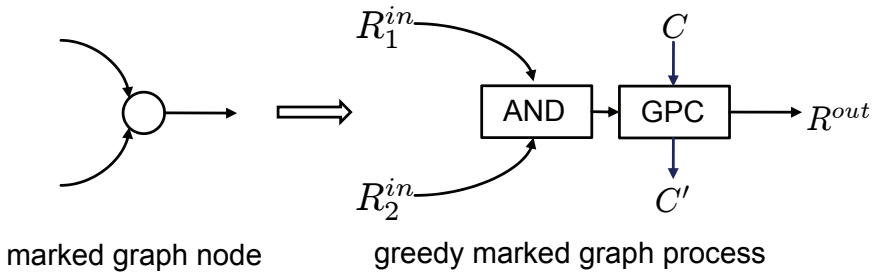

Fig. 4.3: Marked graph node and its representation as a Greedy Marked Graph Process

Example 4.8: *Let us consider now the case that a component needs a certain amount of resources to process a certain amount of token. For example, let us suppose that we have a single Greedy Marked Graph Process as depicted in Fig. 4.3 with integer tokens and each of them needs w resource units to be processed. Then we obtain*

$$R^{out}(t) = \left\lfloor \inf_{0 \le \lambda \le t} \{ \lfloor \min\{R_1^{in}(\lambda), R_2^{in}(\lambda)\} \rfloor + \frac{C(t)}{w} - \frac{C(\lambda)}{w} \} \right\rfloor . \tag{4.5}$$

Note that, all processes defined so far in (4.1), (4.2), and (4.3) are monotone as defined in Definition 4.4.

4.3.5 Execution Semantics of Marked Graphs with Greedy Processes

In this section, we move one step further towards the performance analysis of marked graphs with cyclic dependencies. To this end, we first define the operation of a network of greedy process nodes using fixed points of a system equation. Note, that we are still describing the operation of the marked graph in time domain, i.e. without any sort of abstractions.

In order to determine the semantics of a marked graph, we will derive a set of system equations. To this end, let us first define a step function with height s as follows:

$$I^s(t) = \begin{cases} 0 & \text{if } t = 0 \\ s & \text{if } t > 0 \end{cases}$$

If we now look at the semantics of a channel containing s initial tokens, it provides at its output as many tokens as have been submitted to its input plus the number of initial tokens s.

Now, we can set up a set of equations that describe the semantics of a whole marked graph (V, E, M) as follows:

$$(R_{ik}^{out} : e_{ik} \in E) = \Pi_i \circ (R_{ji}^{in} : e_{ji} \in E) \qquad \forall v_i \in V\ , \qquad (4.6)$$

$$R_{ij}^{in} = R_{ij}^{out} + I^{m_{ij}} \qquad \forall e_{ij} \in E\ , \qquad (4.7)$$

where Π_i denotes the input-output transfer function of a single greedy marked graph process v_i. If we combine (4.6) and (4.7), we get a single equation of the form

$$R = \Pi \circ R\ , \qquad (4.8)$$

where $R = (R_{ij}^{out} : e_{ij} \in E)$ is a vector of arrival functions that contains as elements the output arrival functions of all processes, and Π is the combined mapping of the whole dataflow graph. Note that, the combined mapping Π is monotone if all process mappings Π_i, $v_i \in V$, are monotone.

In order to solve (4.8), we can use results from lattice theory, see [DP02], page 187. It follows that if the mapping Π is monotone, then the fixed-point equation (4.8) has a least and a greatest fixed-points, R^l and R^u, respectively.

We can strengthen this result by assuming δ-causality for all processes of a marked graph, i.e. changes at the input of a process are not visible before a (small) time lag $\delta > 0$: if $R(s) = R'(s)$ for all $s \leq t - \delta$ then we have $(\Pi \circ R')(t) = (\Pi \circ R)(t)$. Then we can determine all solutions of (4.8) inductively, starting from initial conditions at $t = 0$. As the mappings of the processes are deterministic, the solutions to (4.8) are unique.

Example 4.9: *Let us look at the simple dataflow graph MG1 shown in Fig. 4.1 and determine the corresponding mapping $R = \Pi \circ R$ by concatenating (4.1) and (4.3) as follows:*

$$\begin{aligned}
R_{1,2}^{out}(t) &= \inf_{0 \leq \lambda \leq t} \{R_{2,1}^{out}(\lambda) + I^1(\lambda) + C_1(t) - C_1(\lambda)\}\ , \\
R_{2,3}^{out}(t) &= \inf_{0 \leq \lambda \leq t} \{\min\{R_{3,2}^{out}(\lambda) + I^2(\lambda), R_{1,2}^{out}(\lambda)\} + C_2(t) - C_2(\lambda)\}\ , \\
R_{3,2}^{out}(t) &= \inf_{0 \leq \lambda \leq t} \{R_{2,3}^{out}(\lambda) + C_3(t) - C_3(\lambda)\}\ , \\
R_{2,1}^{out}(t) &= \inf_{0 \leq \lambda \leq t} \{\min\{R_{3,2}^{out}(\lambda) + I^2(\lambda), R_{1,2}^{out}(\lambda)\} + C_2(t) - C_2(\lambda)\}\ ,
\end{aligned}$$

where the resources available to v_1, v_2, and v_3 are described by service functions $C_1(t)$, $C_2(t)$, and $C_3(t)$, respectively. The functionality corresponds to a simple processing chain with finite buffer sizes of 1 and 2, respectively.

4.4 Resource Abstraction and System Equations

In order to develop efficient methods for compositional performance analysis of marked graphs, we will need to introduce several abstractions.

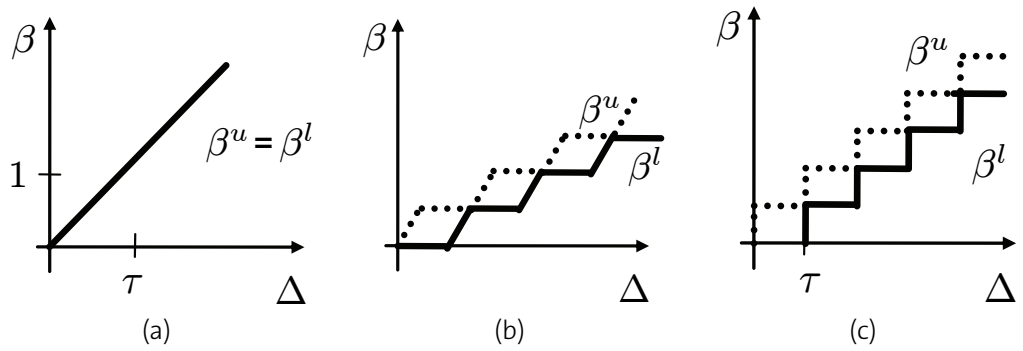

Fig. 4.4: Three examples of service curves: **(a)** fully available resource, **(b)** TDMA resource, and **(c)** locally synchronous resource

Instead of calculating the output arrival function of a process with a single service function $C_i(t)$ in time domain, we will use upper and lower bounds on $C_i(t)$. This will enable us to consider a wider class of processes and process characteristics as well as to derive computationally feasible analysis methods that provide statements about the behavior of a system under a whole set of resource behaviors. This first abstraction step introduces non-determinism as the service function is not provided explicitly anymore, but only its bounds are considered.

4.4.1 Service Curves

Following the ideas of Network Calculus [Cru91a, LBT01], we define upper and lower bounds on service functions, denoted as *service curves*. This way, we abstract from the concrete time domain and operate in the *time interval domain*. Service curves have the following formal definition.

Definition 4.10: (Service Curves) *Upper and lower service curves, β^u and β^l, map positive time intervals $\Delta \in \mathbb{R}^{\geq 0}$ to the maximal and minimal amount of available resources in any time interval of length Δ. They satisfy $\beta^u(0) = \beta^l(0) = 0$ and*

$$\beta^l(\Delta) \leq C(t+\Delta) - C(t) \leq \beta^u(\Delta) \qquad \forall t \geq 0, \Delta > 0 \; .$$

Example 4.11: *Figure 4.4 shows three examples of service curves that model: (a) a fully available resource that leads to a delay of τ for each unit of input token, (b) a TDMA resource that is available only in periodically repeating time slots, and (c) a service curve that models a locally synchronous behavior with cycle time τ, i.e. every τ a complete unit of input token can be processed.*

Now, we can upper and lower bound the mapping of a GPC as given in (4.3) by using service curves as follows:

$$R^{out}(t) \leq \inf_{0 \leq \lambda \leq t} \{R^{in}(\lambda) + \beta^u(t-\lambda)\} ,$$
$$R^{out}(t) \geq \inf_{0 \leq \lambda \leq t} \{R^{in}(\lambda) + \beta^l(t-\lambda)\} .$$

Using the min-plus algebra convolution operator $\otimes$ as defined in Appendix A, we can obtain a more concise notation, see also [Cha00, Cru91a, LBT01]:

$$R^{in} \otimes \beta^l \leq R^{out} \leq R^{in} \otimes \beta^u . \tag{4.9}$$

In other words, for a single GPC component we can bound the number of tokens that arrive in $[0, t)$ by abstracting the available service using β^u and β^l.

Example 4.12: *If we consider again that tokens are integer valued, i.e. a subsequent process can start working only if the whole workload w associated to a token has been available. We obtain for the marked graph process described with (4.5) the following upper and lower bounds on the output arrival function:*

$$(R_1^{in} \wedge R_2^{in}) \otimes \lfloor \frac{1}{w} \beta^l \rfloor \leq R^{out} \leq (R_1^{in} \wedge R_2^{in}) \otimes \frac{1}{w} \beta^u , \tag{4.10}$$

where $a \wedge b = \min\{a, b\}$.

The next step is to apply this abstraction to the whole marked graph. As we will see, we then get upper and lower bounds on the number of tokens that arrive on any channel in the graph.

4.4.2 Bounds for the Marked Graph and System Equations

So far, the (concrete) execution semantics of a marked graph has been described by the single equation (4.8). Now, we will investigate the influence of the resource abstraction introduced in (4.9).

The approach is based on replacing the mapping Π of the whole marked graph by 'larger' and 'smaller' mappings. Then the resulting arrival functions R provide upper and lower bounds on the system behavior, respectively. We say that a mapping Π^u is larger or equal than a mapping Π if the relation holds pointwise or more generally:

$$\Pi^u \geq \Pi \iff \Pi^u \circ R \geq \Pi \circ R \qquad \forall R ,$$
$$\Pi^l \leq \Pi \iff \Pi^l \circ R \leq \Pi \circ R \qquad \forall R .$$

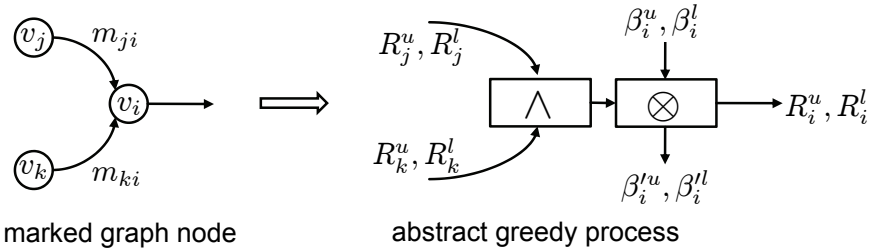

Fig. 4.5: A marked graph node with m_{ji} and m_{ki} initial tokens on input channels e_{ji} and e_{ki}, respectively, and its abstract representation

Theorem 4.13: (Upper and Lower Bounds on Output Arrival Functions in a Marked Graph with Greedy Processes) *Let $\mathcal{R}$ be a complete lattice. Let be given a monotone mapping Π with a unique fixed-point $R \in \mathcal{R}$. Define R^l to be the greatest fixed-point of $R^l = \Pi^l \circ R^l$, and R^u to be the least fixed-point of $R^u = \Pi^u \circ R^u$. Then we have:*

$$R^l \leq R \leq R^u \ .$$

Proof. Under the assumptions of the theorem, we find that the smallest fixed-points of $\underline{R} = \Pi \circ \underline{R}$ and $R^u = \Pi^u \circ R^u$ satisfy $\underline{R} \leq R^u$, see [DP02], page 199. As we have $R \geq \underline{R}$ for all R that satisfy $R = \Pi \circ R$, we find $R \geq \underline{R} \leq R^u$. As the fixed-point of Π is unique, we finally get $R = \underline{R} \leq R^u$. The proof for R^l is similar.

■

As a result of the theorem one can directly show that $R^l \leq R \leq R^u$ holds for any of the fixed-points of $R^l = \Pi^l \circ R^l$ and $R^u = \Pi^u \circ R^u$.

In other words, if we replace the mapping of a dataflow graph by one that is either not smaller or not larger, then we get upper or lower bounds on the arrival functions, i.e. on the number of tokens that passed through the processing elements at each moment in time. This result will now be used in order to replace the service functions $C(t)$ by their abstractions, the service curves $\beta^u(\Delta)$ and $\beta^l(\Delta)$.

To this end, we will determine the above mappings Π^u and Π^l explicitly from the given marked graph structure. As a result, we obtain abstract system equations whose solutions yield upper and lower bounds on the behavior of any marked graph.

Starting point is again the modeling of each process of a marked graph by a Greedy Marked Graph Process, see also (4.1), (4.3), (4.9) and Fig. 4.3.

Using the notation introduced so far, we obtain for the simple two-inputs case as depicted in Fig. 4.5 the following transfer function:

$$R_i = [(R_j + I^{m_{ji}}) \wedge (R_k + I^{m_{ki}})] \otimes \beta_i \ , \tag{4.11}$$

where β_i can be replaced by β_i^u and β_i^l in order to obtain the equations for the upper and lower bounds R_i^u and R_i^l, respectively, where $R_i^u \geq R_i \geq R_i^l$. Using elementary calculus, we can reformulate equation (4.11) to:

$$R_i = [(R_j + I^{m_{ji}}) \otimes \beta_i] \wedge [(R_k + I^{m_{ki}}) \otimes \beta_i] \,,$$
$$R_i = \beta_i \wedge [R_j \otimes (\beta_i + m_{ji})] \wedge [R_k \otimes (\beta_i + m_{ki})] \,.$$

For the following theorem, we will make use of the matrix notation $S = (S_{ij})$ where S contains elements S_{ij}, the vector notations $R = (R_i)$ and $\beta = (\beta_i)$ as well as the matrix product $C = A \otimes B$ with $c_{ij} = \bigwedge_{(k)}(a_{ik} \otimes b_{kj})$. Note again the definition of $\otimes$ in Appendix A and $a \wedge b = \min\{a, b\}$.

Theorem 4.14: (System Equations for a Marked Graph with Greedy Processes) *Given a marked graph (V, E, M) and the vectors of service curves β^u and β^l associated to the graph processes that describe bounds on the corresponding available resources, see Definition 4.10. Define the upper and lower system matrices of the graph as $S^{u,l} = (S_{ij}^{u,l})$ with elements:*

$$S_{ij}^{u,l} = \begin{cases} \beta_i^{u,l} + m_{ji} & e_{ji} \in E \\ \infty & e_{ji} \notin E \end{cases} \tag{4.12}$$

Then we can write system equations for a marked graph as follows:

$$R^u = \beta^u \wedge S^u \otimes R^u \,, \tag{4.13}$$
$$R^l = \beta^l \wedge S^l \otimes R^l \,, \tag{4.14}$$

where R^u and R^l denote upper and lower bounds on any vector of execution traces of the marked graph:

$$R^u \geq R \geq R^l \,. \tag{4.15}$$

Proof. Results follow directly from (4.11) and the structure of the matrices $S^{u,l}$ defined with (4.12).

■

Example 4.15: *If we use integer tokens and scale the resources used to process a single token as in Example 4.12, then we just need to replace β^l and β^u in equations (4.12), (4.13), and (4.14) by $\lfloor \frac{1}{w}\beta^l \rfloor$ and $\frac{1}{w}\beta^u$, respectively.*

4.4.3 Solving the System Equation

Finally, we need to determine solutions to (4.13) and (4.14) in order to determine bounds on the traces of event sequences between the processes, i.e. tight upper and lower bounds on the vector of arrival functions R in (4.15).

To this end, we make use of the corresponding results for distributive dioids as described in [BOQC92], page 193. All solutions to (4.13) and (4.14) can be determined as:

$$R = y \wedge S^* \otimes \beta \qquad \forall y \; : \; y = S \otimes y \; , \tag{4.16}$$

where for simplicity we omit the superscripts u or l that relate to (4.13) or (4.14), respectively. The matrix S^* denotes the min-closure of S which is defined as:

$$S^* = \bigwedge_{k=0}^{\infty} S^{(k)} \; , \tag{4.17}$$

where $S^{(k)} = S \otimes S^{(k-1)}$ for $k \geq 1$ and

$$S^{(0)} = \begin{pmatrix} I^\infty & \infty & \cdots \\ \infty & I^\infty & \cdots \\ \cdots & \cdots & \ddots \end{pmatrix}$$

Investigating the structure of the S^* more closely yields the following interpretation: An element S^*_{ji} of S^* is the minimal 'path length' of all (including cyclic) paths from node i to node j in the marked graph. The 'path length' is defined as the sum of all tokens along the path plus the convolution of all service curves on the path, except that of node i. If $i = j$, then the value of $S^*_{ii}(0)$ is set to 0. We will come back to the structure of S^* in more detail in Section 4.4.4.

In order to determine as tight bounds as possible, we should find now the *least fixed-point* of $R^u = \beta^u \wedge S^u \otimes R^u$ and the *greatest fixed-point* of $R^l = \beta^l \wedge S^l \otimes R^l$.

The greatest solution to $R^l = \beta^l \wedge S^l \otimes R^l$ is simply obtained as $R^l = (S^l)^* \otimes \beta^l$, see [BOQC92], page 192. In order to determine the least solution to $R^u = \beta^u \wedge S^u \otimes R^u$, we need to determine the least y with $y = S^u \otimes y$, i.e. $\underline{y} = \inf\{y \; : \; y = S^u \otimes y\}$.

The least fixed-point of $y = S^u \otimes y$ with $\underline{y} = \inf\{y \; : \; y = S^u \otimes y\}$ is given by $\underline{y} = \lim_{k\to\infty}((S^u)^{(k)} \otimes \bot)$ where $\bot(t) = 0$ for all $t \geq 0$. This result can easily be shown using elementary techniques from lattice theory, see [DP02], and by noting that S^u is monotone. This is the result of the following theorem.

Theorem 4.16: (The Least Fixed-Point Solution of $y = S \otimes y$) *The least fixed-point of $y = S \otimes y$ with $\underline{y} = \inf\{y \; : \; y = S \otimes y\}$ is given by:*

$$\underline{y} = \lim_{k\to\infty}(S^{(k)} \otimes \bot) \; .$$

Proof. Let us consider the sequence: $y_0 = \bot$, $y_1 = S \otimes \bot$, $y_2 = S^{(2)} \otimes \bot, \ldots$. Then we can easily see that it is increasing as $y_1 = S \otimes \bot \geq \bot = y_0$ and $y_k = S \otimes y_{k-1} \geq S \otimes y_{k-2} = y_{k-1}$ if $y_{k-1} \geq y_{k-2}$. Here we use the fact that S is monotone. Therefore, we have $y_0 \leq y_1 \leq y_2 \leq ....$ As a result, the limit $y' = \sup_{k\to\infty}(S^{(k)} \otimes \bot)$ exists.

Now, we will show that $\underline{y} \geq y_k$ for all $k \geq 0$, i.e. the least fixed-point is lower bounded by y_k. Of course, we have $\underline{y} \geq y_0 = \bot$. Therefore, we also find $\underline{y} = S \otimes \underline{y} \geq S \otimes y_0 = y_1$ and in general $\underline{y} = S^{(k)} \otimes \underline{y} \geq S^{(k)} \otimes y_0 = y_k$. Again, we make use of the monotonicity of S. As a result we have $\underline{y} \geq y'$.

Finally, we note that $y' = \lim_{k\to\infty}(S^{(k)} \otimes \bot)$ is actually a fixed-point, i.e. $y' = S \otimes y'$ and therefore $\underline{y} = y'$.

■

We can now show that for all meaningful marked graphs, we can simplify the calculation of the least fixed-point and therefore for R^u. We need the fact that for a given marked graph where the sum of initial tokens in each directed cycle of the network is strictly larger than 0. Then $\underline{y} = \lim_{k\to\infty}((S^u)^{(k)} \otimes \bot) = \top$, where we have $\bot(t) = 0$ and $\top(t) = \infty$ for all $t \geq 0$. More specifically, we have the following theorem. The proof uses some interpretation of $(S^u)^{(k)}$ which will be given in Section 4.4.4.

Theorem 4.17: (Determining the Upper Bound R^u in a Marked Graph with Greedy Processes) *Given a dataflow graph which models a marked graph. Suppose that the sum of initial tokens in each directed cycle of the network is strictly larger than* 0*. Then* $\underline{y} = \lim_{k\to\infty}((S^u)^{(k)} \otimes \bot) = \top$ *and therefore we have:*

$$R^u = (S^u)^* \otimes \beta^u \ . \tag{4.18}$$

Proof. If we show that $\underline{y} = \lim_{k\to\infty}((S^u)^{(k)} \otimes \bot) = \top$, then the theorem follows from (4.16). To this end, we prove that all elements $(S^u)^{(k)}_{ji}$ of $(S^u)^{(k)}$ will approach ∞ for $k \to \infty$. Let us distinguish between two situations. At first, there is no path containing a directed cycle in the dataflow graph between a pair of nodes i and j. Based on the result (4.21), we find $(S^u)^{(k)}_{ji} = \infty$ for all $k > |V|$ where $|V|$ denotes the number of nodes of the dataflow graph. Now, let us suppose that there is a path from i to j that contains a cycle. If $k > |V| \cdot K$, then any path of length k contains at least K cycles. Moreover, let δ denote the minimal sum of initial tokens on any cycle of the dataflow graph. Then we find using (4.21) that $(S^u)^{(k)}_{ji} \geq K\delta$ for $k > |V| \cdot K$. As $\delta > 0$, we find $\lim_{k\to\infty}(S^u)^{(k)}_{ji} = \infty$.

■

Note that the result also shows that the fixed points of the system equations (4.13) and (4.14) are unique for both cases, $S = S^u$ and $S = S^l$, respectively, see also [Cha00].

The main results of this section can be summarized in the following theorem.

Theorem 4.18: (Upper and Lower Bounds, R^u and R^l, in a Marked Graph with Greedy Processes) *Given a marked graph (V, E, M) and service curves β^u and β^l associated to its nodes. Suppose that the sum of initial tokens in each directed cycle of the network is strictly larger than 0. Then we can determine tight upper and lower bounds on any vector of execution traces of the marked graph where $R^u \geq R \geq R^l$ with the following arrival functions:*

$$R^l = (S^l)^* \otimes \beta^l \,, \tag{4.19}$$

$$R^u = (S^u)^* \otimes \beta^u \,, \tag{4.20}$$

where we use S^u and S^l from Theorem 4.14, and the corresponding closures $(S^u)^*$ and $(S^l)^*$ as defined with (4.17).

4.4.4 Interpretations

In this section, we will interpret equations (4.19) and (4.20) in terms of properties of the underlying marked graph, e.g. paths, initial tokens and service curves associated with the processes.

Obviously, the matrix $S^{(n)}$ plays a central role in the solution to the system equation for a marked graph. As will be shown, there is a close relation to results in max-plus algebra, see [BOQC92], page 110.

As has been defined already in (4.12), the elements of the system matrix S are given as $S_{ij} = m_{ji} + \beta_i$ if edge (j, i) exists in the marked graph. Using this definition, one can now determine the elements of $S^{(n)}$ which are denoted as $S^{(n)}_{ij}$. Here, we use the following notation:

- A path p in the marked graph is a set of connected edges, i.e. $p = \{(i_0, i_1), (i_1, i_2), \ldots, (i_{n-1}, i_n)\}$.
- The length n of a path is defined as $n = |p|$.
- The set of nodes of a path is defined as $V(p) = \{i_0, ..., i_n\}$.
- The set of all paths of length n from node i to node j is denoted as $P^n(i, j) = \{p \;:\; (|p| = n) \wedge (i = i_0) \wedge (j = i_n)\}$.
- The set of all paths from i to j is denoted as $P(i, j)$.

Based on the definition of S for $S^{(n)}$ we find:

$$S_{ij}^{(n)} = \bigwedge_{p \in P^n(j,i)} \left[\sum_{(r,s)\in p} m_{rs} + \bigotimes_{(r,s)\in p} \beta_s \right] , \tag{4.21}$$

for $n > 0$. If there exists no path of length n from j to i, then $S_{ij}^{(n)} = \infty$.

The matrix S^+ is defined as:

$$S^+ = \bigwedge_{k=1}^{\infty} S^{(k)} .$$

Using the definition of S, we find:

$$S_{ij}^{+} = \bigwedge_{p \in P(j,i)} \left[\sum_{(r,s)\in p} m_{rs} + \bigotimes_{(r,s)\in p} \beta_s \right] , \tag{4.22}$$

if there exists a path from j to i, and $S_{ij}^{+} = \infty$ otherwise. Note that the min-closure S^* can simply be determined as $S^* = S^{(0)} \otimes S^+$ (or $S_{ij}^* = S_{ij}^+$ if $i \neq j$ and $S_{ii}^* = \min\{I^\infty, S_{ii}^+\}$).

Now, we can more explicitly determine the resulting upper and lower arrival functions in the dataflow graph as stated in (4.19) and (4.20). Using elementary arithmetic, we obtain:

$$R_i = \beta_i \wedge \bigwedge_{p \in P(j,i)} \left(\sum_{(r,s)\in p} m_{rs} + \bigotimes_{k \in V(p)} \beta_k \right) , \tag{4.23}$$

where the equation holds for the upper and for the lower bounds, i.e. by adding the superscripts u or l to R and β. As an interpretation one can say that R_i is the minimal function that is larger than β_i and larger than the 'path length' of any path in the marked graph that ends at node i. Here, the 'path length' is determined as the sum of all initial tokens on the path plus the convolutions of all service curves on the path, including that of the path origin.

Example 4.19: *It is useful to derive explicit formulas for special forms of service curves. The linear approximations used here are common in the analysis of networked systems, see e.g. [LBT01]. Therefore, let us consider the special case of β_i as depicted in Fig. 4.6, where β^l is specified with a rate σ^l and a latency τ, and it is defined as:*

$$\beta^l(\Delta) = \max\{0, \sigma^l \cdot (\Delta - \tau)\} ,$$

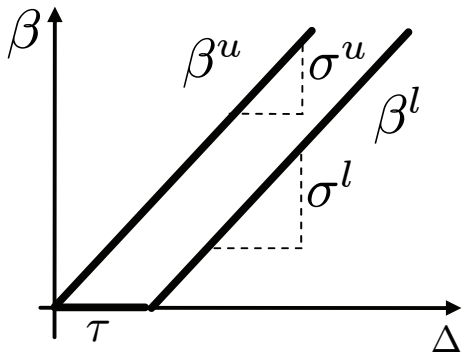

Fig. 4.6: A pair of peak rate and rate-latency service curves

and β^u is specified with a peak rate σ^u and defined as:

$$\beta^u(\Delta) = \sigma^u \cdot \Delta \ .$$

As a shorthand notation for the above, we can also write $\beta^l = (\tau, \sigma^l)$ and $\beta^u = (\sigma^u)$.

From the definition of the convolution operator we can conclude that:

$$\bigotimes_{i \in I} \beta_i^l = \left(\sum_{i \in I} \tau_i, \bigwedge_{i \in I} \sigma_i^l\right) ,$$

where I is a multiset of node indices $i \in V$. One should note that the indices i are not necessarily disjoint, i.e. the result changes if a service curve appears several times in the convolution.

For the upper service curve, the situation is even simpler. Here, we obtain:

$$\bigotimes_{i \in I} \beta_i^u = \left(\bigwedge_{i \in I} \sigma_i^u\right) .$$

Now, we can make the explicit formula (4.23) more concrete by just replacing the convolution of the service curves with the above expressions:

$$R_i^l = (\tau_i, \sigma_i^l) \wedge \bigwedge_{p \in P(j,i)} \left(\sum_{(r,s) \in p} m_{rs} + \left(\sum_{k \in V(p)} \tau_k, \bigwedge_{k \in V(p)} \sigma_k^l \right) \right) ,$$

$$R_i^u = (\sigma_i^u) \wedge \bigwedge_{p \in P(j,i)} \left(\sum_{(r,s) \in p} m_{rs} + \left(\bigwedge_{k \in V(p)} \sigma_k^u \right) \right) .$$

4.4.5 Marked Graphs with External Inputs

So far, we have been dealing with marked graphs that are autonomous, i.e. they do not have any stream of input tokens from the environment.

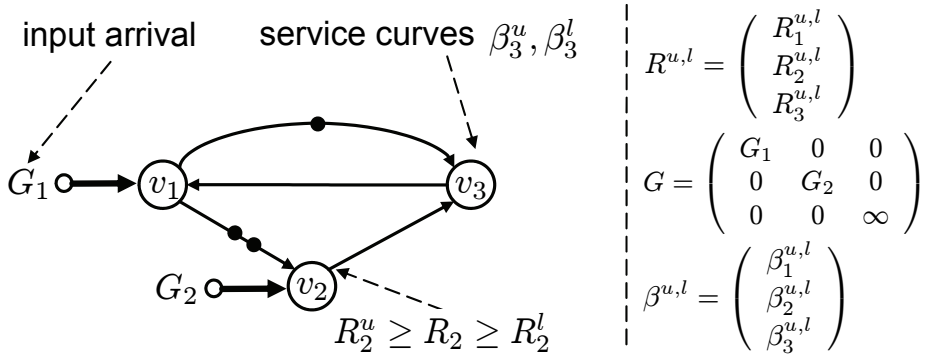

Fig. 4.7: Marked graph with input arrival functions G, service curves β^u, β^l associated to its nodes and the resulting traces characterized by arrival functions R and their bounds R^u, R^l

Token sources can enter the AND elements of Greedy Marked Graph Processes like any other channel, see Fig. 4.5. Therefore, we can start from the elementary system equation (4.11) and integrate external system inputs. Let us define the system input matrix G with elements (G_{ij}), where $G_{ii} = G_i(t)$ if there is an input at node v_i with arrival function $G_i(t)$, otherwise $G_{ii} = G_i(t) = \infty$ for all t if there is no input at node v_i, and all other non-diagonal matrix elements are set to 0, see Fig. 4.7. Then we can obtain the following equation:

$$R_i = [(R_j + I^{m_{ji}}) \wedge (R_k + I^{m_{ki}}) \wedge G_i] \otimes \beta_i \ .$$

Using this information in the fixed-point equation discussed in the previous section, we replace (4.19) and (4.20) in Theorem 4.18 with

$$R^l = (S^l)^* \otimes G \otimes \beta^l \ , \tag{4.24}$$

$$R^u = (S^u)^* \otimes G \otimes \beta^u \ . \tag{4.25}$$

4.4.6 Transfer Functions in Marked Graphs

As a last preparatory step for embedding marked graphs into any compositional performance analysis framework, we need to determine the transfer functions of a marked graph: How does a single input stream G_s at source node v_s influence the output data stream R_d at some destination node v_d?

Since this information is already contained in (4.24) and (4.25), we only have to rewrite the equations such that we (a) only evaluate the bounds on the output arrival functions $R_d^{u,l}$ at the destination v_d, and (b) have only one explicit external input, namely G_s at the source node v_s. In order to simplify the notation, we suppose that the graph has only a single input

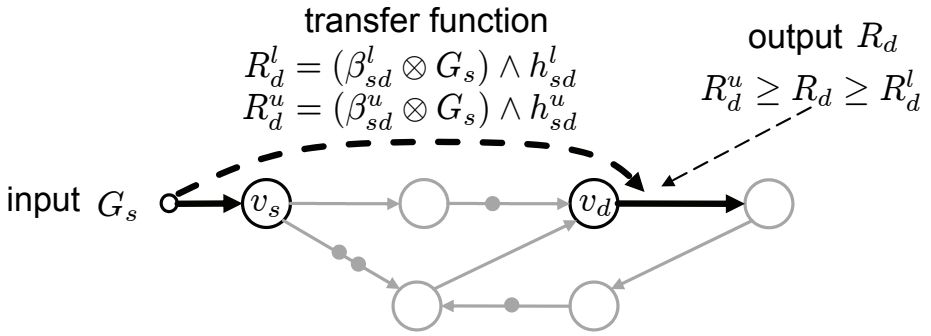

Fig. 4.8: Visualization of transfer functions of marked graphs

at node v_s, an extension to the general case is straightforward. As a result of the whole exercise we find the following expressions:

$$R_d^l = (\beta_{sd}^l \otimes G_s) \wedge h_{sd}^l \,, \tag{4.26}$$

where $\beta_{sd}^l = (S^l)_{ds}^* \otimes \beta_s^l$ and $h_{sd}^l = \bigwedge_{j \neq s}((S^l)_{dj}^* \otimes \beta_j^l)$, and similarly for the upper bound:

$$R_d^u = (\beta_{sd}^u \otimes G_s) \wedge h_{sd}^u \,, \tag{4.27}$$

where $\beta_{sd}^u = (S^u)_{ds}^* \otimes \beta_s^u$ and $h_{sd}^u = \bigwedge_{j \neq s}((S^u)_{dj}^* \otimes \beta_j^u)$.

Here, we note that $\beta_{sd}^{u,l}$ denote the cumulative service curves for the path from source node v_s to destination v_d, and $h_{sd}^{u,l}$ denote 'offset' terms. The latter are functions that are independent from the input arrival, i.e. they represent the constant part in the transfer function which is the response of the marked graph if the input stream does not contain any tokens. Note also, that (4.26) and (4.27) are scalar functions and not matrices or vectors anymore. Figure 4.8 visualizes the concept of a transfer function for a path in a marked graph.

4.5 Performance Analysis

In this section, we will do the last abstraction on marked graphs. After replacing the service functions $C_i(t)$ that represent the availability of a resource at node v_i in the time domain by their corresponding service curves $\beta_i^u(\Delta)$ and $\beta_i^l(\Delta)$ in the time interval domain, we now introduce a similar abstraction for the arrival functions $R_i(t)$ and $G_i(t)$. This last abstraction is necessary for several reasons.

Now, the whole analysis of a marked graph can be done in the time interval domain. This way, we can not only embed the analysis in the

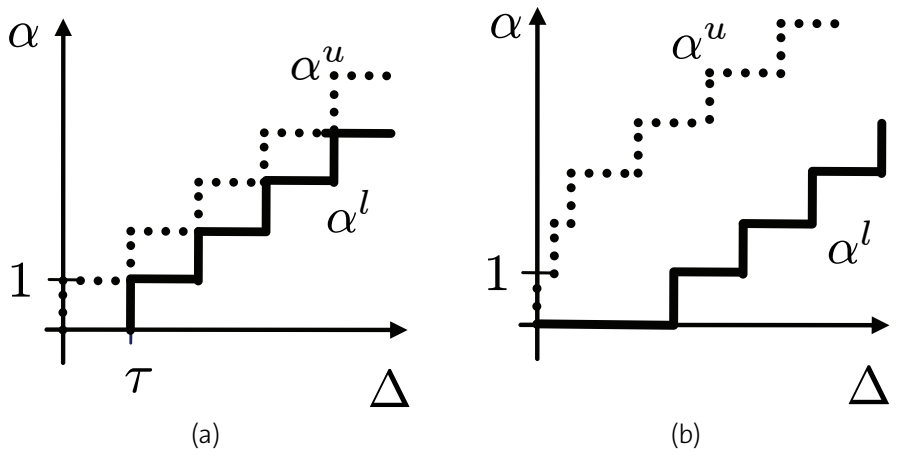

Fig. 4.9: Two examples of arrival curves: **(a)** arrival curves for a periodic stream, **(b)** arrival curves for a periodic stream with jitter and limited bursts

Modular Performance Analysis framework, see for example [WTVL06], but also relate it to classical real-time analysis that expects stream characterizations like periodicity, jitter, and burst size, see [RJE03].

We will be able to determine performance bounds on end-to-end delays, necessary buffer spaces, and the remaining service, i.e. after a given resource has been used for executing a certain marked graph node. The last one enables composability in terms of resources which makes possible the analysis of various resource sharing strategies such as FP and TDMA.

4.5.1 Arrival Curves

In contrast to arrival functions $R(t)$ that count the number of tokens that occurred in $[0, t)$, arrival curves determine upper and lower bounds on the number of tokens in any *time interval* of size Δ, for examples see Fig. 4.9.

Definition 4.20: (Arrival Curves) *Upper and lower arrival curves α^u, α^l map positive time intervals $\Delta \in \mathbb{R}^{\geq 0}$ to the maximal and minimal number of tokens in any time interval of length Δ. They satisfy $\alpha^u(0) = \alpha^l(0) = 0$ and*

$$\alpha^l(\Delta) \leq R(t + \Delta) - R(t) \leq \alpha^u(\Delta) \quad \forall t \geq 0, \Delta > 0 \; .$$

In order to simplify the following discussions, we will use the notation of the min-plus and max-plus deconvolution operators, $\oslash$ and $\overline{\oslash}$, respectively, as defined in Appendix A.

Now we can make use of Definition 4.20 and obtain the tightest arrival curves (i.e. the least upper and greatest lower curves) of an internal stream $R(t)$ and an input stream $G(t)$. Here, we use α and γ to denote arrival

curves related to internal streams (arrival functions R) and external input streams (arrival functions G):

$$\alpha^u = R \oslash R \,, \quad \alpha^l = R \overline{\oslash} R \,, \tag{4.28}$$

$$\gamma^u = G \oslash G \,, \quad \gamma^l = G \overline{\oslash} G \,. \tag{4.29}$$

4.5.2 Bounds on Buffer Size and End-to-End Delay

Let us now determine an upper bound B_{sd} on the difference between the number of tokens that arrived at the input of some source node v_s and left at some destination node v_d at any time. For example, if we set $v_s = v_d$, then B_{ss} is an upper bound on the number of tokens that are stored in front of the marked graph (of node v_s), i.e. at its input queue. In other words, one can then guarantee that an input queue of size B_{ss} would be sufficient to store all necessary tokens during any execution of the marked graph.

The above definition of B_{sd} directly yields:

$$G_s(t) - R_d(t) \leq G_s(t) - ((\beta^l_{sd} \otimes G_s) \wedge h^l_{sd})(t) = B_{sd} \,.$$

Using the transfer functions from (4.26) and (4.27), and the arrival curve corresponding to the input as computed with (4.29), we obtain:

$$\begin{aligned}
&G_s(t) - ((\beta^l_{sd} \otimes G_s) \wedge h^l_{sd})(t) = \\
=&G_s(t) + \max\{-h^l_{sd}(t), \sup_{0\leq\lambda\leq t}(-G_s(t-\lambda) - \beta^l_{sd}(\lambda))\} \\
=&\max\{G_s(t) - h^l_{sd}(t), \sup_{0\leq\lambda\leq t}(G_s(t) - G_s(t-\lambda) - \beta^l_{sd}(\lambda))\} \\
\leq&\max\{\gamma^u_s(t) - h^l_{sd}(t), \sup_{0\leq\lambda\leq t}(\gamma^u_s(\lambda) - \beta^l_{sd}(\lambda))\} \,.
\end{aligned}$$

As a result, we find:

$$B_{sd} \leq \max\left\{\sup_{\lambda\geq 0}\{\gamma^u_s(\lambda) - h^l_{sd}(\lambda)\}, \sup_{\lambda\geq 0}\{\gamma^u_s(\lambda) - \beta^l_{sd}(\lambda)\}\right\} \,. \tag{4.30}$$

In other words, the maximal backlog as defined above can be determined as the maximum of the maximal vertical distances between the functions γ^u_s (the upper arrival curve corresponding to the input stream) and h^l_{sd} as well as between γ^u_s and β^l_{sd}.

Let us now determine a bound D_{sd} on the end-to-end delay of tokens, i.e. the maximal time any token needs from a system input at the source node v_s to the output of a destination node v_d. In order to faithfully determine such a bound on the end-to-end delay we suppose that the system does not produce an output if the input stream is empty, i.e. $G_s(t) = 0$. Therefore, we require that $\beta^l_{sd}(0) = 0$.

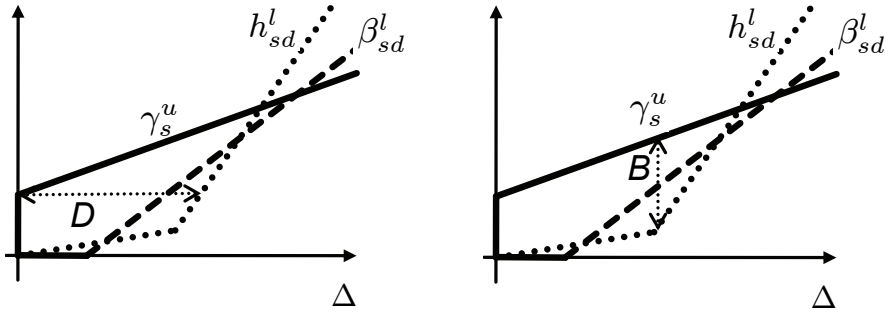

Fig. 4.10: Visualization of: **(a)** delay, and **(b)** backlog bounds in a marked graph

In order to simplify the notation let us first define the maximal horizontal distance of two functions A and B as follows:

$$h(A,B) = \sup_{t\geq 0}\{\inf\{\tau \geq 0 \,:\, B(t+\tau) \geq A(t)\}\} \,.$$

Then an upper bound on the end-to-end delay of any token between an input G_s at source node v_s to a destination node v_d is given by:

$$D_{sd} = h(G_s, R_d) = \sup_{t\geq 0}\{\inf\{\tau \geq 0 \,:\, R_d(t+\tau) \geq G_s(t)\}\} \,.$$

Using similar arguments as in the case of the necessary buffer space, we obtain that:

$$\begin{aligned}
&G_s(t) - R_d(t+\tau) = \\
&= \max\{G_s(t) - h^l_{sd}(t+\tau), \sup_{0\leq\lambda\leq t+\tau}(G_s(t) - G_s(\lambda) - \beta^l_{sd}(t+\tau-\lambda))\} \\
&\leq \max\{0, G_s(t) - h^l_{sd}(t+\tau), \sup_{0\leq\lambda\leq t}(G_s(t) - G_s(\lambda) - \beta^l_{sd}(t+\tau-\lambda))\} \\
&\leq \max\{0, \gamma^u_s(t) - h^l_{sd}(t+\tau), \sup_{0\leq\lambda\leq t}(\gamma^u_s(t-\lambda) - \beta^l_{sd}(t-\lambda+\tau))\} \\
&\leq \max\{0, \sup_{\lambda\geq 0}\{\gamma^u_s(\lambda) - h^l_{sd}(\lambda+\tau)\}, \sup_{\lambda\geq 0}(\gamma^u_s(\lambda) - \beta^l_{sd}(\lambda+\tau))\} \,.
\end{aligned}$$

As a result we get the following upper bound:

$$D_{sd} \leq \max\{h(\gamma^u_s, \beta^l_{sd}), h(\gamma^u_s, h^l_{sd})\} \,. \tag{4.31}$$

In other words, the maximal delay as defined above can be determined as the maximum of the maximal horizontal distances between the functions γ^u_s and h^l_{sd} as well as between γ^u_s and β^l_{sd}. The interpretations of the delay and buffer bounds in marked graphs are visualized in Fig. 4.10.

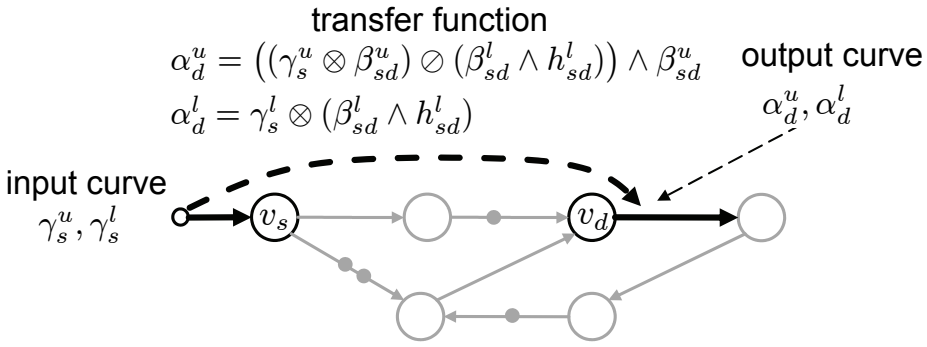

Fig. 4.11: Transfer function of a marked graph in time interval domain

4.5.3 Output Arrival Curves

In this section, we will describe a method that allows to compute bounds on the token streams at any node in a given marked graph. In comparison to the results in Theorem 4.18, these bounds are now given in terms of arrival curves, i.e. not in the time domain but in the time interval domain as can be seen when comparing Fig. 4.8 with Fig. 4.11. Again, it is necessary to abstract from the concrete time domain because it leads to composability in terms of resources and event streams.

The derivation starts from the transfer functions developed in (4.26) and (4.27). In order to apply the relations known from Real-Time Calculus, see [TCGK02], we first approximate the transfer functions from input v_s to node v_d as follows:

$$R_d^l = (\beta_{sd}^l \otimes G_s) \wedge h_{sd}^l \geq (\beta_{sd}^l \otimes G_s) \wedge (h_{sd}^l \otimes G_s) \geq (\beta_{sd}^l \wedge h_{sd}^l) \otimes G_s \ , \tag{4.32}$$

$$R_d^u = (\beta_{sd}^u \otimes G_s) \wedge h_{sd}^u \leq \beta_{sd}^u \otimes G_s \ . \tag{4.33}$$

As a result, we find that upper and lower bounds on the output stream at node v_d in the time domain $R_d^{u,l}$ can be determined by convolving the input stream function G_s with a certain service curve. This fact enables to directly use the results shown in [TCGK02] to compute the corresponding output arrival curves. The results can be summarized in the following theorem.

Theorem 4.21: (Transfer Functions in Marked Graphs in Time Interval Domain) *Given a marked graph (V, E, M), and vector service curves β^u and β^l associated to its nodes according to Theorem 4.18. Suppose that the network has a single input stream at node v_s with arrival curves γ_s^u and γ_s^l. Then we can determine upper and lower arrival curves, α_s^u and α_s^l, associated to any node v_d using the following expressions:*

$$\alpha_d^u = \left((\gamma_s^u \otimes \beta_{sd}^u) \oslash (\beta_{sd}^l \wedge h_{sd}^l)\right) \wedge \beta_{sd}^u \ , \tag{4.34}$$

$$\alpha_d^l = \gamma_s^l \otimes (\beta_{sd}^l \wedge h_{sd}^l) \ , \tag{4.35}$$

where we use β^l_{sd}, β^u_{sd}, and h^l_{sd} as defined for (4.26) and (4.27).

Proof. Results follow directly from (4.28), (4.29), (4.32), and (4.33).

■

4.5.4 Remaining Service

In order to enable compositionality in terms of resources, we need to determine the remaining service curves β'^l_d and β'^u_d at any node v_d, see Fig. 4.5. This enables us to use the remaining service curves as inputs to processes of other marked graphs, and thereby, represent fixed priority scheduling where the first process, i.e. the one that gets the initial service curves, β^l_d and β^u_d, has a higher priority in comparison to the process that just gets the remaining service curves, β'^l_d and β'^u_d.

To this end, we start from the balancing equation (4.4) of a Greedy Processing Component according to Definition 4.7, i.e. the remaining service equals the available service reduced by the produced output, i.e. $C'_d(t) = C_d(t) - R'_d(t)$. Therefore, we can obtain the following expression:

$$C'_d(t+\Delta) - C'_d(t) = [C_d(t+\Delta) - C_d(t)] - [R'_d(t+\Delta) - R'_d(t)] \,,$$

which is bounded by:

$$\beta'^l_d(\Delta) \geq \beta^l_d(\Delta) - \alpha^u_d(\Delta) \,, \qquad \beta'^u_d(\Delta) \leq \beta^u_d(\Delta) - \alpha^l_d(\Delta) \,.$$

Using the fact that the remaining service curves are monotone functions, we can tighten the bounds as follows:

$$\beta'^l_d(\Delta) = \sup_{0 \leq \lambda \leq \Delta} \{\beta^l_d(\lambda) - \alpha^u_d(\lambda)\} \,, \tag{4.36}$$

$$\beta'^u_d(\Delta) = \max\{0, \inf_{\Delta \leq \lambda} \{\beta^u_d(\lambda) - \alpha^l_d(\lambda)\}\} \,. \tag{4.37}$$

In the last two subsections, we have determined the output arrival curves at any process and the corresponding remaining service curves of any process in a marked graph. These representations can then be used in order to compose the marked graph model with other parts of the application. For example, the output of a marked graph characterized by its arrival curves can be linked to the input of some other application that may be given as any MPA or SymTA/S model. One can also link the remaining service to another performance model and analyze a fixed priority setting this way, see also [CKT03b, WTVL06].

4.6 Experimental Results

Here we show the feasibility of the proposed analysis framework by first comparing its accuracy to another recent analysis framework when both frameworks are applied to a simple finite buffer system. Secondly, we analyze a more complex system from the area of software defined radio, and compare the analysis results to simulation measurements.

4.6.1 Comparison

In this section we compare the performance analysis results computed with the method proposed in this chapter with results computed with the methods proposed in [BJLL06] and [BPC09]. These two methods are very similar in their approach and therefore we have implemented only the more recent one described in [BPC09]. It is also based on Real-Time Calculus but it is limited to simple cyclic models of finite buffer systems. For simplicity in this section, we will refer to our method as MG (for marked graph) and the method proposed in [BPC09] as FB (for finite buffer).

System. The system used for evaluation is shown in Fig. 4.12(a). It is a simple chain of three tasks $T1, T2, T3$ which processes a bursty input event stream which timing characteristics are described with period 4 ms, jitter 20 ms, and minimum interarrival distance between two events 1 ms. All tasks have constant execution times of 1 ms. They are mapped on an MpSoC with three processing elements $PE1, PE2$, and $PE3$. $PE1$ exhibits complex behavior due to being shared with other tasks, it may not be available to task $T1$ for 2 ms, then it may provide service of maximum 20 events/ms which eventually slows down to a long-term processing rate of 1 event/ms. Similarly, $PE2$ may not be available for 2 ms, has a maximum speed of 2 events/ms, and a long-term rate of 1 event/ms. $PE3$ may not be available for 1 ms, and has a constant rate of 0.5 events/ms. For simplicity, the communication hardware is not shown here and it is not modeled.

Each task is activated by events that arrive in a FIFO buffer mapped to the same processing element as the task. For modeling purposes, tasks $T1$ and $T2$ have buffers with unlimited capacity. Task $T3$ has a buffer with a finite size $B = 1$. The semantics of the buffer are blocking-write which means that task $T2$ needs to block if the buffer at $T3$ is full. When $T2$ blocks, the service provided by $PE2$ is available to be used by other lower priority tasks mapped on $PE2$.

Model. The system is modeled with a simple marked graph that has a single cycle with one initial token, see Fig. 4.12(b). The abstract model of greedy marked graph processes that is used for performance evaluation

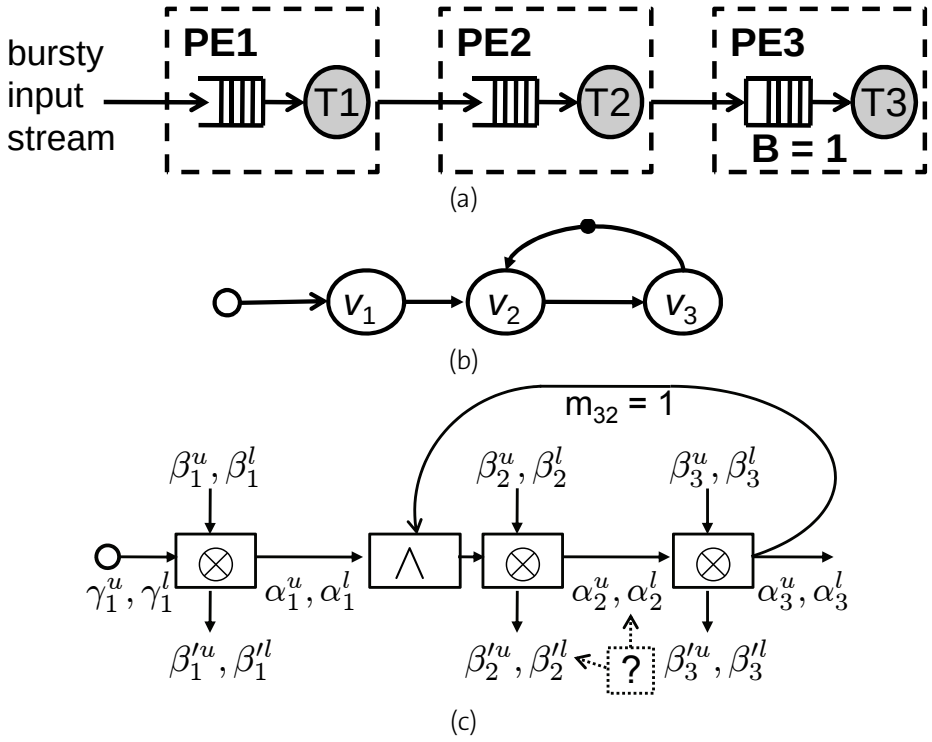

Fig. 4.12: System with one blocking-write buffer at task $T3$ shown as: **(a)** abstract system model, **(b)** marked graph model, and **(c)** marked graph model with greedy processes

of the system with method MG is shown in Fig. 4.12(c). It is annotated with the arrival curves of the input: γ_1^u, γ_1^l, the output arrival curves of the three tasks: $\alpha_i^u, \alpha_i^l, i = 1, 2, 3$, the service curves modeling the service provided by the processing elements to the tasks: $\beta_i^u, \beta_i^l, i = 1, 2, 3$, and the remaining service curves that characterize the unused service from the tasks: $\beta_i'^u, \beta_i'^l, i = 1, 2, 3$.

Scenario. We compare methods MG and FB in terms of tightness (accuracy) of the computed performance metrics for the system in Fig. 4.12(a). More specifically, we compare the bounds on the output of task $T2$: α_2^u, α_2^l computed with the two methods, and the bounds on the remaining service of task $T2$: $\beta_2'^u, \beta_2'^l$ again for both methods. The parameters of interest are marked with a question mark '?' in Fig. 4.12(c). We have chosen these parameters because they are essential for computing bounds on other performance metrics such as end-to-end delays and buffer sizes. Any inaccuracy in computing the chosen parameters will have an influence on all other computed metrics for the system.

Results. Bounds for the output event stream of $T2$ computed with methods FB and MG are shown in Fig. 4.13. Note that method FB does

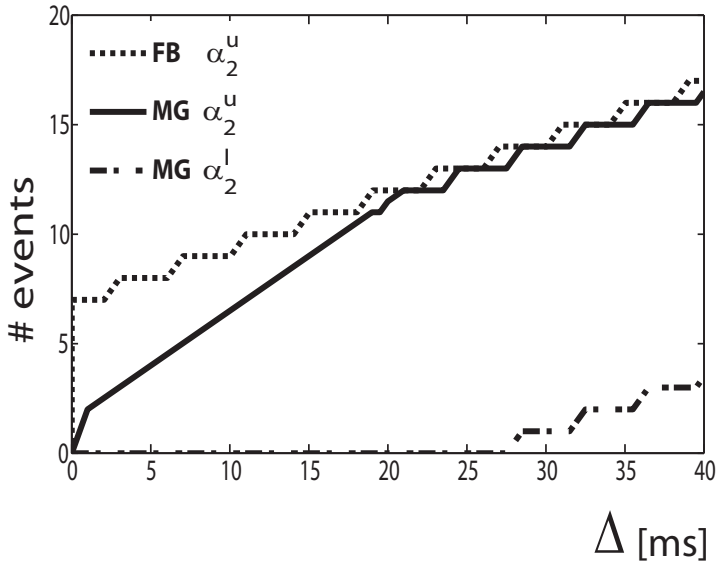

Fig. 4.13: Comparison of methods FB and MG for the output event stream. Note that method FB does not compute lower bounds on the output event streams

not compute the lower bound α_2^l. For the upper bound α_2^u, method MG is tighter, and it accurately shows the fact that there cannot be a burst of events coming out of task $T2$ since the input buffer of $T3$ is of finite size. Even for the long term rate, method FB shows some error.

Bounds for the remaining service of $T2$ are shown in Fig. 4.14. Note, that method FB does not compute an upper bound on the service $\beta_2'^u$. For the lower bound $\beta_2'^l$, method MG is again tighter. This is due to the fact that method FB computes a pessimistic bound on the event output of the task which is then used for computation of the remaining service.

The tightness of the results computed with method MG can be observed even for simple systems such as the one used here. We expect that the differences in results will be more visible for more complex systems. MG is a more general method than method FB since it can analyze not only systems with finite buffers but any system that can be modeled with a marked graph. And more importantly, this gain in generality does not lead to inaccuracies in the computed results.

4.6.2 Validation

In this section, we validate our approach with a more complex scenario and compare the analysis results to simulation measurements.

System and models. We use an application from the area of software defined radio. It is adapted from [MVB07]. The Wireless LAN (WLAN) and the Time Division Synchronous Code Division Multiple Access (TD-

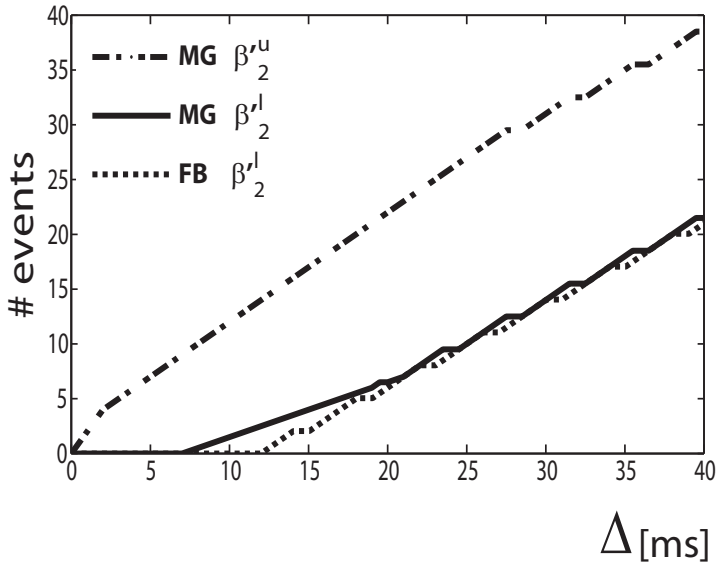

Fig. 4.14: Comparison of methods FB and MG for the remaining service. Note that method FB does not compute upper bounds on the remaining services

SCDMA) applications run in parallel. Both of them are modeled as marked graphs as depicted in Fig. 4.15. In contrast to [MVB07], we will use an idealized scenario. The underlying multiprocessor architecture consists of 5 independent cores where communication time is supposed to be negligible.

It is assumed, that processor 5 provides a TDMA schedule which partitions the period into two equal time slices, named 5.1 and 5.2. Processors 1–4 have speeds of 100 million cycles/sec, processor 5 provides 200 million cycles/sec. Table 4.1 lists the mapping of marked graph nodes

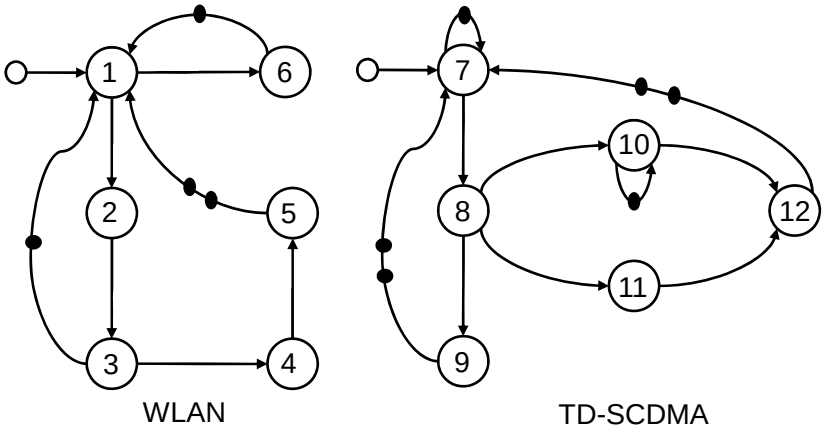

Fig. 4.15: Marked graphs that model WLAN and TD-SCDMA applications

Tab. 4.1: Mapping and worst-case execution demands in cycles for each of the processes from Fig. 4.15

node	1	2	3	4	5	6
WCED [cycles]	$2k$	$0.31k$	$0.33k$	$0.42k$	$4k$	$2k$
core	1	2	4	3	5.1	5.2
node	7	8	9	10	11	12
WCED [cycles]	$50k$	$12.5k$	$20k$	$3.3k$	$0.25k$	$50k$
core	1	2	5.1	3	4	5.2

Tab. 4.2: Maximum end-to-end delays from source to the different outputs observed in simulation and compared to analytical results

output	1	2	3	4	5	6
simulation [ms]	0.02	0.023	0.026	0.031	0.12	0.11
analysis [ms]	0.02	0.023	0.027	0.031	0.151	0.13
output	7	8	9	10	11	12
simulation [ms]	0.56	0.688	1.025	0.721	0.691	1.28
analysis [ms]	0.56	0.688	1.048	0.726	0.692	1.316

to processors, and the number of cycles each of the nodes needs on the respective processor, i.e. their worst-case execution demands. The TDMA-scheduler in processor 5 is assumed to have a period of 0.2 ms and equal slot lengths for slices 5.1 and 5.2, i.e. 0.1 ms each. We further assume, that the inputs to the two applications are periodic with periods equal to 0.2 ms and 0.7 ms, for the WLAN and TD-SDMA applications, respectively.

Let us suppose that we use fixed priority scheduling where all nodes of the WLAN marked graph have higher priority than those of the TD-SCDMA marked graph.

Experimental setup. Simulation models of the two applications have been implemented in the Real-Time Simulation (RTS) Toolbox [TS09]. It is a framework for discrete-event simulation which uses the component structure of MPA [CKT03b, WTVL06] however, instead of using abstracted event and resource models, it uses traces which are produced randomly following the specifications of the processing cores and the input streams. The processes are simulated assuming their worst-case execution demands. The analysis computations have been performed with the Real-Time Calculus (RTC) Toolbox [WT06c].

Scenario. We compare results from analysis computed with inequality (4.31) and simulations for the maximum token delays from the input

node to the outputs of all nodes in the graph for both applications. The simulation has been performed with several traces and from all of them the maximum observed end-to-end delays have been selected.

Results. The results are summarized in Tab. 4.2. They show the tightness of the analysis and the feasibility of the method for the performance analysis of complex data processing applications with cyclic dataflow.

4.7 Discussion

The chapter presents a new compositional performance analysis framework for distributed implementations of cyclic dataflow graphs, in particular marked graphs. It is based on Modular Performance Analysis and Real-Time Calculus. It substantially generalizes previous analysis approaches in that general non-deterministic input event streams can be modeled by means of arrival curves, and general non-deterministic resource interactions can be modeled by means of service curves. This way, it is possible to model implementations with finite buffer sizes, model dynamic scheduling where the processes of different marked graphs are scheduled according to a fixed priority scheme, and take into account other scheduling disciplines like TDMA.

Unlike Chapters 2 and 3 which deal with interface-based design methods for component models of real-time systems free of directed cycles, this chapter focuses on an analysis method for cyclic component models. Developing an interface-based design method for cyclic component models is a topic for future research which can build on the results presented in this chapter. As a component can be considered not only an individual process in a marked graph, but also the whole marked graph. Such a design framework would allow us to specify interfaces on the buffer sizes in finite-buffer systems, and would partly avoid the need for exhaustive search for minimum buffer sizes in throughput-constrained systems.

Part II

Analysis of Adaptive Embedded Systems

5

Analysis of Adaptive Applications

Many application domains require adaptive real-time embedded systems that can change their functionality over time. Performance analysis methods need to analyze these systems not only in the individual modes where functionality does not change, but also during the transitions between operating modes. Such mode change analysis is important because during transitions, the system may execute functions belonging to both of the operating modes which may cause the system to become temporary overloaded. Known approaches that address the problem of timing analysis over mode changes are mostly restricted to fixed priority scheduling policies. In addition, most of them are also limited to simple periodic or sporadic event stream models and therefore, they can not faithfully abstract the bursty timing behavior which can be observed in complex embedded systems. This chapter proposes a new method for the design and analysis of adaptive multi-mode systems that supports any event stream model. The analysis is embedded in the well-established Modular Performance Analysis framework based on Real-Time Calculus [CKT03b].

Chapters 2 and 3 deal with interface-based design frameworks for real-time systems. They specify interfaces for system components that can be used during design-time to answer essential questions like what is the minimum resource required by a component in order to meet a certain deadline constraint, what is the maximum input rate supported by the real-time system, etc. In particular, interfaces specify bounds within which system parameters, such as input event stream rates, processing element speeds, communication buses speeds, can be changed. If the actual system parameter values fall within the bounds given by an interface, then the interface-based design theory guarantees that the system will meet all real-time constraints such as maximum memory usage, end-to-end delays, throughput, etc. However, interfaces do not specify how fast and under what conditions system parameters may

change, e.g. how often the environment can switch from one input rate to another. The problem of analyzing such dynamics and transient effects is the topic of this chapter. It focuses on changes in the parameters of an application such as task activation rates and task execution times. The presented results are valid for fixed priority scheduled sets of tasks that run on uniprocessor systems. As it will be shown, run-time changes in task parameters can be done only under certain conditions, otherwise the system may violate its real-time constraints. Unlike Chapter 4 which deals with performance analysis of cyclic dataflow systems, this chapter considers only cycle-free systems.

5.1 Introduction

Several application domains ask for real-time embedded systems that can adapt their behavior at run-time by changing their operating mode. Examples of multi-mode systems are adaptive control systems in the automotive domain or new mobile phone applications such as software defined radio receivers. A mode change in an embedded application can be requested for several reasons. For example the system might need to switch to an emergency state, to adapt its behavior to changed conditions in the environment, or to change its resource usage.

Another example is the use of scheduling servers within an operating system. They assign a periodic computing service to each application and therefore, lead to a largely reduced timing interference between applications. But the computing bandwidth available to a specific application needs to be adapted to its needs and new servers appear with new applications. Each of these adaptations leads to a mode change.

We assume that a mode change can involve changes in the set of executed tasks, changes in the parameters of tasks (e.g. execution time, deadline) or changes in the activation pattern of tasks. In such adaptive real-time systems, all deadlines must be provably met not only in the individual operating modes, but also during the transitions between modes. It is therefore essential to provide designers of multi-mode real-time systems with appropriate instruments for the verification of timing constraints across mode changes.

While in some domains a change of the operating mode can be performed by simply stopping the execution of all tasks in the system and restarting it with a new configuration, in many applications this is not feasible and the mode changes have to be carried out dynamically at run-time. For instance in many dependable systems there are tasks which cannot be stopped and must reliably execute also during changes in the configuration of the system.

In general, a sudden mode change at run-time can have severe and unexpected impacts on the timing behavior of a system. For instance, if the execution of a task is triggered by an event stream, replacing the event stream instantaneously with a less demanding one (e.g. one with a larger period) can nevertheless harm the system, because bursts of events from both event streams appearing at the switching time can lead to a transient overload of the system and missed deadlines.

In summary the contributions of this chapter can be stated as follows:

- We present a method for timing analysis of uniprocessor multi-mode systems with fixed priority (FP) scheduling of tasks that supports any task activation pattern. It is the first method that is based on Modular Performance Analysis with Real-Time Calculus [CKT03b, TCN00] and can consider general event and resource stream models.
- We show how the method can be applied to transform a non-schedulable mode change into a schedulable one using an offset.
- We show the applicability of the presented method by analyzing a case study.

Section 5.2 briefly discusses some of the related work. Section 5.3 presents a video-processing application which will be used to illustrate some of the problems that occur in multi-mode real-time systems. Section 5.4 presents a short introduction to the abstractions used in Modular Performance Analysis with Real-Time Calculus [CKT03b] that underlie the mode change analysis presented in this chapter. Section 5.5 presents the mode change model that we consider and which system parameters are allowed to change. Section 5.6 presents the mode change analysis for fixed priority scheduled sets of tasks with and without offsets. Section 5.7 applies the analysis framework to the example system introduced in Section 5.3. Finally, Section 5.8 concludes this chapter with a brief summary and a discussion.

5.2 Related Work

The problem of timing analysis across mode changes has been addressed previously, see [RC04]. An analysis approach for mode changes on uniprocessor systems with rate-monotonic scheduling is introduced in [SRLR89]. The analysis approach is improved and extended to deadline-monotonic scheduling in [TBW92]. The model is augmented with transition offsets in [PB98], which permits to avoid overload situations. However, a way to calculate such offsets is not provided. A slightly

different mode change protocol is introduced in [RC04], together with an algorithm for offset calculation. All these analysis methods are limited to strictly periodic task activations. This restriction was recently overcome in the SymTA/S approach [HE07], where the authors consider mode changes under a more general task activation scheme. In their model each event stream is described by three parameters: period, jitter and minimum interarrival distance between events. Although this model captures considerably more complex activation patterns, it still describes only a limited set of event streams.

Mode change analysis for dynamic priority systems with earliest deadline first (EDF) scheduling has been proposed in [And08]. However, it is limited only to tasks activated by sporadic event streams.

This chapter extends all previous results by considering general event stream models modeled with arrival curves and resources modeled with service curves, both of which are known from Network Calculus [LBT01]. The analysis can consider offsets between tasks which can be efficiently calculated using a binary search strategy. The results presented here are applied to fixed priority scheduled systems but can be extended to EDF scheduling as shown in [SPT09].

5.3 Illustrative Example

For illustrative purposes, we present a multimedia system that undergoes a mode change. It will serve as a case study throughout the chapter, and will help us to visualize the presented theoretical results. Figure 5.1 shows the architecture of a digital set-top box implementing a picture-in-picture (PiP) application where two concurrent MPEG-2 video streams

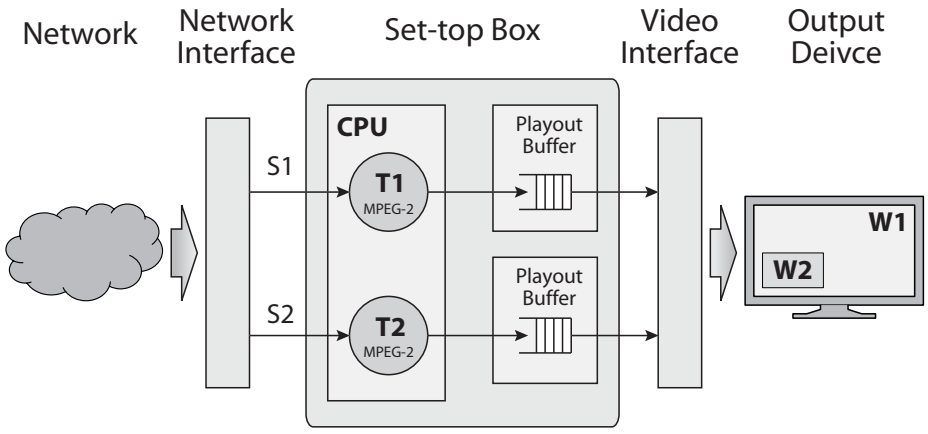

Fig. 5.1: System architecture

are being decoded on a single CPU and displayed on the same output device. The streams correspond to a main window W_1 and a small window W_2 in the output device, and they are denoted as S_1 and S_2, respectively. The CPU executes two tasks T_1 and T_2 which perform the MPEG-2 decoding for S_1 and S_2, respectively. The inputs to the tasks are compressed bitstreams and their outputs are decoded macroblocks, which are written into playout buffers. The video interface reads from the playout buffers at a constant rate, and sends the data to the display.

Consider now that the video displayed on W_1 changes from video stream S_1 (mode I) to a video stream $S_{1\,\text{new}}$ with a lower workload (mode II). Such an instantaneous mode change is likely to result in a transient overload of the system and missed deadlines, which in this scenario translates to distorted playback. A common way to reduce the workload during mode changes is to delay the start of mode II by an offset [PB98]. In this case, workload from mode I is accepted up to the time of the mode change request t_{MCR}, while workload from mode II is not accepted before $t_{\text{MCR}} + \delta$, where δ is the length of the offset. The designer of the set-top box needs to find an offset of sufficient length in order to avoid distorted images during stream changes. At the same time, the offset should be kept as short as possible. Adding an offset delays the switch to the new video stream however, the disruption is completely predictable at design time.

Choosing the offset for a mode change is not trivial at all. To illustrate this, consider that the two tasks T_1 and T_2 are scheduled according to a preemptive FP scheduling policy, with T_1 having higher priority than T_2. For the sake of simplicity, assume that T_1 and T_2 are triggered by periodic event streams with jitter and have constant execution times. In Section 5.7 we will analyze the set-top box mode change scenario for realistic MPEG-2 video streams. For now, consider that task T_1 in mode I has a constant execution time of 2 ms, a relative deadline of 11 ms, and it is triggered by stream S_1 described with period of 11 ms and jitter of 10 ms. In mode II, the execution time becomes 3 ms, the relative deadline is 18 ms, and the activation stream $S_{1\,\text{new}}$ is described by period of 18 ms and jitter of 10 ms. For task T_2, the parameters in mode I and mode II are equal, and the task continues to run uninterruptedly during the mode change. Its execution time is 30 ms, the relative deadline is 41 ms, and it is triggered by stream S_2 that is described by period of 41 ms and jitter of 5 ms.

A reasonable guess for the length of the offset at the mode change is $\delta = 21$ ms, which is the maximum distance between the arrival of two events in stream S_1. Since the workload for $S_{1\,\text{new}}$ is slightly smaller ($3/18 < 2/11$), one might assume that this offset is sufficient for meeting all deadlines. However, the trace depicted in Fig. 5.2 shows that with $\delta = 21$ ms, an activation of task T_2 can miss a deadline due to the video

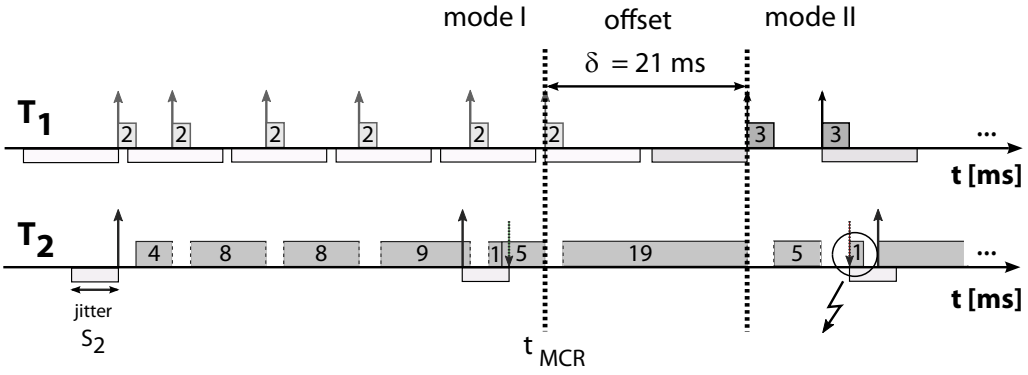

Fig. 5.2: Missed deadline due to transient overload during mode change

change in the higher priority stream. In other words, switching between two video streams in W_1 can cause an unpredictable disruption in the video playback of W_2. The example shows that formal methods are desirable for the design and analysis of multi-mode real-time systems.

5.4 System Abstractions

In this section we shortly describe the framework of Modular Performance Analysis with Real-Time Calculus (in short called MPA), on top of which we will develop the analysis of mode changes. MPA is a compositional framework for system-level performance analysis of distributed real-time systems [CKT03b, TCN00]. It has its roots in Network Calculus [Cru91a, LBT01]. It analyzes the flow of event streams through a network of processing and communication resources in order to compute worst-case backlogs, end-to-end delays, and throughput.

5.4.1 A General Event Stream Model

Event streams are abstracted by a tuple $\alpha(\Delta) = [\alpha^u(\Delta), \alpha^l(\Delta)]$ of upper and lower arrival curves which provide an upper and a lower bound on the number of events in *any* time interval of length Δ. If $R[s,t)$ denotes the number of events that arrive in the time interval $[s,t)$, then the following inequality holds:

$$\alpha^l(t-s) \leq R[s,t) \leq \alpha^u(t-s) \qquad \forall s < t \ ,$$

where $\alpha^u(\Delta) = \alpha^l(\Delta) = 0$ for $\Delta \leq 0$. Arrival curves substantially generalize conventional event stream models such as sporadic, periodic, or periodic with jitter. Note that often the domain of arrival curves are workload units. Event-based arrival curves can be converted to workload-based

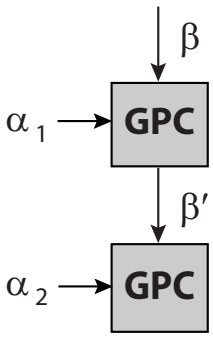

Fig. 5.3: Two abstract components GPC modeling two tasks processing two streams with FP scheduling

arrival curves by scaling with the best-case/worst-case execution demand of events. We will use the workload-based interpretation in the rest of the chapter. For more general characterizations of the workload refer to Chapter 3 Section 3.8.1.

5.4.2 A General Resource Model

The availability of processing or communication resources is described by a tuple $\beta(\Delta) = [\beta^u(\Delta), \beta^l(\Delta)]$ of upper and lower service curves which provide an upper and a lower bound on the available service in *any* time interval of length Δ. The service is expressed in an appropriate workload unit compatible to that of the arrival curve, like number of cycles for computing resources or bits for communication resources. If $C[s,t)$ denotes the amount of workload units available from a resource in the time interval $[s,t)$, then the following inequality holds:

$$\beta^l(t-s) \leq C[s,t) \leq \beta^u(t-s) \qquad \forall s < t \ .$$

5.4.3 Processing Model and Analysis

In real-time systems, event streams are typically processed by a sequence of HW/SW components. In the framework of MPA such processing or communication components are modeled by abstract performance components that act as curve transformers in the domain of arrival and service curves, where the transfer function depends on the modeled processing semantics.

A typical example for an abstract performance component in the context of MPA-RTC is a *Greedy Processing Component* (GPC), shown in Fig. 5.3. It models a task that is triggered by the events of the incoming event stream which queue up in a first-in first-out (FIFO) buffer. The task processes the events in a greedy (work-conserving) fashion, while

being restricted by the availability of processing resources. The worst-case response time experienced by an event at a GPC and the worst-case backlog of a GPC satisfy the following upper bounds:

$$\sup_{\lambda \geq 0} \left\{ \inf\{\tau \geq 0 : \alpha^u(\lambda) \leq \beta^l(\lambda + \tau)\} \right\} \stackrel{\text{def}}{=} Del(\alpha^u, \beta^l) , \tag{5.1}$$

$$\sup_{\lambda \geq 0} \{\alpha^u(\lambda) - \beta^l(\lambda)\} \stackrel{\text{def}}{=} Buf(\alpha^u, \beta^l) . \tag{5.2}$$

Having a delay requirement D for a GPC component means that we require $Del(\alpha^u, \beta^l) \leq D$ for $\forall \Delta \in \mathbb{R}^{\geq 0}$ which can be expressed as the following inequality:

$$\alpha^u(\Delta - D) \leq \beta^l(\Delta) \qquad \forall \Delta \in \mathbb{R}^{\geq 0} . \tag{5.3}$$

A GPC is said to be schedulable, if for all events of the triggering event stream the above inequality is satisfied.

Performance components are connected in a network which reflects the dataflow as well as the hardware architecture of the modeled system. Some scheduling policies for shared resources are expressed by the way abstract resource streams β are distributed among the different abstract components. For instance, in a preemptive FP scheduling scheme with two tasks the lower priority task only gets the resources left over by the higher priority task. In the MPA framework this is modeled by connecting the service stream output β' of the higher priority GPC with the service stream input of the lower priority GPC as shown in Fig. 5.3. We find a lower bound for β' as follows:

$$\beta'^l(\Delta) = \sup_{0 \leq \lambda \leq \Delta} \{\beta^l(\lambda) - \alpha^u(\lambda)\} . \tag{5.4}$$

Other scheduling policies are modeled by means of abstract performance components with multiple event stream inputs and tailored internal relations. An example is the abstract performance component for EDF scheduling as described in Chapter 3 Section 3.4.1.3.

5.5 Mode Change Model

In this section we define a model for mode changes which provides the basis for the timing analysis described in the following sections. A *mode change request* (MCR) can be initiated by the environment or by the system. An MCR is asynchronous and can happen at any time of a mode execution denoted by t_{MCR}. In order to exclude interference of multiple mode changes, we assume that a new MCR cannot occur during a transition

between modes. We will refer to the mode in which a change is initiated as mode I. The target mode of a change will be called mode II. Each mode comprises a set of tasks to execute. The task set or parameters of single tasks can change only at mode switches. Each task is associated with an activation stream expressed as an arrival curve α, a best-case execution demand b, a worst-case execution demand w, and a deadline D. These parameters are defined for all modes however, they may be different for the different modes. For mode I, they will be denoted as $(\alpha_I, b_I, w_I, D_I)$, and for mode II as $(\alpha_{II}, b_{II}, w_{II}, D_{II})$, respectively. During a mode change, we differentiate between four types of tasks which are illustrated in Fig. 5.4. They are described as follows:

Added tasks are active in mode II, and inactive in mode I. Therefore, we have $\alpha_I(\Delta) = 0\ \forall\Delta$. Activations for these tasks will be accepted only at time $t \geq t_{\text{MCR}}$.

Completed tasks are active in mode I, and inactive in mode II, thus $\alpha_{II}(\Delta) = 0\ \forall\Delta$. Activations for these tasks will be accepted only at time $t < t_{\text{MCR}}$. For all activations the task execution needs to be completed, even if this happens after t_{MCR}.

Unchanged tasks are active in mode I and mode II with the same parameters. An MCR does not affect them.

Changed tasks are active in mode I and mode II with different parameters $\alpha_I \neq \alpha_{II} \vee w_I \neq w_{II} \vee b_I \neq b_{II} \vee D_I \neq D_{II}$. They are activated with the first set of parameters for $t < t_{\text{MCR}}$ and with the second set of parameters for $t \geq t_{\text{MCR}}$. As for the completed tasks, changed tasks need to complete all executions even if this happens after t_{MCR}.

The potential transient overlap in the workload deriving from tasks of modes I and II after the MCR can lead to an overload situation for the system. As indicated in Section 5.3, the overload can be avoided by delaying the start of tasks in mode II by an offset of length δ. In this case, activations for added tasks will be accepted only at time $t \geq t_{\text{MCR}} + \delta$. For changed tasks, no activations are accepted in the interval $[t_{\text{MCR}}, t_{\text{MCR}} + \delta)$. They are activated with the parameters of mode I for $t < t_{\text{MCR}}$ and with the parameters of mode II for $t \geq t_{\text{MCR}} + \delta$.

Schedulability for a system undergoing a mode change is defined as all tasks in the system *always* meeting their deadlines (in mode I, mode II, and during the transition). In the next section we present a method for schedulability analysis across mode changes under the FP scheduling policy. We assume schedulability for both modes in mutual exclusion, and we show how to analyze schedulability during the transition.

As MPA allows us to incorporate task parameters b and w into the arrival curve of the activating stream α, in the next section we discuss only changes in α and D, i.e. we consider only workload-based arrival curves.

5.6 Mode Change Analysis

Consider a uniprocessor system which executes multiple tasks according to a preemptive FP scheduling policy. Consider two modes, I and II, in which the system is schedulable, and a change from mode I to mode II for which schedulability needs to be proven. Assume that for a generic task τ a service curve $\tilde{\beta}^l$ is given as input that lower bounds the service which is available to τ for *all* time intervals, i.e. it is valid in mode I, mode II, and during the transition. If τ is an unchanged task with activation pattern α and deadline D, using (5.3) we can check for schedulability, and with (5.4) we can directly compute bounds on the remaining service $\tilde{\beta}'$.

If τ is a changed task, we have to take into account that its workload changes from α_I to α_{II} and its deadline from D_I to D_{II}. We want to find an arrival curve $\tilde{\alpha}^u$ which upper bounds the workload of τ for *all* time intervals. Then given $\tilde{\alpha}^u$ we can compute the remaining service that is available for lower priority tasks by means of (5.4). Here we consider two different cases. In the first case, we assume that the switch from mode I to mode II is immediate, i.e. there is no offset, $\delta = 0$. In the second case, we consider an offset of length $\delta > 0$ between the two modes. Note that added and completed tasks can be treated as changed tasks and therefore are not discussed separately here.

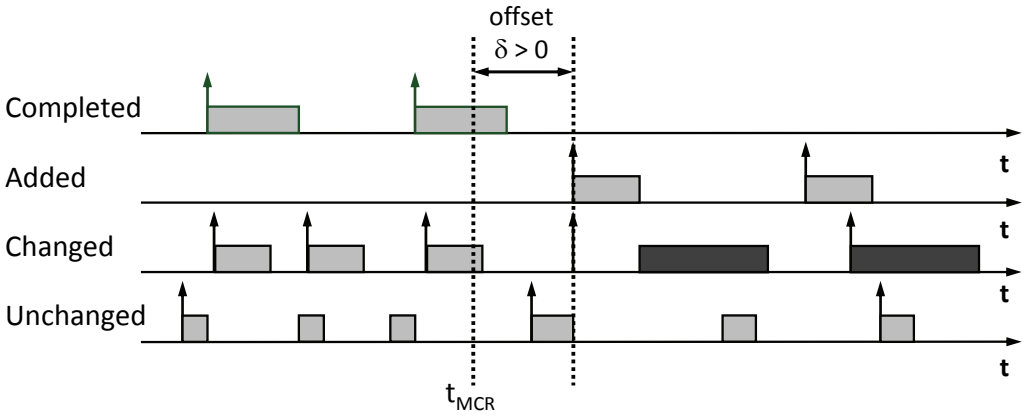

Fig. 5.4: Types of tasks considered during the mode change analysis

5.6.1 Immediate Start of Mode II Tasks

In this case, mode I activations of τ stop at t_{MCR}, and mode II activations are accepted for $t \geq t_{\text{MCR}}$. In order to guarantee the schedulability of τ, we have to verify that the task always meets its deadlines. For events of mode I the deadline is guaranteed to be met if $Del(\alpha_I^u, \tilde{\beta}^l) \leq D_I$. For events of mode II, we have to consider not only the worst-case activation pattern of mode II given by α_{II}^u, but also the fact that all buffered events of mode I need to be processed before any event of mode II. Thus, all events of mode II are guaranteed to meet their deadline if $Del(\alpha_{II}^u + Buf(\alpha_I^u, \beta_I^l), \tilde{\beta}^l) \leq D_{II}$.[1]

In order to safely bound the remaining service for lower priority tasks, we need to find an arrival curve $\tilde{\alpha}^u$ that upper bounds the workload of τ for *all* time intervals, i.e. which is valid in mode I, mode II, and also during the transition. Such an arrival curve is computed in the following theorem.

The result can be understood by looking at time intervals of length Δ that start before the mode change request, and end after it. In terms of the resulting arrival curve we need to consider the worst-case with respect to *any* location of this time interval relative to the mode change request, i.e. starting $\Delta - \lambda$ time units before t_{MCR} and ending λ time units after t_{MCR}. It takes into account the fact that the bursts from both activation streams, α_I^u and α_{II}^u, can appear very close to the mode change request.

Theorem 5.1: (Upper Bound on the Active Workload of a Changed Task during Mode Changes without Offset) *Given a changed task τ that is activated by stream α_I^u in mode I, and by stream α_{II}^u in mode II, when switching from mode I to mode II with offset $\delta = 0$, has a maximum workload $\tilde{\alpha}^u$ computed as follows:*

$$\tilde{\alpha}^u(\Delta) = \sup_{0 \leq \lambda \leq \Delta} \{\alpha_I^u(\Delta - \lambda) + \alpha_{\text{II}}^u(\lambda)\} . \tag{5.5}$$

Proof. Consider a time interval $[s, t)$ with $t > s$ and $t - s = \Delta$. Then we can distinguish three cases that are depicted in Fig. 5.5:

a) $t \leq t_{\text{MCR}}$ (the interval lies entirely before the MCR)

b) $s \geq t_{\text{MCR}}$ (the interval lies entirely after the MCR)

c) $s < t_{\text{MCR}} < t$ (the interval spans across the MCR)

[1] The sum of an arrival curve α and a constant C is an arrival curve α_S defined as $\alpha_S(\Delta) = \alpha(\Delta) + C$ for $\Delta > 0$ and $\alpha_S(\Delta) = 0$ otherwise.

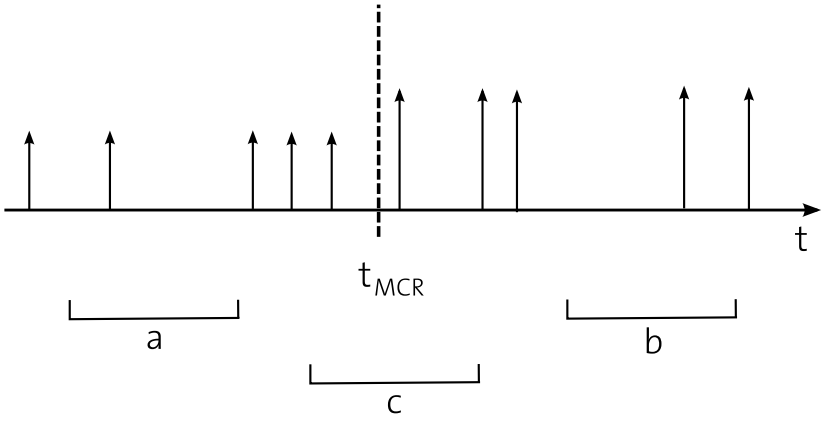

Fig. 5.5: Case distinction for workload bounding when there is no offset, $\delta = 0$

From the definition of an upper arrival curve we can derive the following inequalities:

$$R[s,t) \leq \alpha_I^u(\Delta) \qquad \text{for} \quad t \leq t_{\text{MCR}} \quad \text{(case a)}$$

$$R[s,t) \leq \alpha_{II}^u(\Delta) \qquad \text{for} \quad s \geq t_{\text{MCR}} \quad \text{(case b)}$$

$$R[s,t) = R_I[s,t_{\text{MCR}}) + R_{II}[t_{\text{MCR}},t) \qquad \text{for} \quad s < t_{\text{MCR}} < t \quad \text{(case c)}$$

$$\begin{aligned}
&\text{Subst.} \quad \lambda := t - t_{\text{MCR}} \\
&= R_I[s,t-\lambda) + R_{II}[t-\lambda,t) \\
&= R_I[s,s+\Delta-\lambda) + R_{II}[s+\Delta-\lambda,s+\Delta) \\
&\leq \alpha_I^u(\Delta-\lambda) + \alpha_{II}^u(\lambda) \\
&\leq \sup_{0\leq\lambda\leq\Delta} \left\{\alpha_I^u(\Delta-\lambda) + \alpha_{II}^u(\lambda)\right\}
\end{aligned}$$

Since we have the inequalities:

$$\begin{aligned}
\sup_{0\leq\lambda\leq\Delta} \left\{\alpha_I^u(\Delta-\lambda) + \alpha_{II}^u(\lambda)\right\} &\geq \alpha_I^u(\Delta) \qquad \forall\Delta \in \mathbb{R}^{\geq 0}\,, \\
\sup_{0\leq\lambda\leq\Delta} \left\{\alpha_I^u(\Delta-\lambda) + \alpha_{II}^u(\lambda)\right\} &\geq \alpha_{II}^u(\Delta) \qquad \forall\Delta \in \mathbb{R}^{\geq 0}\,.
\end{aligned}$$

Then we have that $\tilde{\alpha}^u(\Delta) := \sup_{0\leq\lambda\leq\Delta} \left\{\alpha_I^u(\Delta-\lambda) + \alpha_{II}^u(\lambda)\right\}$ is a valid upper bound for all three cases.

■

5.6.2 Delayed Start of Mode II Tasks

In this case, mode I activations of τ stop at t_{MCR}, while mode II activations are accepted only for $t \geq t_{\text{MCR}} + \delta$ with $\delta > 0$. In order to guarantee

schedulability, again we have to make sure that deadlines are always met. For events of mode I we have to check that $Del(\alpha_I^u, \tilde{\beta}^l) \leq D_I$. For events of mode II, we have to consider not only the existence of events remaining from mode I, but also that some of them could be processed during the offset interval. Thus, all events of the mode II are guaranteed to meet their deadline if $Del(\alpha_{II}^u + \max\{0, Buf(\alpha_I^u, \beta_I^l) - \tilde{\beta}^l(\delta)\}, \tilde{\beta}^l) \leq D_{II}$. The workload of τ is safely upper bounded by the equation in the following theorem which is valid in mode I, mode II, and also during the transition.

In comparison to (5.5), we just shift the arrival curve associated to the second mode by the offset δ. In addition, we need to explicitly consider the arrival curve of mode II; in (5.5) it is implicitly taken into account.

Theorem 5.2: (Upper Bound on the Active Workload of a Changed Task during Mode Changes with Offset) *Given a changed task τ that is activated by stream α_I^u in mode I, and by stream α_{II}^u in mode II, when switching from mode I to mode II with offset $\delta > 0$, has a maximum workload $\tilde{\alpha}^u$ computed as follows:*

$$\tilde{\alpha}^u(\Delta) = \max\left\{\alpha_{\mathrm{II}}^u(\Delta),\ \sup_{0\leq\lambda\leq\Delta}\{\alpha_I^u(\Delta-\lambda) + \alpha_{\mathrm{II}}^u(\lambda-\delta)\}\right\}. \tag{5.6}$$

Proof. Consider a time interval $[s, t)$ with $t > s$ and $t - s = \Delta$. Then we can distinguish six cases that are depicted in Fig. 5.6:

a) $t \leq t_{\mathrm{MCR}}$

b) $s \geq t_{\mathrm{MCR}} + \delta$

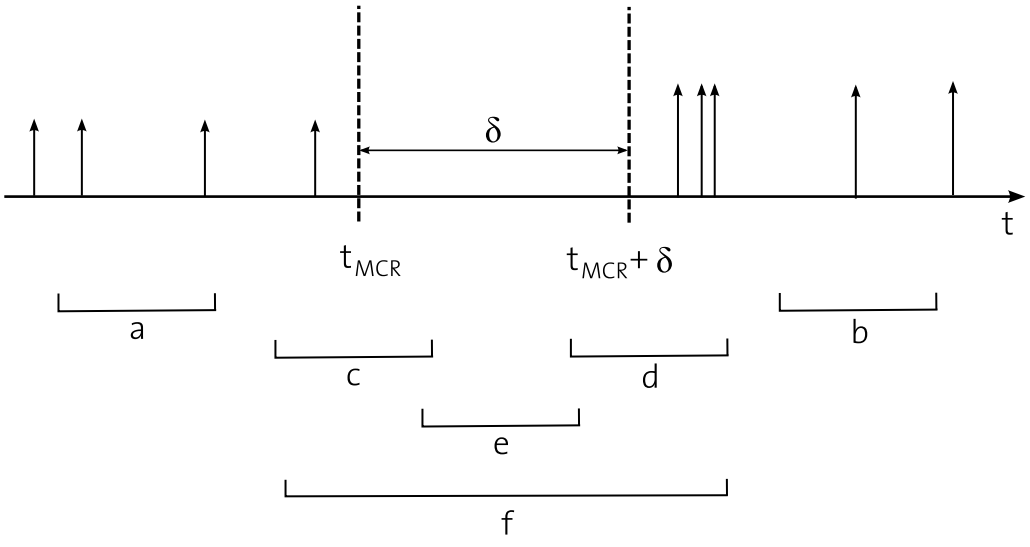

Fig. 5.6: Case distinction for workload bounding in the presence of an offset time $\delta > 0$

c) $s < t_{\text{MCR}} < t \leq t_{\text{MCR}} + \delta$

d) $t_{\text{MCR}} \leq s < t_{\text{MCR}} + \delta < t$

e) $t_{\text{MCR}} \leq s < t \leq t_{\text{MCR}} + \delta$

f) $s < t_{\text{MCR}} < t_{\text{MCR}} + \delta < t$

Using the definition of arrival curve and considering that no events arrive during the offset interval $[t_{\text{MCR}}, t_{\text{MCR}} + \delta)$, we can derive the following inequalities:

$$
\begin{aligned}
&R[s,t) \leq \alpha_I^u(\Delta) && \text{for } t \leq t_{\text{MCR}} \text{ (case a)}\\
&R[s,t) \leq \alpha_{II}^u(\Delta) && \text{for } s \geq t_{\text{MCR}} + \delta \text{ (case b)}\\
&R[s,t) \leq \alpha_I^u(t_{\text{MCR}} - s) \leq \alpha_I^u(\Delta) && \text{for } s < t_{\text{MCR}} < t \leq t_{\text{MCR}} + \delta \text{ (case c)}\\
&R[s,t) \leq \alpha_{II}^u(t - t_{\text{MCR}} - \delta) \leq \alpha_{II}^u(\Delta) && \text{for } t_{\text{MCR}} \leq s < t_{\text{MCR}} + \delta < t \text{ (case d)}\\
&R[s,t) = 0 && \text{for } t_{\text{MCR}} \leq s < t \leq t_{\text{MCR}} + \delta \text{ (case e)}\\
&R[s,t) = R_I[s, t_{\text{MCR}}) + R_{II}[t_{\text{MCR}} + \delta, t) && \text{for } s < t_{\text{MCR}} < t_{\text{MCR}} + \delta < t \text{ (case f)}\\
&\qquad \leq \alpha_I^u(t_{\text{MCR}} - s) + \alpha_{II}^u(t - t_{\text{MCR}} - \delta)\\
&\qquad \text{Subst.} \quad \lambda := t - t_{\text{MCR}}\\
&\qquad = \alpha_I^u(t - \lambda - s) + \alpha_{II}^u(\lambda - \delta)\\
&\qquad = \alpha_I^u(\Delta - \lambda) + \alpha_{II}^u(\lambda - \delta)\\
&\qquad \leq \sup_{0 \leq \lambda \leq \Delta} \left\{ \alpha_I^u(\Delta - \lambda) + \alpha_{II}^u(\lambda - \delta) \right\}
\end{aligned}
$$

If we take the maximum of the right hand sides of the inequalities, we get an upper bound for the workload of τ which is valid in all of the cases:

$$\tilde{\alpha}^u(\Delta) := \max \left\{ \alpha_I^u(\Delta), \alpha_{II}^u(\Delta), \sup_{0 \leq \lambda \leq \Delta} \left\{ \alpha_I^u(\Delta - \lambda) + \alpha_{II}^u(\lambda - \delta) \right\} \right\} .$$

Since we have the following inequality:

$$\sup_{0 \leq \lambda \leq \Delta} \left\{ \alpha_I^u(\Delta - \lambda) + \alpha_{II}^u(\lambda - \delta) \right\} \geq \alpha_I^u(\Delta) \qquad \forall \Delta \in \mathbb{R}^{\geq 0} ,$$

therefore we have the following upper bound:

$$\tilde{\alpha}^u(\Delta) := \max \left\{ \alpha_{II}^u(\Delta), \sup_{0 \leq \lambda \leq \Delta} \{ \alpha_I^u(\Delta - \lambda) + \alpha_{II}^u(\lambda - \delta) \} \right\} .$$

■

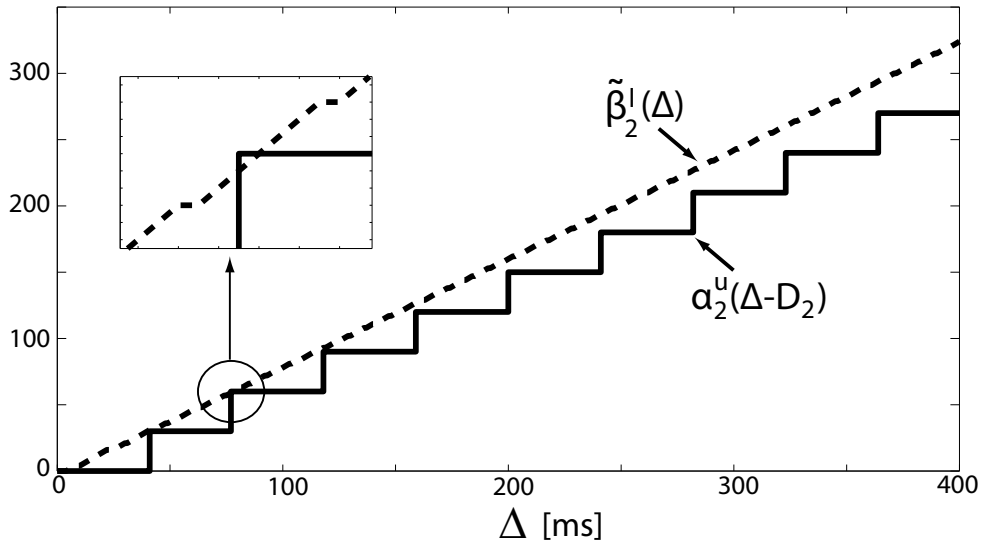

Fig. 5.7: Insufficient remaining service for T_2

Example 5.3: *Let us consider again the application scenario presented in Section 5.3. We want to determine the length of the offset δ which is sufficient to guarantee all timing constraints for the video decoding. First we represent the arrival patterns of S_1, $S_{1\,new}$, and S_2 with appropriate arrival curves $\alpha_{1,I}$, $\alpha_{1,II}$, and α_2, respectively. This is simple due to the periodic with jitter nature of the event streams. Then, we compute bounds for the workload of T_1 which are valid for all time intervals by means of (5.6). At this point we can safely lower bound the remaining service $\tilde{\beta}_2$ available for T_2 during the mode switch using (5.4). In order to verify whether it is sufficient for the timely decoding of video stream S_2, we use (5.3) with $\tilde{\beta}_2^{\,l}$ and α_2^u. For the described mode change with an offset of $\delta = 21\,ms$ the inequality is not satisfied, as highlighted in the plot of Fig. 5.7. This confirms the deadline miss observed in the trace shown in Fig. 5.2.*

A good approximation for a safe size of δ can be found by iterating the analysis procedure according to a binary search strategy. Following this method, we can derive an offset of length $\delta = 24\,ms$ for a safe mode change.

5.7 Case Study

In this section we show how the proposed theory can be applied to the analysis of the system described in Section 5.3 with realistic MPEG-2 video streams. We consider that a mode change occurs in one of the video streams and we want to reduce its effects on the other video stream. The case study illustrates that the proposed methods can be applied to the mode change analysis of tasks scheduled with fixed priorities and

activated by streams with arbitrary complex arrival patterns. We show that a designer can find a sufficient offset such that a mode change in certain tasks does not violate the timing constraints of other tasks. In the case study, first we run a simulation with different sets of video streams and collect data for their arrival patterns and workload demands, then we compute workload arrival curves which we use for the mode change analysis.

Experimental setup. We simulate the system architecture shown in Fig. 5.1 with two different sets of video clips. The first set consists of regular clips with moderate to high motion content. It is selected to be representative for clips having high workload for the CPU. The second set consists of clips displaying still images. It is representative for low workload clips. All of the clips are MPEG-2 encoded and have the same frame resolution of 704×576 pixels. Each frame is made up of 1584 macroblocks. The down-scaling for the small PiP window is being done at the output device. The two video streams arrive at a constant bitrate of 8 Mbps and the two playout buffers are read at a constant rate of 25 frames/s. The CPU is set to run at a frequency of 3 GHz.

The two MPEG-2 decoding tasks have been mapped to the CPU. Both of them implement the variable length decoding (VLD), the inverse quantization (IQ), the inverse discrete cosine transform (IDCT), and the motion compensation (MC) functions for the two streams, respectively. A scheduler schedules the two tasks according to the selected QoS parameters for each stream. In this application the QoS has been reflected in the predetermined priorities of the tasks. The simulation for the two sets of video clips have been performed using a SimpleScalar model [ALE02] of the CPU (with the *sim-profile* configuration and the PISA instruction set). The video streams are modeled at the macroblock granularity. Each compressed macroblock in the input stream is made up of a variable number of bits. Therefore, at the macroblock granularity, a constant bitrate translates into a bursty arrival pattern. This can be observed in the derived upper arrival curve for the set of high workload video streams shown in Fig. 5.8(a) which is expressed in number of macroblocks arriving per time unit. Similarly, the number of processor cycles required to process a macroblock is also variable. During the simulation, workload data was gathered for the two tasks processing the two different sets of streams. It represents the worst-case execution demand in CPU cycles for processing one, two, and more consecutive macroblocks in a video stream. For the analysis, this data was combined with the arrival pattern of macroblocks in order to obtain workload-based arrival curves. The corresponding curve for the high workload set is shown in Fig. 5.8(b).

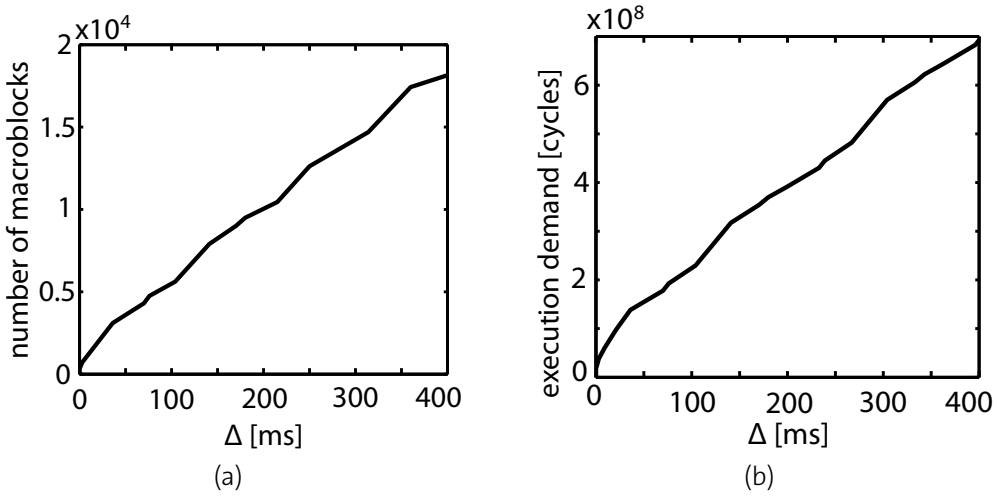

Fig. 5.8: Upper bound on the workload for the set of high-motion videos in terms of: **(a)** number of macroblocks per ms, and **(b)** number of required processor cycles per ms

Mode change analysis. The analysis starts from the fact that the system can meet all deadlines in all modes in mutual exclusion. Mode changes are first analyzed with offset $\delta = 0$. If the system is found schedulable then the mode change can be performed safely without any offset. However, if the system is found unschedulable, the analysis is performed repetitively with different sizes of δ which are chosen with a binary search strategy. Analysis stops when the smallest δ is found that makes the system schedulable during the mode change.

Results. The CPU scheduler implements a FP scheduling scheme. Context switches are considered to take negligible times. For both scenarios we perform a mode change where the video displayed on window W_1 changes from video stream S_1 (high motion content) to a video stream $S_{1\,\text{new}}$ with a lower workload (still images). We find an offset which is the smallest offset such that video stream S_2 meets its deadlines during the mode change, i.e. there is no disruption in the video displayed in window W_2.

We consider FP scheduling of the two MPEG-2 decoding tasks where stream S_1 ($S_{1\,\text{new}}$) is processed with a higher priority than S_2. S_1 and $S_{1\,\text{new}}$ have both a maximum delay requirement of 14 ms. In both modes, stream S_2 does not change. It has low workload and a maximum delay requirement of 89 ms. The analysis is first performed for an immediate change from S_1 to $S_{1\,\text{new}}$ without any offset. As can be seen in Fig. 5.9, when $\delta = 0$ the CPU service remaining for task T_2 which processes stream S_2 is not sufficient to meet the delay requirement. After repeating the

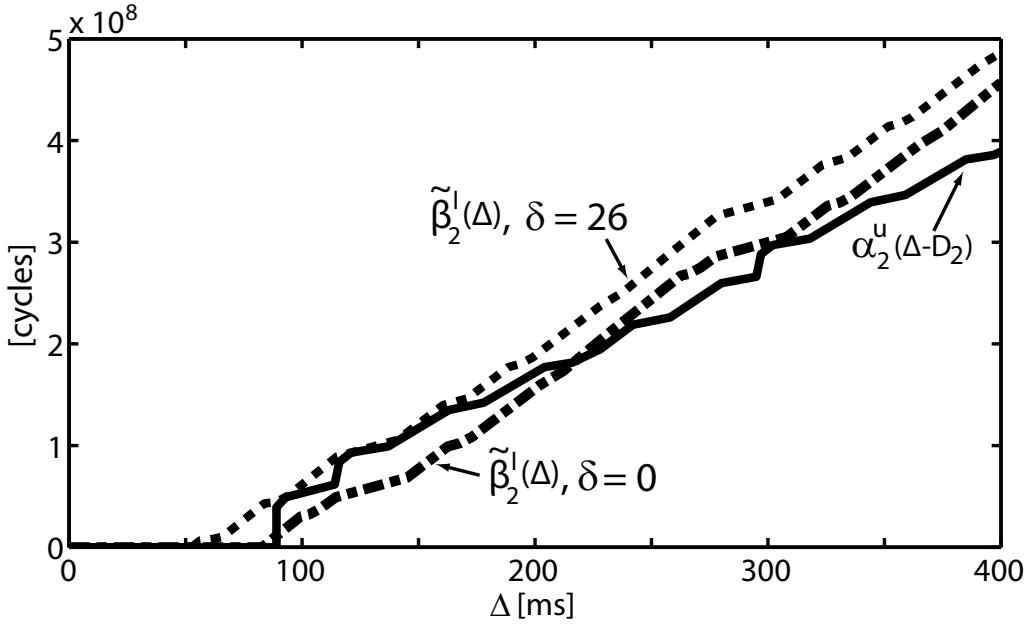

Fig. 5.9: Remaining service for stream S_2 with and without an offset during a mode change

analysis for various δ chosen by binary search, the smallest offset found is $\delta = 26$ ms. The result is shown in Fig. 5.9.

The chosen scenario shows the counterintuitive result that performing a mode change to a mode with lower system utilization can lead to timing violations during the transition period.

5.8 Discussion

In this chapter we presented a new approach for the design and analysis of adaptive multi-mode embedded real-time systems. We introduced methods for schedulability analysis during mode transitions. They guarantee the timing behavior of multi-mode systems with FP scheduling of an arbitrary number of tasks. They can also be applied to systems with EDF scheduling or any hierarchical combination of the FP and EDF scheduling policies [SPT09]. The analysis is not bound to particular event stream models, but supports any arbitrary task activation pattern. We looked at immediate switches between modes, and showed that such changes often involve a transient overload of the system. Subsequently, we considered mode changes with an offset for the start of the new mode tasks, and we showed how an offset can be used to transform an unschedulable transition into a schedulable one.

The method presented here is restricted to analysis of mode changes in uniprocessor systems. Mode change analysis techniques for multiprocessor systems have been considered in [NGA09, YNG10] but they are restricted only to task sets without dataflow dependencies between them which are activated by sporadic event streams.

Mode changes for more general distributed systems are considered in [HE07]. However, the authors limit their discussion to identifying recurring effects of a mode change as the main problem for the analysis of an example system, without devising a detailed analysis procedure for the general distributed case.

The results presented in this chapter can be a starting point for developing a schedulability analysis for mode changes in distributed systems where tasks have dataflow dependencies. While such a method would be trivial to derive for systems without dataflow cycles, considering the general case where tasks may participate in dataflow cycles seems to be much more difficult as mode change effects may propagate in the system for arbitrary long time intervals.

This chapter discusses only mode changes in applications, i.e. changes in the task sets and their parameters. However, many real-time systems use scheduling servers in order to guarantee temporal isolation between tasks. It is likely that server parameters may need to be changed dynamically during run-time, e.g. when tasks are dynamically added or removed, or when tasks change their execution demands during run-time. Such scheduler mode changes require more complex schedulability analysis methods as they need to guarantee during mode transitions not only that individual servers do not miss deadlines, but also that none of the applications served by the servers do not miss deadlines. A designer would need not only to verify mode change schedulability, but also to design algorithms (protocols) for server mode changes that will guarantee schedulability of both servers and applications. This is the topic of discussion of Chapter 6.

6

Analysis of Adaptive Schedulers

Many real-time applications are designed to work in different operating modes each characterized by different functionality and resource demands. With each mode change, resource demands of applications change, and static resource reservations may not be feasible anymore. Dynamic environments where applications may be added and removed at run-time also need to adapt their resource reservations. In such scenarios, resource reconfigurations are needed for changing the resource reservations during run-time and achieve better resource allocations.

Chapter 5 discusses how an application can perform safe mode transitions. It develops a schedulability mode change analysis for fixed priority scheduled uniprocessor systems. It considers how to switch from one schedulable task set to another one with different parameters, and still guarantee that no task misses a deadline during the mode switch. However, the problem of mode switches in schedulers or how to switch safely from one set of schedulable resource reservations to another one has not been addressed. A resource scheduler should be reconfigured online in such a way that it still guarantees a certain amount of resources during the reconfiguration process, otherwise applications may miss deadlines. The current chapter proposes a framework for scheduling real-time applications through scheduling servers that provide resource reservations, and algorithms for changing the resource reservations online while still guaranteeing the feasibility of the system and the schedulability of applications.

Chapters 2 and 3 deal with interface-based design frameworks for real-time systems. They specify interfaces for system components that can be used during design-time to find out for example what is the minimum resource reservation needed by a component in order to make it schedulable. If a component changes its operating mode and functionality, interface-based design theories can be used to compute the new minimum resource reservation needed for scheduling however,

the methods do not provide any information how to switch safely to the new reservation. The current chapter goes into more details about mode switches in resource schedulers. It deals particularly with resource reservations provided by Time Division Multiple Access (TDMA) schedulers, and develops algorithms that can switch under different scenarios from one set of schedulable resource reservations to another one while keeping the system schedulable during the switch.

6.1 Introduction

The server architecture paradigm has been seriously considered in the past years for its ability to separate the scheduling concerns between the system and the application levels. A server mechanism is strictly connected with the resource partition idea [LLB06, SBNN08] where a shared resource, e.g. CPU computation time, is used by several applications. Servers are used to isolate the temporal behavior of real-time tasks through resource reservations [MRZ94]. Abeni and Buttazzo [AB04] introduced a bandwidth reservation mechanism, namely the Constant Bandwidth Server (CBS), that allows real-time tasks to execute in dynamic environments under a temporal protection mechanism, so that a server never exceeds a predefined bandwidth, independently of the actual requests of the tasks served by it.

Server models can be classified into *event-driven servers* where the servers are driven by the application requirements. The CBS [AB98] and the sporadic server [SSL89] are typical examples. In *time-triggered servers*, the server resource supply is driven by a predefined timing pattern that depends only on the server properties. An example is the Time Division Multiple Access (TDMA) server paradigm where the resource is periodically partitioned [WT06b]. In particular, a TDMA server assigns time slots of fixed lengths to its applications that repeat in each cycle.

Nowadays, dynamic real-time applications ask for real-time systems that can adapt their behavior at run-time by changing their operating mode: the computing environment and the available resource of a system may change over time. For example, adding a new task into the system at run-time may result in a reduction of the computing resources being allocated to existing tasks. Moreover a change in the operating mode of an application, e.g. from start-up to normal, or from normal to shut-down, may also demand re-allocation of the computing resources among the tasks. That and many other scenarios require flexible workload management and resource allocation.

Whereas a server manages an application by supplying the resource it requires [DB06], adaptive applications must rely on adaptive servers

to meet their changing resource requirements. Servers need to be reconfigured dynamically to adapt the resource reservations and reflect the changes in the system or its environment. Such reconfigurations need to be performed online without jeopardizing schedulability. It is therefore essential to develop appropriate resource reconfiguration criteria and algorithms to manage the criticality of the resource reconfiguration phase.

In summary, this chapter makes the following contributions:

- The problem of scheduler adaptations in resource partitioned architectures is considered from the perspective of adaptive servers that provide real-time guarantees. It is the first discussion on the topic where not only schedulability of reconfigurable servers is considered, but also schedulability of applications during server reconfigurations.
- An adaptive scheduling server framework based on the TDMA partitioning paradigm is developed.
- Conditions are established that need to be met during a reconfiguration of the framework.
- All possible reconfiguration scenarios are classified.
- Algorithms are developed for each reconfiguration scenario that guarantee meeting of the real-time constraints during reconfigurations.
- Server reconfiguration schedulability analysis is presented based on the Real-Time Calculus [TCN00].

Section 6.2 discusses briefly some of the related work. Section 6.3 classifies the problems that may occur during reconfigurations of some common servers and provides simple examples illustrating them. Section 6.4 describes the proposed Adaptive Server with Guarantees (ASG), and defines the service guarantees that it provides during operation and reconfiguration. Section 6.5 classifies the possible reconfiguration scenarios, provides algorithms, and analyzes schedulability for each of them. Section 6.6 illustrates the algorithms with a case study. Finally, Section 6.7 concludes this chapter with a brief discussion.

6.2 Related Work

To cope with applications in which the computational demand is highly variable, using fixed reservations can lead to under-utilization of system

resources, hence adaptive scheduling schemes need to be adopted. Buttazzo et al. [BA00] proposed an elastic scheduling methodology for adapting the rates of a periodic task set to different workload scenarios, without affecting the system schedulability. Abeni et al. [AB99] presented a framework for dynamically allocating the CPU resource to tasks whose execution times are not known a priori. Adaptive reservation techniques based on feedback scheduling have also been investigated by the authors in [AB01]. All of these frameworks are only suitable for soft real-time systems as they may experience missed deadlines.

There are also systems in which the application is characterized by multiple execution modes, each consisting of a specific task set and workload requirement. For these systems, the feasibility of the schedule has to be guaranteed not only within each individual mode, but also during mode transitions. This problem has been deeply investigated in the real-time literature [TBW92, BLCA02, PB98, SRLR89] and in Chapter 5 of this thesis. Crespo et al. [RC04] presented a survey of mode change protocols for uniprocessor systems under fixed-priority scheduling and proposed a new protocol along with its schedulability analysis. Guangming [Gua09] computed the earliest time at which a new task can be safely added to a system scheduled by the Earliest Deadline First (EDF) policy, without jeopardizing the feasibility of the task set. All of these results and the ones presented in Chapter 5 address the problem of performing mode transitions in applications without violating their schedulability. None of them considers how to *change resource reservations* online without violating schedulability of applications which is the goal of this chapter.

In real-time operating systems, servers are a specific scheduling mechanism that handles aperiodic requests as soon as possible while preserving hard periodic tasks from missing their deadlines. Another classification distinguishes between fixed priority and dynamic priority servers, depending on the scheduling policy used to schedule them. Among fixed priority servers, deferrable server [LSS87, SLS95] and sporadic server [SSL89] are the most well-known techniques that preserve their capacity when no request is pending upon the invocation of a server. Spuri et al. [SB96] presented a survey of dynamic priority servers that can efficiently work under EDF. It is also notable that time-triggered architectures play an increasingly important role in large distributed embedded systems as described in [HHK03, HE05, WT06b]. Mainly, time-triggered servers offer high predictability with enormous benefits to the analysis of real-time systems.

However, classical server paradigms and models do not allow adaptations to changing conditions. To the best of our knowledge, none of the schedulers that provide isolation and real-time guarantees

have mechanisms for online reconfiguration that can provide guarantees during the reconfiguration process. It may be possible to wait for an idle time in the system in order to reconfigure the scheduler as in [CPL08], however, it is highly unlikely that idle times occur at the same time for all applications.

Several papers have tried to face and cope with this lack. Fohler [Foh93] investigated the problem of mode changes in both the applications and the scheduler in the context of pre run-time scheduled hard real-time systems. Applications are specified with periodically activated graphs with precedence constraints for which safe switching points are pre-computed using heuristic search techniques. The FRESCOR project [HSdE09] has proposed a mode change protocol for a system with virtual resources based on the sporadic server and periodic tasks where budgets may change. Both frameworks are not as general as the results presented in this chapter which can deal with hierarchically scheduled systems with mixed schedulers and complex task activation schemes.

New mechanisms have been proposed to dynamically change server paramters at run-time. de Olivera et al. [OCL09] addressed the problem of finding optimal CBS parameters and dynamically reconfiguring the servers, offering support for multi-mode adaptive real-time applications. Valls et al. [VAdlP09] presented an adaptation protocol based on the definition of a contract model for filtering peaks in resource demands where applications are modeled with periodic, continuous, and imprecise tasks. However, in both frameworks there are no algorithms and analysis of applications schedulability for the proposed online resource reconfigurations.

Brandt et al. [BBLB03] propose the rate-based earliest deadline (RBED) scheduler where servers are periodic tasks scheduled with EDF. The paper gives system schedulability conditions for adaptations in the periods and utilizations of the servers. However, the paper does not go neither into characterizing the service provided by the servers during reconfigurations, nor into algorithms for how to control this service.

Craciunas et al. [CKP+09] propose the variable-bandwidth server (VBS) which is based on CBS but allows for adaptations. Applications are specified as sequences of actions which execute on a VBS. Activations of actions may change the parameters of the VBS, and schedulability is based on the maximum utilization from all actions of an application. Our framework is more general as server reconfigurations may happen at any time, are independent of application model, and can take advantage of application operating modes.

6.3 Motivational Examples

To illustrate the different problems that may occur during reconfigurations, we have chosen three examples of systems with TDMA servers [WT06b], static polling servers [SLR86], and CBSs [AB04]. Similar examples can be derived with other kinds of servers and show that a naive online change of parameters is not able to guarantee the system schedulability in hard real-time scenarios.

Example 6.1: *Consider the timing diagram shown in Fig. 6.1. Three TDMA servers, S_A, S_B, and S_C can operate in two modes, denoted as Old Mode and New Mode, respectively. We suppose that given an operating mode, all TDMA servers operate with the same period which equals the cycle of the TDMA. When there is a mode change, the allocated slots in the TDMA and the cycle of the TDMA may change. When a server slot becomes available, it is available regardless of whether there is workload to use it.*

Server S_A serves a single task τ_A with worst-case execution time (WCET) of 2 *ms and period of* 20 *ms which we will denote as* $(2, 20)$. *In Old Mode, server S_A has a reserved slot of* 1 *ms in a TDMA cycle of* 10 *ms denoted as* $(1, 10)$. *In New Mode, server S_A has parameters* $(3, 12)$. *Server S_B serves a single task τ_B with parameters* $(2, 5)$. *The server in Old Mode has parameters* $(5, 10)$ *and in New Mode* $(6, 12)$. *Server S_C serves a single task τ_C with parameters* $(1, 16)$. *The server in Old Mode has parameters* $(1, 10)$ *and in New Mode* $(1, 12)$.

Figure 6.1 shows a server reconfiguration performed at time $t = 20$ ms. For task τ_B this means that it suffers longer worst-case response time (WCRT) of 9 *ms during the reconfiguration whereas its WCRT is* 7 *ms in the Old Mode and* 8 *ms in the New Mode. Similarly task τ_C has a longer WCRT during the*

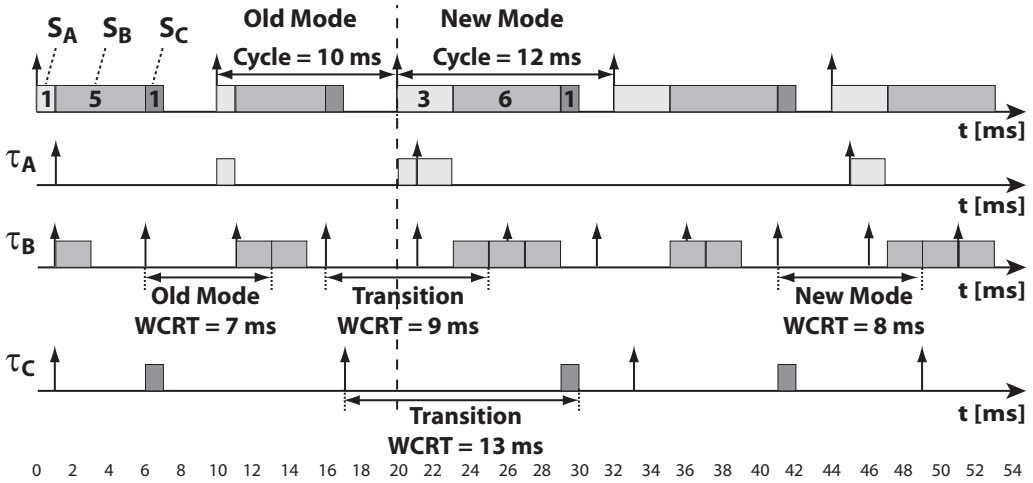

Fig. 6.1: TDMA servers reconfigured at $t = 20$ ms (dashed line) causes longer WCRTs for tasks τ_B and τ_C

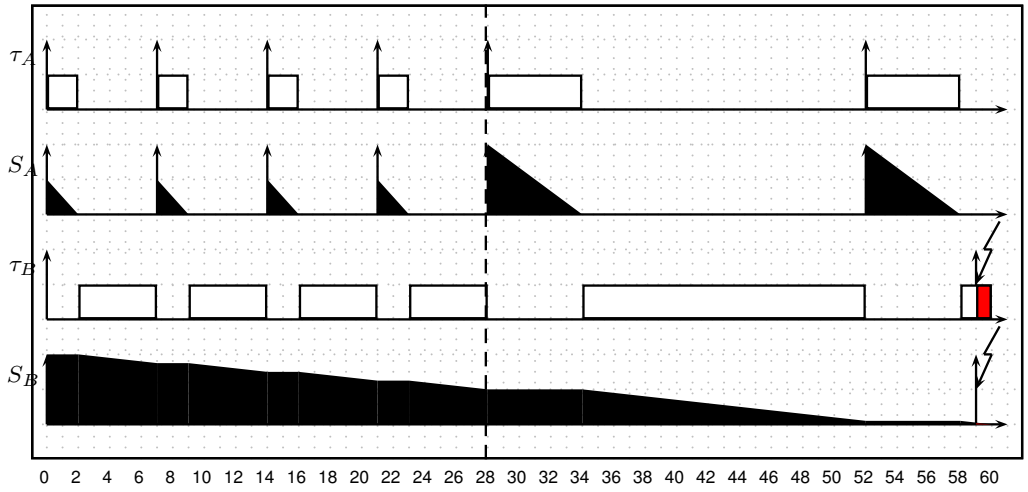

Fig. 6.2: Polling server S_A reconfigured at $t = 28$ ms (dashed line) causes a deadline miss for task τ_B and a capacity miss for server S_B

reconfiguration which equals to 13 *ms, whereas in the Old and in the New Modes it is* 10 *ms and* 12 *ms, respectively.*

Hence, a reconfiguration of TDMA servers may cause several tasks to miss deadlines.

Example 6.2: *Consider the timing diagram shown in Fig. 6.2. The example is adapted from [TBW92]. Two polling servers, S_A and S_B, are scheduled with the fixed priority policy. Server S_A has higher priority. It can operate in two modes. In Old Mode it has a budget of* 2 *ms and a period of* 7 *ms, denoted as* $(2, 7)$. *It serves a single task τ_A with WCET of* 2 *ms and deadline equal to period of* 7 *ms, denoted as* $(2, 7)$. *In New Mode S_A and τ_A have parameters* $(6, 24)$ *and* $(6, 24)$, *respectively. Server S_B and its task τ_B operate in a single mode and their parameters are* $(40, 59)$ *and* $(40, 59)$, *respectively. The system is schedulable separately in both modes.*

Figure 6.2 shows a server reconfiguration without a proper transition algorithm. Server S_A and task τ_A simultaneously enter the New Mode at time $t = 28$ ms which leads to a capacity miss for server S_B and a deadline miss for task τ_B at time $t = 59$ ms even though the mode change was performed at the end of the periods for server S_A and task τ_A.

The example illustrates that reconfiguration of one server may cause other servers to not be able to deliver their guaranteed budgets.

Example 6.3: *Consider the timing diagram shown in Fig. 6.3. CBS S_A can operate in two modes. In Old Mode it has a budget of* 4 *ms with a period of* 5 *ms denoted as* $(4, 5)$. *It serves a single task τ_A with WCET of* 8 *ms and deadline equal to period of* 10 *ms denoted as* $(8, 10)$. *In New Mode, the parameters for S_A*

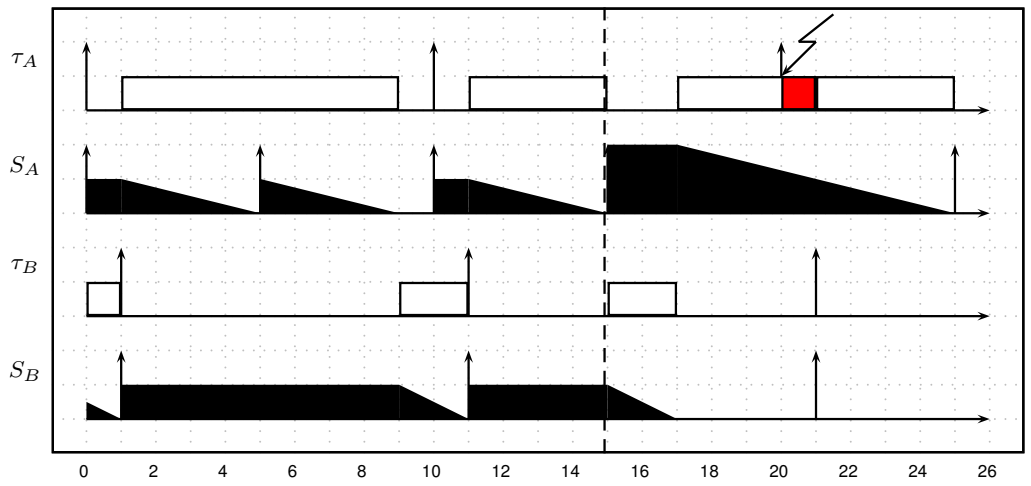

Fig. 6.3: CBS S_A reconfigured at $t = 15$ ms (dashed line) causes a missed deadline for task τ_A

are $(8, 10)$ *and* τ_A *is unchanged. CBS* S_B *serves a single task* τ_B *with parameters* $(2, 10)$ *and* $(2, 10)$*, respectively. The system is schedulable when server* S_A *is either in Old Mode or in New Mode as* $U = U_{S_A} + U_{S_B} = 1$*.*

Figure 6.3 shows a reconfiguration for server S_A *at the end of a server deadline at time* $t = 15$ *ms which leads to a missed deadline for task* τ_A *at time* $t = 20$ *ms.*

The example illustrates that reconfiguration of a server may cause the application that it serves to miss deadlines.

In summary, the problems observed during online reconfiguration of servers fall in two classes:

1. **Isolation violation:** a reconfiguration of one server may cause other servers to not be able to deliver their guaranteed capacities.

2. **Deadline violation:** a reconfiguration of a server may affect the application that it serves by making it miss deadlines.

Safe reconfiguration algorithms will have to address both problems in order to be suitable for hard real-time systems.

6.4 System Framework

In this section, we give an overview of a framework with adaptive resource reservations. There are many scenarios for the use of such a framework and many different ways to realize it. We focus on the scheduling servers and their properties. In our framework, applications

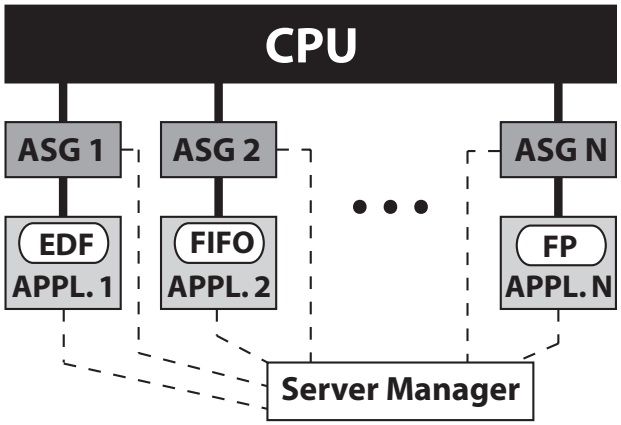

Fig. 6.4: Overview of a system where the CPU is shared by applications through multiple ASGs

share a common processor using servers and we refer to them as Adaptive Servers with Guarantees (ASG) as they guarantee resource reservations and can be reconfigured dynamically while still providing a guarantee even during the reconfiguration.

We consider a uniprocessor system that runs a set of applications. Each application is scheduled on an individual ASG. The servers provide resource reservations and guarantee isolation between applications. Applications can be of arbitrary complexity and they may even have their own schedulers, as in hierarchically scheduled systems [WT06a]. An ASG is only concerned with guaranteeing a minimum service supply to its application. The system has a single Server Manager that can control the parameters of all servers (such as their budgets and period) and is able to communicate with the applications in order to accommodate their changing resource requirements.

The overall system framework is illustrated in Fig. 6.4.

6.4.1 The Adaptive Server with Guarantees

Servers are scheduled statically by a TDMA scheme. For each server a slot of fixed size Q called budget is reserved in the TDMA time-wheel. A server is activated, i.e. its budget becomes available, when the slot of the server arrives in the TDMA time-wheel. All servers in the system are activated periodically with the same period P which equals to the cycle of the TDMA. Servers can have different budgets but always a common period. An ASG is denoted with the tuple (Q, P). A schedule of four ASGs is illustrated in Fig. 6.5.

Budgets are always given to applications regardless of whether they use them or not, like in a traditional TDMA schedule. In the following

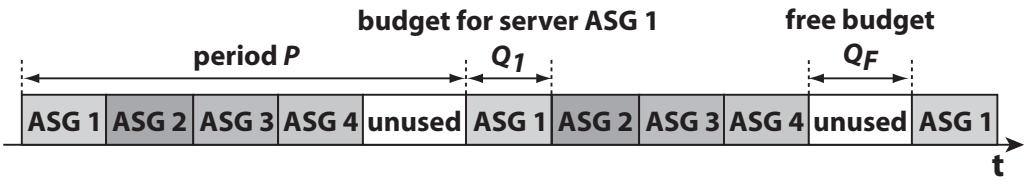

Fig. 6.5: Schedule for four ASGs

discussion, we assume that context switch overheads take negligible time but they can be trivially added to our analysis. The description of an ASG can be summarized in the following definition.

Definition 6.4: (Adaptive Server with Guarantees) *An ASG* (Q, P) *guarantees to an application access to a shared resource for* $Q > 0$ *time units every* $P > 0$ *time units, where* $Q \leq P$.

The total utilization for a system with N ASGs is defined as:

$$U = \frac{\sum_{i=1}^{N} Q_i}{P} ,$$

which is the sum of the single server utilizations Q_i/P. Such a system is schedulable when the total utilization is smaller or equal to 1:

$$U \leq 1 . \tag{6.1}$$

When the total utilization is less than 1, there is some unused budget in the system, Q_F, called the free budget. We suppose that all ASGs are scheduled from the beginning of every period one after the other, and the free budget is always at the end, as illustrated in Fig. 6.5. The free budget may be given to non real-time applications on the basis that it can always be reclaimed by the system. The free budget is essential in our framework during reconfigurations as it will be shown in Section 6.5.

6.4.2 Resource Supply of an ASG

An ASG (Q, P) may not have access to the CPU for a time interval Δ that is upper bounded by $P - Q$. After this interval, the server will have guaranteed access to the resource for Q time units. Therefore, an ASG cannot guarantee resource access for any interval of size $0 \leq \Delta \leq P - Q$. However, it guarantees service of $S(\Delta - (P - Q))$, in any interval $(P - Q) \leq \Delta \leq P$, where S is the CPU speed, e.g. cycles per time unit. Without loss of generality, we assume that $S = 1$, as all parameters in the system can be normalized according to this speed. Then the minimum resource

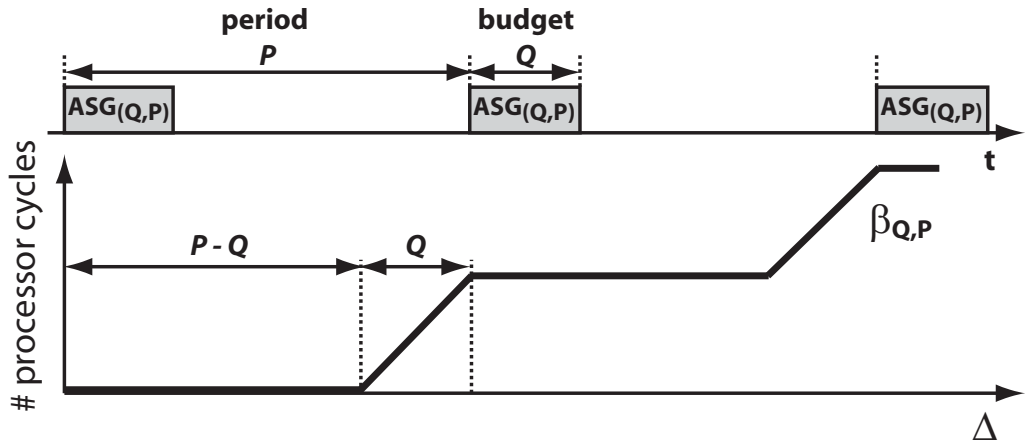

Fig. 6.6: Resource supply of an ASG (Q, P)

supply of an ASG (Q, P) in any time interval Δ can be lower bounded by the following function:

$$\beta_{Q,P}(\Delta) = \max\left(\left\lfloor \frac{\Delta}{P} \right\rfloor Q, \Delta - \left\lceil \frac{\Delta}{P} \right\rceil (P - Q)\right) ,$$

or more compactly as:

$$\beta_{Q,P}(\Delta) = \sup_{0 \le \lambda \le \Delta} \left\{ \lambda - \left\lceil \frac{\lambda}{P} \right\rceil (P - Q) \right\} . \tag{6.2}$$

The minimum resource supply for an ASG (Q, P) is illustrated in Fig. 6.6.

The minimum resource supply function (6.2) is actually a lower *service curve* as known from Network and Real-Time Calculus [Cru91a, LBT01, TCN00]. Service curves are abstract representations for the availability of processing and communication resources. A service curve $\beta(\Delta)$ gives a lower bound on the available service in *any* time interval of length $\Delta > 0$ where for $\Delta \le 0$ we have $\beta(\Delta) = 0$. The service is usually expressed in a suitable workload unit such as number of cycles for computing resources or bits for communication resources.

6.4.3 Performance Analysis

Application tasks are activated by the arrivals of events. The timing characteristics of event arrivals are described abstractly with *arrival curves* as known from Network and Real-Time Calculus. The arrival curve $\alpha(\Delta)$ denotes an upper bound on the number of events that arrive in *any* time interval of length $\Delta > 0$ where for $\Delta \le 0$, $\alpha(\Delta) = 0$. Arrival curves substantially generalize traditional event stream models such as

periodic, periodic with jitter, and sporadic. Often the domain of arrival curves are workload units. Event-based arrival curves can be converted to workload-based arrival curves by scaling with the best-case/worst-case execution demands of events. The units of the arrival and service curves used in an analysis need to be the same. In this chapter, we will use the workload-based interpretation and assume that each event has a fixed execution demand. More general concepts for characterization of these units are discussed in [MKT04] as well as in Chapter 3 Section 3.8.1.

Now given the minimum resource supply of an ASG and a characterization of the activation stream of a task, we can compute the worst-case response time (WCRT) for the task. To this end, we use results from Network and Real-Time Calculus where for a resource supply characterized with a service curve β and an input stream characterized with an arrival curve α, the WCRT of an event from the stream is the maximum horizontal distance between the arrival and the service curves computed as follows:

$$\sup_{\lambda \geq 0} \{\inf\{\tau \geq 0 : \alpha(\lambda) \leq \beta(\lambda + \tau)\}\} \stackrel{\text{def}}{=} Del(\alpha, \beta) \; . \tag{6.3}$$

Example 6.5: *To illustrate this let us consider Example 6.1 from Section 6.3. Consider server S_B in Old Mode which is an ASG with budget $Q = 5\,ms$ and a period $P = 10\,ms$. The respective service curve can be computed with equation (6.2). It serves a single periodic task τ_B with a period of $5\,ms$ and WCET of $2\,ms$. The WCRT of the task computed with equation (6.3) is shown in Fig. 6.7. The computed WCRT is equal to the one observed on the trace in Fig. 6.1.*

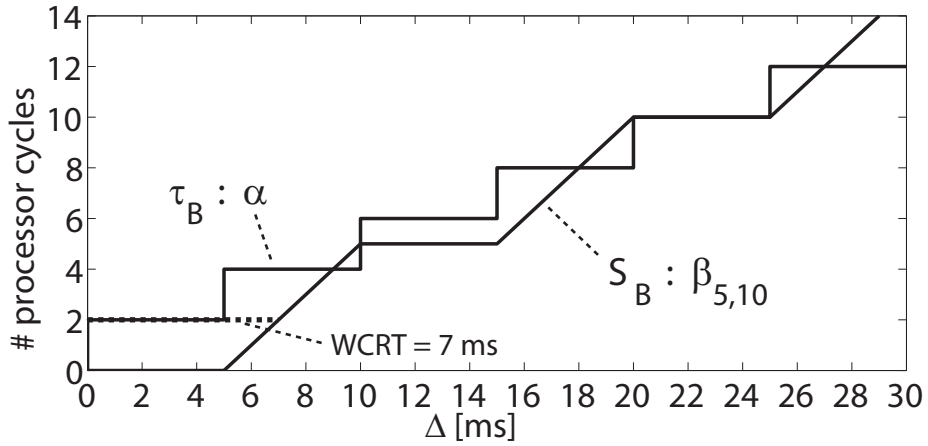

Fig. 6.7: Server S_B and task τ_B WCRT analysis

6.4.4 Schedulability of Applications

An application is schedulable if its real-time requirements are satisfied by the system. If we consider the case of a single task, we may have the requirement that all activations are processed within a relative deadline D. Given (6.3), this is expressed as $Del(\alpha, \beta) \leq D$. Inverting it w.r.t. β, we can compute a lower bound on the minimum resource demand required to meet the deadline requirement. This is expressed as follows:

$$\underline{\beta}(\Delta) \geq \alpha(\Delta - D) \qquad \forall \Delta \in \mathbb{R}^{\geq 0} \,. \tag{6.4}$$

In other words, the minimum resource demand has a lower service curve that equals to $\underline{\beta}(\Delta) = \alpha(\Delta - D)$.

By using previous results on demand bound functions by Baruah et al. [BCGM99] and interface-based design as described in Chapter 3, such a task is schedulable if a resource can supply service that is larger or equal to the demanded one. For an ASG (Q, P), schedulability would mean that:

$$\beta_{Q,P}(\Delta) \geq \underline{\beta}(\Delta) \qquad \forall \Delta \in \mathbb{R}^{\geq 0} \,, \tag{6.5}$$

where $\beta_{Q,P}$ is computed with (6.2).

Example 6.6: *In the case of task τ_B from Example 6.1, it is schedulable with a relative deadline $D = 7$ ms by server S_B with Old Mode parameters* $(5, 10)$. *This can be seen in Fig. 6.8 where the service curve of server S_B is above the shifted arrival curve of task τ_B which expresses the resource demand of the task.*

The same schedulability condition applies not only for single tasks, but even for complex applications as we can compute the minimum resource demand of an application as a single service curve $\underline{\beta}$, for details see Chapters 2 and 3.

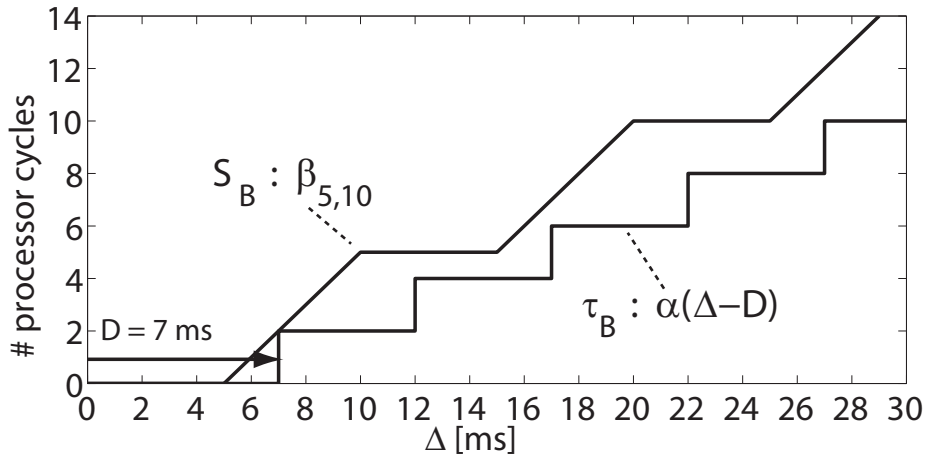

Fig. 6.8: Server S_B and task τ_B schedulability condition (6.5)

6.4.5 Schedulability during a Reconfiguration

A reconfiguration may change the server parameters such as their budgets and period from one mode to another. We consider a single reconfiguration. For a system with N ASGs before a reconfiguration they operate with parameters (Q_i^O, P^O), $1 \leq i \leq N$, (for Old Mode), and after the reconfiguration with parameters (Q_i^N, P^N), $1 \leq i \leq N$, (for New Mode). We assume that the system is schedulable in Old Mode and New Mode separately, i.e. condition (6.5) is satisfied by assumption for all servers in Old Mode:

$$\beta_{Q_i^O, P^O}(\Delta) \geq \underline{\beta_i}(\Delta) \qquad \forall \Delta \in \mathbb{R}^{\geq 0} \quad \forall i \,,$$

and for all servers in New Mode:

$$\beta_{Q_i^N, P^N}(\Delta) \geq \underline{\beta_i}(\Delta) \qquad \forall \Delta \in \mathbb{R}^{\geq 0} \quad \forall i \,.$$

During a reconfiguration or the changing from one set of server parameters to another, the system should not suffer a degraded performance. Let us consider the two problems described in Section 6.3. To prevent isolation violations, each server should be able to guarantee a service curve during a reconfiguration. To prevent deadline violations, each server should be able to guarantee a service curve that is sufficiently large during a reconfiguration.

Let us denote as $\widetilde{\beta_i}(\Delta)$ the service provided by an ASG during time intervals Δ that span Old Mode, the Reconfiguration, and New Mode. In order to prevent a degraded performance during a reconfiguration we need to have for all servers that:

$$\widetilde{\beta_i}(\Delta) \geq \min\{\beta_{Q_i^O, P^O}(\Delta), \beta_{Q_i^N, P^N}(\Delta)\} \qquad \forall \Delta \in \mathbb{R}^{\geq 0} \quad \forall i \,. \tag{6.6}$$

The above condition ensures that each server guarantees during a reconfiguration at least the minimum of the services guaranteed in Old and New Modes. This implies that each application served by an ASG during a reconfiguration is guaranteed that it will not violate the larger of the deadlines from Old and New Modes.

Example 6.7: *To illustrate this, consider server S_B from Example 6.1. During the transition from Old Mode to New Mode, if the server were able to meet condition (6.6), then the WCRT of task τ_B would have been at most the maximum of the WCRTs from the two modes which is 8 ms, and it would not have experienced the WCRT of 9 ms. This is illustrated in Fig. 6.9.*

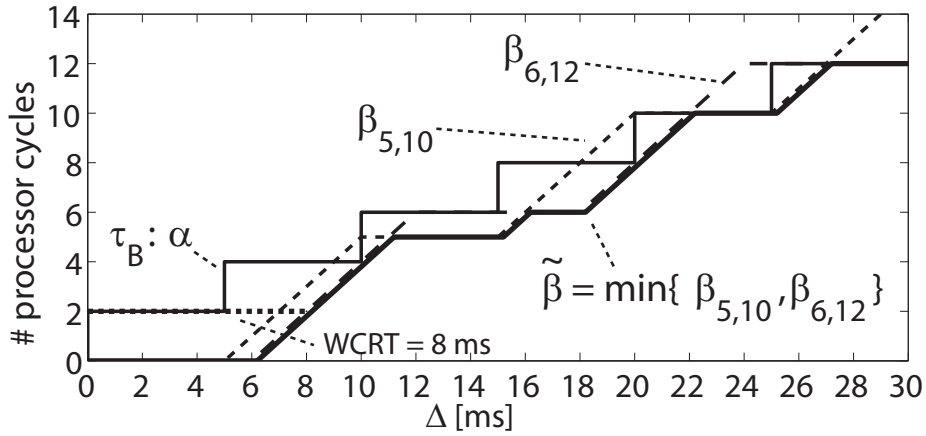

Fig. 6.9: Condition (6.6) can guarantee a WCRT of 8 ms for task τ_B during the reconfiguration in Example 6.1

6.5 Algorithms and Analysis

In this section, we classify the scenarios for feasible resource reconfigurations and provide schedulability analysis for each of them to show that they meet condition (6.6). The proposed algorithms are implemented in the Server Manager and executed by it. Initiation of a reconfiguration can be done by an application in order to request a different resource reservation, or by the Server Manager in order to achieve better resource allocation (decreased system utilization). The proposed algorithms work regardless of what the reason for reconfiguration is.

We differentiate between reconfigurations that do not change the period of the servers, i.e. $P^O = P^N$, and those that do, i.e. $P^O \neq P^N$. The possible reconfiguration scenarios are summarized in Tab. 6.1.

Reconfigurations that do not require change of period have simple feasibility conditions, and they do not require any pre-computed information except budgets and period as the decision for performing them can be made online. For the case of changing periods, the conditions

Tab. 6.1: Reconfiguration Scenarios

$P^O = P^N$	Remove a server
	Decrease of a budget
	Add a server
	Increase of a budget
$P^O \neq P^N$	Increase of period $P^O < P^N$
	Decrease of period $P^O > P^N$

are much more involved as we will see, and some parameters need to be pre-computed and stored in the Server Manager to be used online.

6.5.1 Notation

The time of the k-th activation of server (Q_i, P) is denoted as $s_{i,k}$. The time when the free budget starts is $s_{F,k}$. An activation frame k contains the k-th activations of all servers and the free budget. The time when activation frame k starts is the activation time of the first scheduled server (Q_1, P) denoted as $s_{1,k}$ and it ends when the same server is scheduled again $s_{1,k+1}$. When we would like to differentiate between any of the parameters and indicate that they belong to the Old Mode or the New Mode, we will add the superscripts O or N, respectively. In the Old Mode, all activation frames have the same length which equals to the period, $P^O = s^O_{1,k+1} - s^O_{1,k}$ for frames k in the Old Mode, unless otherwise stated. Similarly for the New Mode.

Algorithms that change the period of servers will require an intermediate phase called Reconfiguration where budgets and period will be different from the ones in Old and New Modes. Parameters belonging to the Reconfiguration will carry the superscript R when necessary. The notation is illustrated in Fig. 6.10.

For the proofs of some theorems we will need the definition of the min-plus convolution operator $\otimes$ and some of its properties which can be found in Appendix A. We will also need the facts that the Old Mode service curve of an ASG can be expressed as:

$$\beta_{Q^O,P^O}(\Delta) = (\beta_{Q^O,P^O} \otimes \beta_{Q^O,P^O})(\Delta + P^O - Q^O)\,, \tag{6.7}$$

and the New Mode service curve as:

$$\beta_{Q^N,P^N}(\Delta) = (\beta_{Q^N,P^N} \otimes \beta_{Q^N,P^N})(\Delta + P^N - Q^N)\,, \tag{6.8}$$

which can be easily proven.

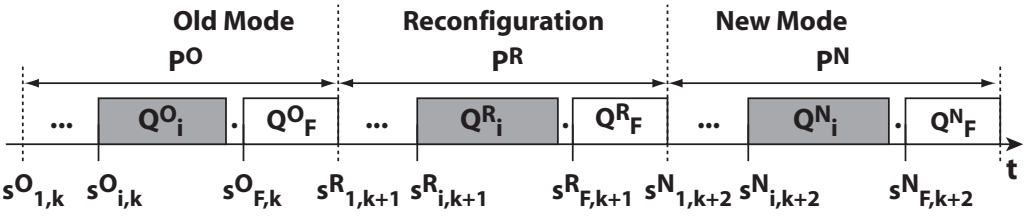

Fig. 6.10: Notation. Three activation frames where activation frame k belongs to the Old Mode, frame $k+1$ to the Reconfiguration, and frame $k+2$ to the New Mode

6.5.2 No Change of Period

Here for brevity we do not differentiate between P^O and P^N but refer to the period as P. In these scenarios the last activation frame of the Old Mode which we denote as k is followed immediately by the first activation frame of the New Mode denoted as $k+1$.

6.5.2.1 Removing an Existing ASG

Removing a server from the schedule means that in the Old Mode, it has budget $Q^O > 0$, and in the New Mode, its budget is $Q^N = 0$. The budgets of all other servers are unchanged. This is an operation that can always be performed since it decreases the utilization of the system by Q^O/P, and increases the free budget, $Q_F^N = Q_F^O + Q^O$.

Algorithm 6.1 describes removing server (Q_i^O, P) from a schedule with N servers. When the server is removed, activations of all preceding servers are unchanged while activations of succeeding servers are shifted earlier by the removed budget. This is illustrated in Fig. 6.11.

Theorem 6.8: (Service Guarantee during Removing an ASG) *Removing server (Q_i^O, P) from a schedule of N servers using Algorithm 6.1 satisfies condition (6.6) for all other servers in the system as each of them gets at least a guaranteed service during the reconfiguration of $\widetilde{\beta}_j \geq \beta_{Q_j,P}$, $1 \leq j \leq N,\ j \neq i$.*

Proof. For all servers except the removed one we have that $\beta_{Q_j^O,P} = \beta_{Q_j^N,P}$ which we denote as $\beta_{Q_j,P}$.

Algorithm 6.1 `Removing an ASG`

Input: $s_{j,k}^O,\ 1 \leq j \leq N$ ▷ Schedule in the last frame (k) of Old Mode
Input: P ▷ Current period
Input: (Q_i^O, P) ▷ Server to be removed
Output: $s_{j,k+1}^N,\ 1 \leq j \leq N,\ j \neq i$ ▷ Schedule in the first frame ($k+1$) of New Mode

1: **for** $j \leftarrow 1$ to N **do**
2: **if** $j < i$ **then**
3: $s_{j,k+1}^N \leftarrow s_{j,k}^O + P$
4: **else if** $j > i$ **then**
5: $s_{j,k+1}^N \leftarrow s_{j,k}^O + P - Q_i^O$
6: **end if**
7: **end for**

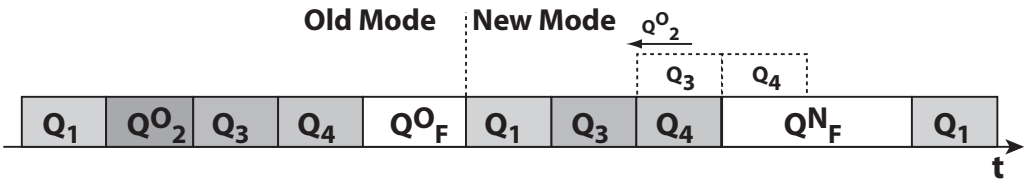

Fig. 6.11: Removing server (Q_2^O, P) from a schedule of four ASGs. The activation times of servers (Q_3^N, P) and (Q_4^N, P) have been shifted to the left by Q_2^O in New Mode, and Q_2^O has been used to increase the free budget. The dashed boxes show where servers (Q_3^O, P) and (Q_4^O, P) would have been scheduled if there were no reconfiguration

The algorithm does not change the schedule of servers (Q_1, P), ..., (Q_{i-1}, P). Then we have $\widetilde{\beta}_j = \beta_{Q_j,P}$ for $1 \leq j \leq i-1$ and condition (6.6) follows from this.

We need to show that condition (6.6) holds for the servers with shifted activation times, i.e. (Q_{i+1}, P), ..., (Q_N, P). We will do this for server (Q_{i+1}, P) but the proof is the same for all of them.

Let us consider a time interval $[l, h)$ where $h > l$ and $\Delta = h - l$. There are three cases for the position of this interval with respect to time $t = s_{i+1,k}^O + Q_{i+1}$ which is the end of the last activation of server (Q_{i+1}, P) in Old Mode. Cases are illustrated in Fig. 6.12.

Case 1: $h \leq t$**.** Up to time t, server (Q_{i+1}, P) is scheduled without changes with Old Mode parameters. Then we have $\widetilde{\beta}_{i+1} \geq \beta_{Q_{i+1},P}$.

Case 2: $t \leq l$**.** After time t, server (Q_{i+1}, P) is scheduled with New Mode parameters which are the same as for the Old Mode. By construction

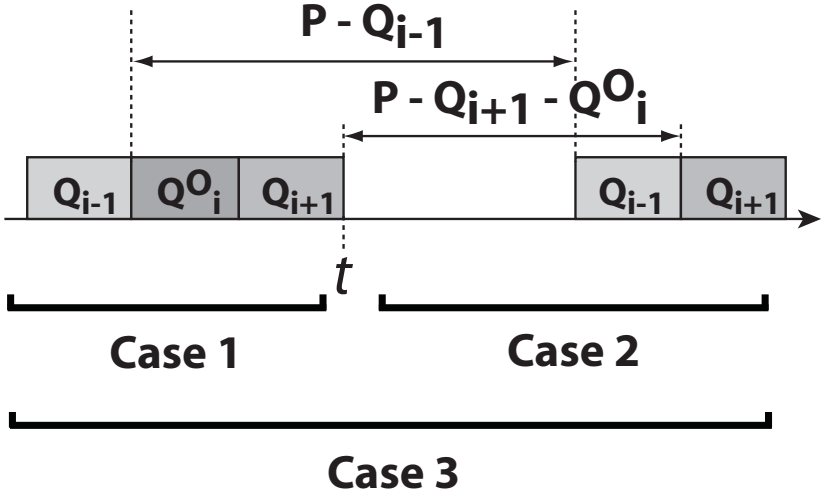

Fig. 6.12: Cases in the proof of Theorem 6.8

because of the shift of the starting time, the first activation of the server after time t comes at most after $P - Q_i^O - Q_{i+1}$ time units which is smaller than the maximum distance between the end of an activation and the start of the next activation in New Mode, $P - Q_{i+1}$. Afterwards, in the New Mode, the start of each activation is separated by P time units. Therefore we have that $\widetilde{\beta}_{i+1} \geq \beta_{Q_{i+1},P}$.

Case 3: $l < t < h$**.** Denote the service supplied by the server in interval $[l, t)$ as $C[l, t)$. We know that the end of the last activation of the server was at time t which means that in Old Mode, the next activation of the server will not happen until time $t + P - Q_{i+1}$. Then it follows that the service provided in interval $[l, t)$ can be lower bounded with the service curve for interval $[l, t+P-Q_{i+1})$. Therefore, we have that $C[l, t) \geq \beta_{Q_{i+1},P}(t+P-Q_{i+1}-l)$.

For interval $[t, h)$, let the service supplied by the server be $C[t, h)$. With the reconfiguration algorithm we have shifted earlier the activation times of server (Q_{i+1}, P) by the budget of the removed server Q_i^O, then the service supplied in interval $[t, h)$ can be lower bounded with the service curve for interval $[t - Q_i^O, h)$. Then we have $C[t, h) \geq \beta_{Q_{i+1},P}(h - t + Q_i^O)$.

Now, we can compute the service for interval $\Delta = (h - l)$ and get a lower bound for $\widetilde{\beta}_{i+1}(\Delta)$ as follows:

$$
\begin{aligned}
C[l, h) &= C[l, t) + C[t, h) \\
&\geq \beta_{Q_{i+1},P}(t + P - Q_{i+1} - l) + \beta_{Q_{i+1},P}(h - t + Q_i^O) \\
&\quad \text{Substitute: } \lambda = h - t + Q_i^O, \text{ where } 0 \leq \lambda \leq (h - l) \\
&= \beta_{Q_{i+1},P}(h - \lambda + Q_i^O + P - Q_{i+1} - l) + \beta_{Q_{i+1},P}(\lambda) \\
&= \beta_{Q_{i+1},P}(\Delta - \lambda + P - Q_{i+1} + Q_i^O) + \beta_{Q_{i+1},P}(\lambda) \\
&\geq \inf_{0 \leq \lambda \leq \Delta} \{\beta_{Q_{i+1},P}(\Delta - \lambda + P - Q_{i+1} + Q_i^O) + \beta_{Q_{i+1},P}(\lambda)\} \\
&= (\beta_{Q_{i+1},P} \otimes \beta_{Q_{i+1},P})(\Delta + P - Q_{i+1} + Q_i^O) \,, \qquad (6.9)
\end{aligned}
$$

where we use the definition of $\otimes$ from Appendix A.

Combining cases 1, 2, and 3, we get the following lower bound for $\widetilde{\beta}_{i+1}(\Delta)$:

$$
\widetilde{\beta}_{i+1}(\Delta) \geq \min\{\beta_{Q_{i+1},P}(\Delta),\ (\beta_{Q_{i+1},P} \otimes \beta_{Q_{i+1},P})(\Delta + P - Q_{i+1} + Q_i^O)\} \,.
$$

Since, the service curve $\beta_{Q_{i+1},P}$ is a wide-sense increasing function (see Appendix A), and we have that (6.9) is greater or equal to (6.7), then condition (6.6) follows.

■

6.5.2.2 Decreasing the Budget of an Existing ASG

Decreasing the budget of a server means that in Old Mode, the server has budget $Q^O > 0$, and in New Mode, its budget is $0 < Q^N < Q^O$. The

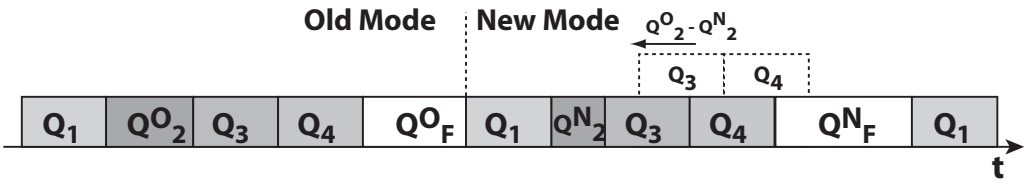

Fig. 6.13: Decreasing the budget from Q_2^O to Q_2^N in a schedule of four ASGs. The activation times of servers (Q_3, P) and (Q_4, P) have been shifted earlier in New Mode by $(Q_2^O - Q_2^N)$, and $(Q_2^O - Q_2^N)$ has been used to increase the free budget. The dashed boxes show where servers (Q_3, P) and (Q_4, P) would have been scheduled if there were no reconfiguration

budgets of all other servers are unchanged. This is an operation that can always be performed since it decreases the utilization of the system by $(Q^O - Q^N)/P$, and increases the free budget, $Q_F^N = Q_F^O + (Q^O - Q^N)$.

Algorithm 6.2 describes decreasing the budget of server (Q_i, P) from Q_i^O in Old Mode to Q_i^N in New Mode in a schedule of N servers. In the first frame when the budget is decreased, activations of all preceding servers are unchanged while activations of succeeding servers are shifted earlier by the amount of decrease of budget. This is illustrated in Fig. 6.13.

Theorem 6.9: (Service Guarantee during Decreasing the Budget of an ASG) *Decreasing the budget of a server from (Q_i^O, P) to (Q_i^N, P) in a schedule of N servers using Algorithm 6.2 satisfies condition (6.6) for all servers in the system.*

Algorithm 6.2 `Decreasing the budget of an ASG`

Input: $s^O_{j,k}$, $1 \le j \le N$ ▷ Schedule in the last frame (k) of Old Mode
Input: P ▷ Current period
Input: (Q_i^O, P) ▷ Server to be modified with Old Mode parameters
Input: (Q_i^N, P) ▷ Server to be modified with New Mode parameters
Output: $s^N_{j,k+1}$, $1 \le j \le N$ ▷ Schedule in the first frame ($k+1$) of New Mode

1: **for** $j \leftarrow 1$ to N **do**
2: **if** $j \le i$ **then**
3: $s^N_{j,k+1} \leftarrow s^O_{j,k} + P$
4: **else if** $j > i$ **then**
5: $s^N_{j,k+1} \leftarrow s^O_{j,k} + P - (Q_i^O - Q_i^N)$
6: **end if**
7: **end for**

Unchanged servers get at least a guaranteed service during the reconfiguration of $\widetilde{\beta}_j \geq \beta_{Q_j,P}$, $1 \leq j \leq N$, $j \neq i$. *For the decreased server, this is* $\widetilde{\beta}_i \geq \beta_{Q_i^N,P}$.

Proof. Similarly to removing a server, the schedule of servers $(Q_1, P), \ldots, (Q_{i-1}, P)$ does not change. Then we have that $\widetilde{\beta}_j = \beta_{Q_j,P}$ for $1 \leq j \leq i-1$ and condition (6.6) follows from this.

We need to show that condition (6.6) holds for the servers with shifted activation times $(Q_{i+1}, P), \ldots, (Q_N, P)$. For each of them we have that:

$$\widetilde{\beta}_j(\Delta) \geq \min\{\beta_{Q_j,P}(\Delta),\ (\beta_{Q_j,P} \otimes \beta_{Q_j,P})(\Delta + P - Q_j + Q_i^O - Q_i^N)\} \quad j \in [i+1, N]\ .$$

Following the same argument as for the removal of a server we can show that condition (6.6) is satisfied. The complete proof is omitted here.

We also need to show that condition (6.6) holds for the server with decreased budget (Q_i, P). Let us consider a time interval $[l, h)$ where $h > l$ and $\Delta = h - l$. There are three cases for the position of this interval with respect to time $t = s_{i,k}^O + Q_i^O$ which is the end of the last activation of server (Q_i^O, P) in the Old Mode. Cases are illustrated in Fig. 6.14.

Case 1: $h \leq t$**.** Up to time t server (Q_i^O, P) is scheduled without changes with Old Mode parameters. Then we have $\widetilde{\beta}_i \geq \beta_{Q_i^O,P}$.

Case 2: $t \leq l$**.** After time t server (Q_i^N, P) is scheduled with New Mode parameters. By construction, the first activation of the server after time t comes after $P - Q_i^O$ time units which is smaller than the maximum distance between the end of an activation and the start of the next activation in New Mode, $P - Q_i^N$. Afterwards, the distance between the starts of all

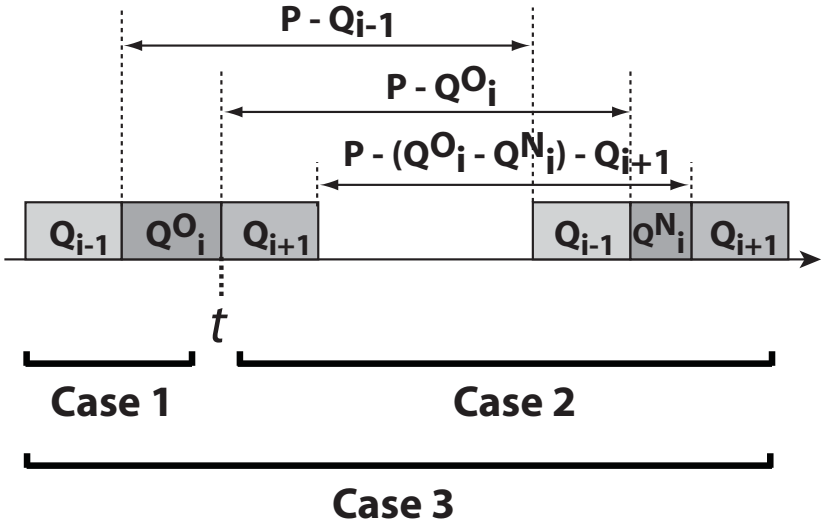

Fig. 6.14: Cases in the proof of Theorem 6.9

subsequent activations of the server is equal to P. This means that we have $\widetilde{\beta}_i \geq \beta_{Q_i^N,P}$.

Case 3: $l < t < h$**.** Server (Q_i, P) is scheduled with budget Q_i^O before time t and with budget Q_i^N after time t.

Denote the service supplied by the server in interval $[l, t)$ as $C[l, t)$. We know that the end of the last activation of the server was at time t which means that in Old Mode, the next activation of the server will not happen until time $t + P - Q_i^O$. From which it follows that the service provided in interval $[l, t)$ is lower bounded by the Old Mode service curve for interval $[l, t + P - Q_i^O)$. Therefore, we have that $C[l, t) \geq \beta_{Q_i^O,P}(t + P - Q_i^O - l)$.

Now consider interval $[t, h)$ where the service supplied is $C[t, h)$. After time t the next activation comes after $P - Q_i^O$ time units which is smaller than the one for New Mode, $P - Q_i^N$. We have a difference of $(Q_i^O - Q_i^N)$. Therefore we can bound the actual service in interval $[t, h)$ using the New Mode service curve for interval $(h - t + (Q_i^O - Q_i^N))$ as $C[t, h) \geq \beta_{Q_i^N,P}(h - t + (Q_i^O - Q_i^N))$.

Now, we can compute the service for interval $\Delta = (h - l)$ and get a lower bound for $\widetilde{\beta}_i(\Delta)$ as follows:

$$
\begin{aligned}
C[l, h) &= C[l, t) + C[t, h) \\
&\geq \beta_{Q_i^O,P}(t + P - Q_i^O - l) + \beta_{Q_i^N,P}(h - t + (Q_i^O - Q_i^N)) \\
&\quad \text{Substitute: } \lambda = h - t + (Q_i^O - Q_i^N), \text{ where } 0 \leq \lambda \leq (h - l) \\
&= \beta_{Q_i^O,P}(h - \lambda + Q_i^O - Q_i^N + P - Q_i^O - l) + \beta_{Q_i^N,P}(\lambda) \\
&= \beta_{Q_i^O,P}(\Delta - \lambda - Q_i^N + P) + \beta_{Q_i^N,P}(\lambda) \\
&\geq \inf_{0 \leq \lambda \leq \Delta} \{\beta_{Q_i^O,P}(\Delta - \lambda + P - Q_i^N) + \beta_{Q_i^N,P}(\lambda)\} \\
&= (\beta_{Q_i^O,P} \otimes \beta_{Q_i^N,P})(\Delta + P - Q_i^N) \,. \qquad (6.10)
\end{aligned}
$$

Combining cases 1, 2, and 3, we get the following lower bound for $\widetilde{\beta}_i(\Delta)$:

$$
\widetilde{\beta}_i(\Delta) \geq \min\{\beta_{Q_i^O,P}(\Delta),\ \beta_{Q_i^N,P}(\Delta),\ (\beta_{Q_i^O,P} \otimes \beta_{Q_i^N,P})(\Delta + P - Q_i^N)\}
$$

Since we have that $\beta_{Q_i^O,P}(\Delta) \geq \beta_{Q_i^N,P}(\Delta)$ for $\forall \Delta \in \mathbb{R}^{\geq 0}$ and from the isotonicity property of the min-plus convolution operator $\otimes$ (see Appendix A), it follows that (6.10) is greater or equal to (6.8), then condition (6.6) follows.

■

Algorithm 6.3 `Adding an ASG`

Input: $s^O_{j,k}$, $1 \le j \le N$ ▷ Schedule in the last frame (k) of Old Mode
Input: $s^O_{F,k}$ ▷ Start of free budget in frame (k) in Old Mode
Input: P ▷ Current period
Input: Q^O_F ▷ Free budget in Old Mode
Input: (Q^N_{N+1}, P) ▷ Server to be added in New Mode
Require: $Q^N_{N+1} \le Q^O_F$
Output: $s^N_{j,k+1}$, $1 \le j \le N+1$ ▷ Schedule in the first frame ($k+1$) of New Mode

1: **for** $j \leftarrow 1$ to N **do**
2: $\quad s^N_{j,k+1} \leftarrow s^O_{j,k} + P$
3: **end for**
4: $s^N_{N+1,k+1} \leftarrow s^O_{F,k} + P$

6.5.2.3 Adding a New ASG

Adding a server to the schedule means that in Old Mode, it has budget $Q^O = 0$, while in New Mode, its budget is $Q^N > 0$. Budgets of all other servers are unchanged. From condition (6.1), this is an operation that is feasible if there is sufficient free budget in the system:

$$Q^N \le Q^O_F \ .$$

The reconfiguration decreases the free budget in the system, $Q^N_F = Q^O_F - Q^N$, and increases the utilization by Q^N/P.

Algorithm 6.3 describes adding server (Q^N_{N+1}, P) to a schedule of N servers. In the first frame where the server is added, it is scheduled at the beginning of the free budget slot. This is illustrated in Fig. 6.15.

Theorem 6.10: (Service Guarantee during Adding an ASG) *Adding server* (Q^N_{N+1}, P) *to a schedule of* N *servers using Algorithm 6.3 satisfies condition*

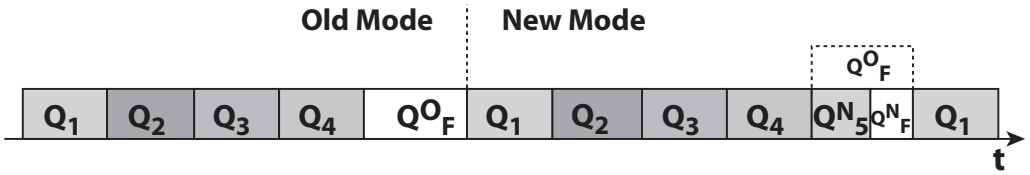

Fig. 6.15: Addition of server (Q_5, P) to a schedule of four ASGs. Activation times of existing servers do not change as the added server is scheduled after all other servers in New Mode. Free budget has been decreased by the budget of the added server, $Q^N_F = Q^O_F - Q^N_5$. The dashed box shows where the free budget Q^O_F would have been if there were no reconfiguration

(6.6) for all other servers in the system as each of them gets at least a guaranteed service during the reconfiguration of $\widetilde{\beta}_j = \beta_{Q_j,P}$, $1 \le j \le N$.

Proof. In this scenario, proving condition (6.6) is trivial, as addition of server (Q^N_{N+1}, P) does not affect the schedule of any other server in the system, i.e. $\widetilde{\beta}_j = \beta_{Q_j,P}$ for $1 \le j \le N$.

■

6.5.2.4 Increasing the Budget of an Existing ASG

Increasing the budget of a server means that in Old Mode it has budget $Q^O > 0$, and in New Mode it has budget $Q^N > Q^O$. Budgets of all other servers are unchanged. From condition (6.1), this is an operation that is feasible if there is sufficient free budget in the system:

$$Q^N - Q^O \le Q^O_F \ . \tag{6.11}$$

The reconfiguration decreases the free budget in the system, $Q^N_F = Q^O_F - (Q^N - Q^O)$, and increases the utilization of the system by $(Q^N - Q^O)/P$.

Algorithm 6.4 shows increasing the budget of a server from (Q^O_i, P) to (Q^N_i, P) in a schedule of N servers. In the first frame where the budget is increased, all preceding servers are activated earlier in the free budget of the previous frame by the amount of the increase of budget, and all succeeding servers are activated without change. This is illustrated in Fig. 6.16.

Theorem 6.11: (Service Guarantee during Increasing the Budget of an ASG) *Increasing the budget of a server from* (Q^O_i, P) *to* (Q^N_i, P) *in a schedule of* N *servers using Algorithm 6.4 satisfies condition (6.6) for all servers in the system.*

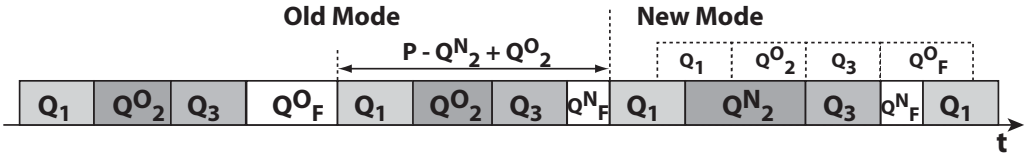

Fig. 6.16: Increasing the budget of server (Q^O_2, P) to Q^N_2 in a schedule of three ASGs. Last frame of Old Mode has a decreased length, $P - Q^N_2 + Q^O_2$. This causes the subsequent activation times of server (Q_1, P) to be shifted earlier. Activation times of server (Q_3, P) do not change as the shorter activation frame cancels with the increased budget for all New Mode activations. Free budget has been decreased by the increase of server budget, $Q^N_F = Q^O_F - Q^N_2 + Q^O_2$. The dashed boxes show where the activations of servers (Q_1, P), (Q_2, P), (Q_3, P) and the free budget Q_F would have been if there were no reconfiguration

Algorithm 6.4 `Increasing the budget of an ASG`

Input: $s^O_{j,k}$, $1 \le j \le N$ ▷ Schedule in the last frame (k) of Old Mode
Input: P ▷ Current period
Input: Q^O_F ▷ Free budget in Old Mode
Input: (Q^O_i, P) ▷ Server to be modified with Old Mode parameters
Input: (Q^N_i, P) ▷ Server to be modified with New Mode parameters
Require: $Q^N_i - Q^O_i \le Q^O_F$
Output: $s^N_{j,k+1}$, $1 \le j \le N$ ▷ Schedule in the first frame ($k+1$) of New Mode

1: **for** $j \leftarrow 1$ to N **do**
2: **if** $j \le i$ **then**
3: $s^N_{j,k+1} \leftarrow s^O_{j,k} + P - (Q^N_i - Q^O_i)$
4: **else if** $j > i$ **then**
5: $s^N_{j,k+1} \leftarrow s^O_{j,k} + P$
6: **end if**
7: **end for**

Unchanged servers get at least a guaranteed service during the reconfiguration of $\widetilde{\beta}_j \ge \beta_{Q_j,P}$, $1 \le j \le N$, $j \ne i$. *For the increased server this is* $\widetilde{\beta}_i \ge \beta_{Q^O_i,P}$.

Proof. Showing schedulability for servers (Q_{i+1}, P), ..., (Q_N, P) is trivial as their schedule is not affected by the algorithm, i.e. $\widetilde{\beta}_j = \beta_{Q_j,P}$ for $i+1 \le j \le N$.

The shifting of activation times for servers (Q_1, P), ..., (Q_{i-1}, P) is exactly the same operation as in the scenario of removing a server, therefore the proof is similar to the one for Theorem 6.8 and will be omitted here. The service provided by each of them in the transition is lower bounded by:

$$\widetilde{\beta}_j(\Delta) \ge \min\{\beta_{Q_j,P}(\Delta),\ (\beta_{Q_j,P} \otimes \beta_{Q_j,P})(\Delta + P - Q_j + Q^N_i - Q^O_i)\} \quad j \in [1, i-1]\ ,$$

which meets condition (6.6).

We need to show the schedulability for the server with increased budget (Q_i, P). Let us consider a time interval $[l, h)$ where $h > l$ and $\Delta = h - l$. There are three cases for the position of this interval with respect to time $t = s^O_{i,k} + Q^O_i$ which is the end of the last activation of server (Q^O_i, P) in the Old Mode. Cases are illustrated in Fig. 6.17.

Case 1: $h \le t$**.** Up to time t server (Q^O_i, P) is scheduled without changes using only Old Mode parameters. This implies that $\widetilde{\beta}_i \ge \beta_{Q^O_i,P}$.

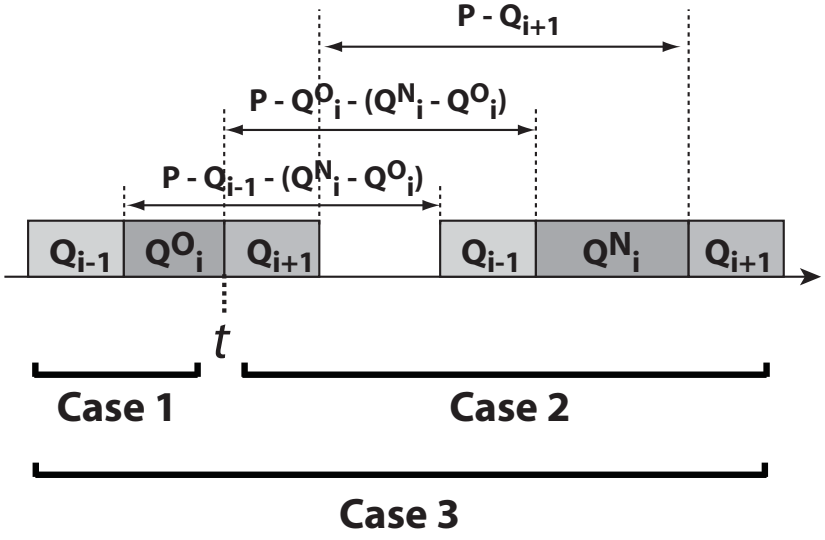

Fig. 6.17: Cases in the proof of Theorem 6.11

Case 2: $t \leq l$**.** After time t server (Q_i^N, P) is scheduled with New Mode parameters. By construction, the first activation of the server after time t comes after $P - Q_i^N$ time units which equals the maximum distance between the end of an activation and the start of the next activation in the New Mode, $P - Q_i^N$. Afterwards, the distance between the starts of all subsequent activations of the server is equal to P. This means that $\widetilde{\beta}_i(\Delta) \geq \beta_{Q_i^N,P}(\Delta)$.

Case 3: $l < t < h$**.** Server (Q_i, P) is scheduled with budget Q_i^O before time t and with budget Q_i^N after time t.

Denote the service supplied by the server in interval $[l, t)$ as $C[l, t)$. We know that the end of the last activation of the server was at time t which means that in Old Mode, the next activation of the server will not happen until time $t + P - Q_i^O$. Then the service provided in interval $[l, t)$ is lower bounded by the Old Mode service curve for interval $[l, t + P - Q_i^O)$. Therefore, we have that $C[l, t) \geq \beta_{Q_i^O,P}(t + P - Q_i^O - l)$.

Now consider interval $[t, h)$ where the service supplied is $C[t, h)$. After time t, the next activation of the server comes after $P - Q_i^N$ time units which equals the maximum distance between the end of an activation and the start of the next activation in the New Mode. Afterwards, the distance between the starts of all subsequent activations of the server is equal to P. Therefore we can bound the actual service in interval $[t, h)$ using the New Mode service curve as $C[t, h) \geq \beta_{Q_i^N,P}(h - t)$.

Now, we can compute the service for interval $\Delta = (h - l)$ and get a lower bound for $\widetilde{\beta}_i(\Delta)$ as follows:

$$\begin{aligned}
C[l,h) &= C[l,t) + C[t,h) \\
&\geq \beta_{Q_i^O,P}(t + P - Q_i^O - l) + \beta_{Q_i^N,P}(h - t) \\
&\quad \text{Substitute: } \lambda = h - t, \text{ where } 0 \leq \lambda \leq (h - l) \\
&= \beta_{Q_i^O,P}(h - \lambda + P - Q_i^O - l) + \beta_{Q_i^N,P}(\lambda) \\
&= \beta_{Q_i^O,P}(\Delta - \lambda + P - Q_i^O) + \beta_{Q_i^N,P}(\lambda) \\
&\geq \inf_{0 \leq \lambda \leq \Delta} \{\beta_{Q_i^O,P}(\Delta - \lambda + P - Q_i^O) + \beta_{Q_i^N,P}(\lambda)\} \\
&= (\beta_{Q_i^O,P} \otimes \beta_{Q_i^N,P})(\Delta + P - Q_i^O) \,. \qquad (6.12)
\end{aligned}$$

Combining cases 1, 2, and 3, we get the following lower bound for $\widetilde{\beta}_i(\Delta)$:

$$\widetilde{\beta}_i(\Delta) \geq \min\{\beta_{Q_i^O,P}(\Delta),\ \beta_{Q_i^N,P}(\Delta),\ (\beta_{Q_i^O,P} \otimes \beta_{Q_i^N,P})(\Delta + P - Q_i^O)\} \,.$$

Since we have that $\beta_{Q_i^N,P}(\Delta) \geq \beta_{Q_i^O,P}(\Delta)$ for $\forall \Delta \in \mathbb{R}^{\geq 0}$ and because of the isotonicity of the min-plus convolution operator $\otimes$ (see Appendix A), we have that (6.12) is greater or equal to (6.7), then condition (6.6) follows.

■

6.5.3 Change of Period

We perform analysis given the configurations of the system (such as budgets and periods) in Old and New Modes. The results of the analysis are whether a transition is feasible with the given configurations, and in the case of feasibility with what parameters for the Reconfiguration phase it can be executed online.

6.5.3.1 Increase of Period

We suppose that there are N servers in the system. In the Old Mode, they operate with parameters (Q_i^O, P^O), $1 \leq i \leq N$, and in the New Mode with (Q_i^N, P^N), $1 \leq i \leq N$, where $P^O < P^N$. Assume that for every server we have that $Q_i^O \leq Q_i^N$. If this is not the case, namely there is a server that requires a smaller budget in the bigger period, $Q_i^O > Q_i^N$, we can reduce its budget first by using the algorithms proposed in Section 6.5.2 as we can be sure that schedulability will be satisfied with the new budget in the old (smaller) period, and then perform the reconfiguration involving increase of period.

The proposed reconfiguration algorithm is subject to the feasibility condition that the sum of all New Mode server budgets is smaller than the Old Mode period which is expressed as follows:

$$\sum_{i=1}^{N} Q_i^N \leq P^O \ . \tag{6.13}$$

The condition ensures that the increase of budgets does not lead to service guarantee violations in intervals of time beginning P^O time units before the reconfiguration and ending P^O time units after the reconfiguration. It can be related to the feasibility condition (6.11) from Section 6.5.2.4, $\sum_{i=1}^{N}(Q_i^N - Q_i^O) \leq Q_F^O$.

Assume that *condition (6.13)* is satisfied. Consider the last server in the schedule, (Q_N^O, P^O) and suppose that the reconfiguration starts immediately when it finishes executing. In the first activation frame when the budgets of all servers are increased, the last server will not receive any service for at most $\sum_{i=1}^{N-1} Q_i^N$ time units. Just before this waiting time however, the server was scheduled with budget Q_N^O. Then the server will receive a budget of Q_N^O in a time interval of $P^O + \sum_{i=1}^{N-1} Q_i^N$. From (6.13), we have that $\sum_{i=1}^{N-1} Q_i^N \leq P^O - Q_N^O$, and this will satisfy the Old Mode service guarantee $\beta_{Q_N^O, P^O}$. Similarly, the New Mode service guarantee $\beta_{Q_N^N, P^N}$ is satisfied as the waiting time before the server is given budget of Q_N^N is $\sum_{i=1}^{N-1} Q_i^N$ time units which is upper bounded by $P^N - Q_N^N$. We have shown that condition (6.13) is a sufficient condition which ensures that condition (6.6) holds for the time interval beginning P^O time units before the reconfiguration and ending P^O time units after the reconfiguration. However, to do this for bigger time intervals we need to develop a more involved reconfiguration algorithm and analysis.

When condition (6.13) is not satisfied, i.e. staying in the most pessimistic configuration is not feasible, the reconfiguration algorithm would need to go through *one or more intermediate* modes (budgets and periods) where for each successive pair of them condition (6.13) holds. We will not discuss this further and assume that the feasibility condition is met.

The algorithm for performing safely the increase of period can be summarized in three steps: (1) Increase to New Mode budgets following Algorithm 6.4. (2) Schedule the ASG servers for $K \geq 1$ activation frames using the New Mode budgets and Old Mode period. (3) Increase to New Mode period by increasing free budget. The second step of the algorithm we denote as the Reconfiguration phase which is K activation frames long. For simplicity of the presentation, we suppose that it has the Old Mode period but it can actually have a shorter period which would require a

Algorithm 6.5 `Increase of Period`

Input: $s^O_{j,k}$, $1 \le j \le N$ ▷ Schedule in the last frame (k) of Old Mode
Input: P^O ▷ Old Mode period
Input: P^N ▷ New Mode period
Input: (Q^O_i, P^O), $1 \le i \le N$ ▷ Servers in Old Mode
Input: (Q^N_i, P^N), $1 \le i \le N$ ▷ Servers in New Mode
Input: K ▷ Number of activation frames during the Reconfiguration
Require: $\sum_{i=1}^{N} Q^N_i \le P^O$
Output: $s^N_{j,k+p}$, $1 \le j \le N$, $1 \le p \le K$ ▷ Schedule in all frames during the Reconfiguration
Output: $s^N_{j,k+K+1}$, $1 \le j \le N$ ▷ Schedule in the first frame ($k + K + 1$) of New Mode

(* First frame of Reconfiguration - increase budgets *)
1: $s^R_{1,k+1} \leftarrow s^O_{1,k} + P^O - \sum_{i=1}^{N}(Q^N_i - Q^O_i)$
2: **for** $j \leftarrow 2$ to N **do**
3: $\quad s^R_{j,k+1} \leftarrow s^R_{j-1,k+1} + Q^N_{j-1}$
4: **end for**

(* All subsequent frames of Reconfiguration *)
5: **for** $p \leftarrow 2$ to K **do**
6: $\quad$ **for** $j \leftarrow 1$ to N **do**
7: $\quad\quad s^R_{j,k+p} \leftarrow s^R_{j,k+p-1} + P^O$
8: $\quad$ **end for**
9: **end for**

(* First frame of New Mode - increase period *)
10: **for** $j \leftarrow 1$ to N **do**
11: $\quad s^N_{j,k+K+1} \leftarrow s^R_{j,k+K} + P^N$
12: **end for**

small modification in the analysis. At the moment we assume that K is given as input to the algorithm, later we will show how to compute it. Algorithm 6.5 describes the details for performing the increase of period. It is illustrated in Fig. 6.18.

The following theorem gives a lower bound for the guaranteed resource supply of an ASG server during an increase of period reconfiguration.

Theorem 6.12: (Service Guarantee during Increasing the Period of the ASGs) *Reconfiguring a server from* (Q^O_i, P^O) *to* (Q^N_i, P^N) *in a schedule of* N

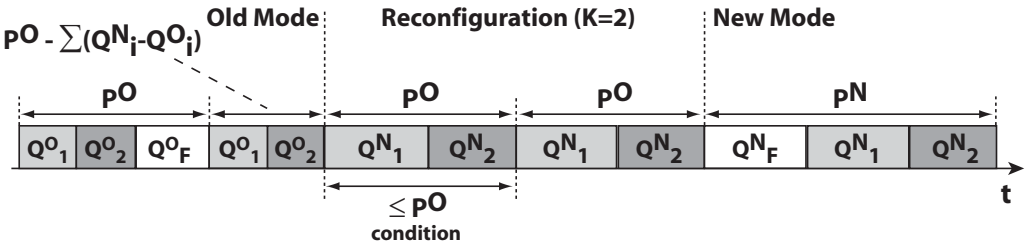

Fig. 6.18: Increase of period with $K = 2$

servers using Algorithm 6.5 provides at least a guaranteed service of[1]*:*

$$\widetilde{\beta}_i(\Delta) = \min\Big\{\beta_{Q_i^O,P^O}(\Delta),\ \beta_{Q_i^N,P^N}(\Delta),$$
$$(\beta_{Q_i^O,P^O} \otimes \beta_{Q_i^N,P^N})(\Delta - K \cdot P^O + P^O - Q_i^O) \tag{6.14}$$
$$+ \beta_{Q_i^N,P^O}(K \cdot P^O)\Big\}\,, \tag{6.15}$$

which satisfies condition (6.6) when $K \geq 1$ is found as:

$$K = \max_{1 \leq i \leq N} \Big\{ \min\Big\{\kappa \mid \forall \Delta \in \mathbb{R}^{\geq 0},\ \kappa \in \mathbb{Z}^+,$$
$$(\beta_{Q_i^O,P^O} \otimes \beta_{Q_i^N,P^N})(\Delta - \kappa \cdot P^O + P^O - Q_i^O) + \beta_{Q_i^N,P^O}(\kappa \cdot P^O)$$
$$\geq \min\{\beta_{Q_i^O,P^O}(\Delta),\ \beta_{Q_i^N,P^N}(\Delta)\}\Big\}\Big\}\,.$$

The guaranteed service in the above theorem can be explained informally as follows. It is computed as the minimum of the services from Old Mode, New Mode, and an expression which describes the service in time intervals that span Old Mode, Reconfiguration, and New Mode. The last one consists of two subexpressions. Expression (6.14) lower bounds the service guaranteed in the time window part that is *outside* of the Reconfiguration time window and hence the service curve depends only on the Old and the New Modes parameters, it is 'shifted to the right' by the size of the Reconfiguration time window which is at most $K \cdot P^O$ time units. Expression (6.15) lower bounds the service guaranteed only in the Reconfiguration time window which uses New Mode budgets with Old Mode period, and the service is defined for a fixed length interval of size $K \cdot P^O$.

In expressions (6.14) and (6.15), we can increase the size of the Reconfiguration phase by increasing the number of activation frames in it K. In order to meet condition (6.6) for each server, we have to find the *minimum* K that will make the guaranteed service $\widetilde{\beta}_i$ greater or equal to the minimum of the Old and New Modes services. After doing this

[1]See Appendix A for the definition of the min-plus convolution operator $\otimes$.

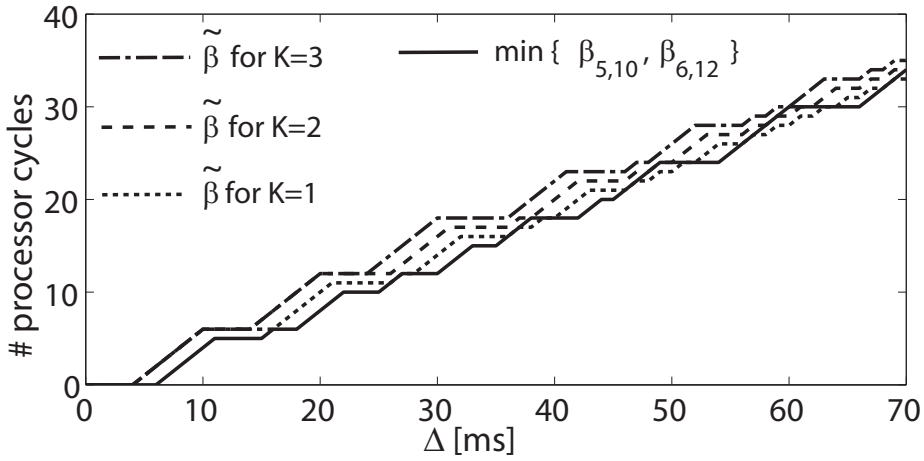

Fig. 6.19: Effect of $K = \{1,2,3\}$ for server S_B from Example 6.1. Only $K = 3$ is feasible

for all servers, we have to take the *maximum* K which will make the reconfiguration feasible for all of them.

We can find the minimum K for a server efficiently by starting with an initial value of $K = 1$. If this is not feasible, we choose successive values of K by using a binary search strategy until the smallest one is found that is feasible. With bigger K we are increasing the service guaranteed in the Reconfiguration which is service greater than Old Mode and New Mode services (it has the larger New Mode budget and the smaller Old Mode period), therefore we are guaranteed to find a finite value for K which will make condition (6.6) satisfied.

Example 6.13: *We can illustrate this argument by considering server S_B from Example 6.1. It will need $K = 3$ to perform a safe reconfiguration from* $(5,10)$ *to* $(6,12)$. *This is illustrated in Fig. 6.19 as well as the violations of condition (6.6) for $K = \{1,2\}$. The trace illustrating the violation for $K = 2$ for server S_B is shown in Fig. 6.20.*

Proof. Let us analyze the service guaranteed by an ASG server (Q_i, P) during the reconfiguration performed with Algorithm 6.5. The Reconfiguration phase is K frames long.

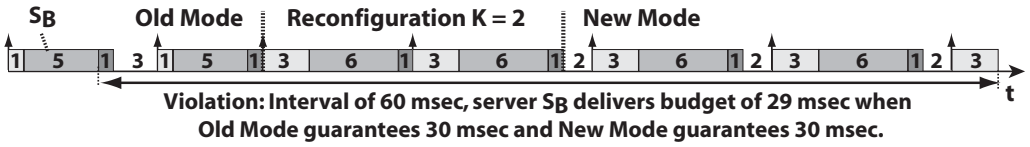

Fig. 6.20: Violation of condition (6.6) when increasing period with $K = 2$ for server S_B from Example 6.1

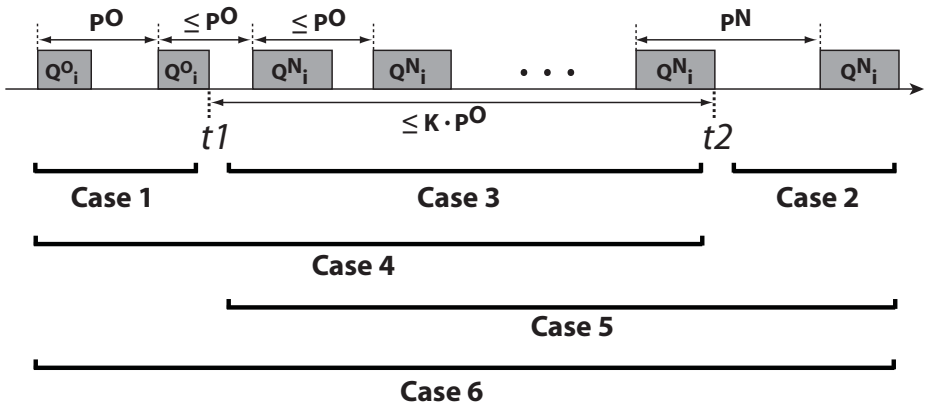

Fig. 6.21: Cases for the proof of Theorem 6.12

Consider a time interval $[l, h)$ where $h > l$ and $\Delta = h - l$. There are six cases for the position of this interval with respect to time $t_1 = s^O_{i,k} + Q^O_i$ which is the end of the last activation of the server in the Old Mode, and time $t_2 = s^R_{i,k+K} + Q^N_i$ which is the end of the last activation of the server in the Reconfiguration. Cases are illustrated in Fig. 6.21.

Case 1: $h \leq t_1$**.** Up to time t_1 server (Q^O_i, P^O) is scheduled without changes using only Old Mode parameters. This implies that $\widetilde{\beta_i} \geq \beta_{Q^O_i, P^O}$.

Case 2: $t_2 \leq l$**.** After time t_2 server (Q^N_i, P^N) is scheduled with New Mode parameters. By construction, the first activation of the server after time t_2 comes at most after $P^N - Q^N_i$ time units which equals the maximum distance between the end of an activation and the start of the next activation in the New Mode. Afterwards, the distance between the starts of all subsequent activations of the server is equal to P^N. This means that $\widetilde{\beta_i} \geq \beta_{Q^N_i, P^N}$.

Case 3: $t_1 \leq l < h \leq t_2$**.** Between times t_1 and t_2, the server is scheduled with New Mode budget Q^N_i during a period $P^R = \sum_{j=1}^{N} Q^N_j$ which is upper bounded by P^O from feasibility condition (6.13). Therefore the service guaranteed by the ASG server is $\widetilde{\beta_i} \geq \beta_{Q^N_i, P^O}$ which is greater or equal to $\beta_{Q^O_i, P^O}$ as we have that $Q^N_i \geq Q^O_i$, and greater or equal to $\beta_{Q^N_i, P^N}$ as we have that $P^O < P^N$ (all functions are wide-sense increasing, see Appendix A).

Case 4: $l < t_1 < h \leq t_2$**.** Consider the service provided in interval $[l, t_1)$ and denote it as $C[l, t_1)$. Before time t_1 the server is scheduled with Old Mode budget and period. Since t_1 is the end of the last activation, the server should not be scheduled for $P^O - Q^O_i$ time units. Therefore we can bound the service provided in interval $[l, t_1)$ with the Old Mode service curve for interval $[l, t_1 + P^O - Q^O_i)$. Therefore, we have that $C[l, t_1) \geq \beta_{Q^O_i, P^O}(t_1 + P^O - Q^O_i - l)$.

Consider the service provided in interval $[t_1, h)$ and denote it as $C[t_1, h)$. The server is activated after at most $P^O - Q_i^N$ time units with New Mode budget Q_i^N, and then repeatedly every P^O time units. Therefore we have a lower bound for the service here with the service curve $\beta_{Q^N,P^O}(h - t_1)$.

Now for the service guaranteed in this case $\widetilde{\beta_i}(h - l)$ we get a lower bound as follows:

$$\begin{aligned}
C[l,h) &= C[l,t_1) + C[t_1,h) \\
&\geq \beta_{Q_i^O,P^O}(t_1 + P^O - Q_i^O - l) + \beta_{Q_i^N,P^O}(h - t_1) \\
&\quad \text{Substitute: } \lambda = h - t_1, \text{ where } 0 \leq \lambda \leq (h - l) \\
&= \beta_{Q_i^O,P^O}(h - \lambda + P^O - Q_i^O - l) + \beta_{Q_i^N,P^O}(\lambda) \\
&= \beta_{Q_i^O,P^O}(\Delta - \lambda + P^O - Q_i^O) + \beta_{Q_i^N,P^O}(\lambda) \\
&\geq \inf_{0\leq\lambda\leq\Delta}\{\beta_{Q_i^O,P^O}(\Delta - \lambda + P^O - Q_i^O) + \beta_{Q_i^N,P^O}(\lambda)\} \\
&= (\beta_{Q_i^O,P^O} \otimes \beta_{Q_i^N,P^O})(\Delta + P^O - Q_i^O)
\end{aligned}$$

Case 5: $t_1 \leq l < t_2 < h$**.** Consider the service provided in interval $[l, t_2)$ and denote it as $C[l, t_2)$. The server is activated after at most $P^O - Q_i^N$ time units with New Mode budget and then repeatedly every P^O time units. Since t_2 is the end of the last activation of the server during the Reconfiguration, the server should not receive any service for $P^O - Q_i^N$ time units. Therefore we have that the service provided in interval $[l, t_2)$ can be lower bounded with a service curve for interval $[l, t_2 + P^O - Q_i^N)$ which is $\beta_{Q^N,P^O}(t_2 + P^O - Q_i^N - l)$.

Consider the service provided in interval $[t_2, h)$ and denote it as $C[t_2, h)$. The server is activated after at most $P^N - Q_i^N$ time units, and afterwards every P^N time units. It is activated with New Mode budget and period, therefore the actual service is lower bounded by $\beta_{Q^N,P^N}(h - t_2)$.

Now for the service guaranteed in this case $\widetilde{\beta_i}(h - l)$ we get a lower bound as follows:

$$\begin{aligned}
C[l,h) &= C[l,t_2) + C[t_2,h) \\
&\geq \beta_{Q_i^N,P^O}(t_2 + P^O - Q_i^N - l) + \beta_{Q_i^N,P^N}(h - t_2) \\
&\quad \text{Substitute: } \lambda = h - t_2, \text{ where } 0 \leq \lambda \leq (h - l) \\
&= \beta_{Q_i^N,P^O}(h - \lambda + P^O - Q_i^N - l) + \beta_{Q_i^N,P^N}(\lambda) \\
&= \beta_{Q_i^N,P^O}(\Delta - \lambda + P^O - Q_i^N) + \beta_{Q_i^N,P^N}(\lambda) \\
&\geq \inf_{0\leq\lambda\leq\Delta}\{\beta_{Q_i^N,P^O}(\Delta - \lambda + P^O - Q_i^N) + \beta_{Q_i^N,P^N}(\lambda)\} \\
&= (\beta_{Q_i^N,P^O} \otimes \beta_{Q_i^N,P^N})(\Delta + P^O - Q_i^N)
\end{aligned}$$

Case 6: $l < t_1 < t_2 < h$**.** Consider the service provided in interval $[l, t_1)$ and denote it as $C[l, t_1)$. Since t_1 is the end of the last activation, the server

will not get any service in interval of $P^O - Q_i^O$ time units. Therefore we can bound the service in interval $[l, t_1)$ with the Old Mode service curve for interval $[l, t_1 + P^O - Q_i^O)$. Then we have that $C[l, t_1) \geq \beta_{Q_i^O, P^O}(t_1 + P^O - Q_i^O - l)$.

Consider interval $[t_1, t_2)$. The length of the interval is upper bounded by $K \cdot P^O$. Let the service provided in this interval be denoted as $C[t_1, t_2)$. As the server is provided New Mode budget Q_i^N during Old Mode period P^O, the service is lower bounded by $\beta_{Q_i^N, P^O}(t_2 - t_1) = \beta_{Q_i^N, P^O}(K \cdot P^O)$.

Consider the service provided in interval $[t_2, h)$ and denote it as $C[t_2, h)$. The server is activated after at most $P^N - Q_i^N$ time units, and then repeatedly every P^N time units. It is activated with New Mode budget and period, therefore the service is lower bounded by $\beta_{Q^N, P^N}(h - t_2)$.

Now for the service guaranteed in this case $\widetilde{\beta}_i(h - l)$ we get a lower bound as follows:

$$
\begin{aligned}
C[l, h) &= C[l, t_1) + C[t_1, t_2) + C[t_2, h) \\
&\geq \beta_{Q_i^O, P^O}(t_1 + P^O - Q_i^O - l) + \beta_{Q_i^N, P^O}(K \cdot P^O) + \beta_{Q_i^N, P^N}(h - t_2) \\
&\quad \text{Substitute: } \lambda = h - t_2 = h - t_1 - K \cdot P^O \\
&\quad \text{where } 0 \leq \lambda \leq (h - l) \\
&= \beta_{Q_i^O, P^O}(h - \lambda - K \cdot P^O + P^O - Q_i^O - l) + \beta_{Q_i^N, P^N}(\lambda) + \beta_{Q_i^N, P^O}(K \cdot P^O) \\
&= \beta_{Q_i^O, P^O}(\Delta - \lambda - K \cdot P^O + P^O - Q_i^O) + \beta_{Q_i^N, P^N}(\lambda) + \beta_{Q_i^N, P^O}(K \cdot P^O) \\
&\geq \inf_{0 \leq \lambda \leq \Delta} \{\beta_{Q_i^O, P^O}(\Delta - \lambda - K \cdot P^O + P^O - Q_i^O) + \beta_{Q_i^N, P^N}(\lambda)\} + \beta_{Q_i^N, P^O}(K \cdot P^O) \\
&= (\beta_{Q_i^O, P^O} \otimes \beta_{Q_i^N, P^N})(\Delta - K \cdot P^O + P^O - Q_i^O) + \beta_{Q_i^N, P^O}(K \cdot P^O)
\end{aligned}
$$

Combining cases 1, 2, 3, 4, 5, and 6, we get as a lower bound for $\widetilde{\beta}_i(\Delta)$ the following expression:

$$
\begin{aligned}
\widetilde{\beta}_i(\Delta) \geq \min \Big\{ & \beta_{Q_i^O, P^O}(\Delta), && (6.16) \\
& \beta_{Q_i^N, P^N}(\Delta), && (6.17) \\
& \beta_{Q_i^N, P^O}(\Delta), && (6.18) \\
& (\beta_{Q_i^O, P^O} \otimes \beta_{Q_i^N, P^O})(\Delta + P^O - Q_i^O), && (6.19) \\
& (\beta_{Q_i^N, P^O} \otimes \beta_{Q_i^N, P^N})(\Delta + P^O - Q_i^N), && (6.20) \\
& (\beta_{Q_i^O, P^O} \otimes \beta_{Q_i^N, P^N})(\Delta - K \cdot P^O + P^O - Q_i^O) + \beta_{Q_i^N, P^O}(K \cdot P^O) \Big\} && (6.21)
\end{aligned}
$$

For expressions (6.16) and (6.17), which correspond to cases 1 and 2, respectively, condition (6.6) holds trivially.

For expression (6.18) corresponding to case 3, we have that $\beta_{Q_i^N, P^O}(\Delta) \geq \beta_{Q_i^O, P^O}(\Delta)$ for $\forall \Delta \in \mathbb{R}^{\geq 0}$, as we have $Q_i^N \geq Q_i^O$, and $\beta_{Q_i^N, P^O}(\Delta) \geq \beta_{Q_i^N, P^N}(\Delta)$ for

$\forall\Delta \in \mathbb{R}^{\geq 0}$, as we have $P^O < P^N$ (all functions are wide-sense increasing, see Appendix A). From these, condition (6.6) follows.

For expression (6.19) corresponding to case 4, we have that $\beta_{Q_i^N,P^O}(\Delta) \geq \beta_{Q_i^O,P^O}(\Delta)$ for $\forall\Delta \in \mathbb{R}^{\geq 0}$, as we have $Q_i^N \geq Q_i^O$. From this and the isotonicity of the min-plus convolution (see Appendix A), it follows that (6.19) is greater or equal to (6.7), then condition (6.6) follows.

For expression (6.20) corresponding to case 5, we have that $\beta_{Q_i^N,P^O}(\Delta + P^O - Q_i^N) \geq \beta_{Q_i^N,P^N}(\Delta + P^N - Q_i^N)$ for $\forall\Delta \in \mathbb{R}^{\geq 0}$ as $P^O < P^N$. From this and the isotonicity of the min-plus convolution operator (see Appendix A), we have that (6.20) is greater or equal to (6.8), then condition (6.6) follows.

For expression (6.21) corresponding to case 6, we cannot prove that it meets condition (6.6) because this depends on parameter K which is the length of the Reconfiguration phase. Therefore we have to find a sufficiently large K that will make (6.21) greater or equal to (6.6). We can be sure that such a K exists as for the service in the Reconfiguration phase we know that $\beta_{Q_i^N,P^O}(\Delta) \geq \beta_{Q_i^O,P^O}(\Delta)$ for $\forall\Delta \in \mathbb{R}^{\geq 0}$, as we have $Q_i^N \geq Q_i^O$, and $\beta_{Q_i^N,P^O}(\Delta) \geq \beta_{Q_i^N,P^N}(\Delta)$ for $\forall\Delta \in \mathbb{R}^{\geq 0}$, as we have $P^O < P^N$, i.e. with increasing K we are providing a service that is larger than both of the services in the Old and the New Modes.

Therefore, K is found as:

$$K = \min\Big\{\kappa \mid \forall\Delta \in \mathbb{R}^{\geq 0},\ \kappa \in \mathbb{Z}^+,\ (\beta_{Q_i^O,P^O} \otimes \beta_{Q_i^N,P^N})(\Delta - \kappa \cdot P^O + P^O - Q_i^O) + \beta_{Q_i^N,P^O}(\kappa \cdot P^O) \geq \min\{\beta_{Q_i^O,P^O}(\Delta),\ \beta_{Q_i^N,P^N}(\Delta)\}\Big\}.$$

■

6.5.3.2 Decrease of Period

This scenario is very similar to the one for increasing the period. In the Old Mode servers operate with parameters (Q_i^O, P^O), $1 \leq i \leq N$, and in the New Mode with (Q_i^N, P^N), $1 \leq i \leq N$, where $P^O > P^N$. We assume that for each server we have that $Q_i^O \geq Q_i^N$.

It is subject to the feasibility condition that the sum of Old Mode budgets is smaller than the New Mode period which is expressed as:

$$\sum_{i=1}^{N} Q_i^O \leq P^N .$$

The algorithm can be summarized in three steps: (1) Decrease to New Mode period by decreasing free budget. (2) Schedule the ASG servers for $K \geq 1$ activation frames using Old Mode budgets and New Mode period.

Algorithm 6.6 `Decrease of Period`

Input: $s^O_{j,k}$, $1 \le j \le N$ ▷ Schedule in the last frame (k) of Old Mode
Input: P^O ▷ Old Mode period
Input: P^N ▷ New Mode period
Input: (Q^O_i, P^O), $1 \le i \le N$ ▷ Servers in Old Mode
Input: (Q^N_i, P^N), $1 \le i \le N$ ▷ Servers in New Mode
Input: K ▷ Number of activation frames during the Reconfiguration
Require: $\sum_{i=1}^{N} Q^O_i \le P^N$
Output: $s^N_{j,k+p}$, $1 \le j \le N$, $1 \le p \le K$ ▷ Schedule in all frames during the Reconfiguration
Output: $s^N_{j,k+K+1}$, $1 \le j \le N$ ▷ Schedule in the first frame ($k + K + 1$) of New Mode

(* First frame of Reconfiguration - with decreased period *)
1: **for** $j \leftarrow 1$ to N **do**
2: $\quad s^R_{j,k+1} \leftarrow s^O_{j,k} + P^O$
3: **end for**

(* All subsequent frames of Reconfiguration *)
4: **for** $p \leftarrow 2$ to K **do**
5: $\quad$ **for** $j \leftarrow 1$ to N **do**
6: $\quad\quad s^R_{j,k+p} \leftarrow s^R_{j,k+p-1} + P^N$
7: $\quad$ **end for**
8: **end for**

(* First frame of New Mode - decrease budgets *)
9: $s^N_{1,k+K+1} \leftarrow s^R_{1,k+K} + P^N$
10: **for** $j \leftarrow 2$ to N **do**
11: $\quad s^N_{j,k+K+1} \leftarrow s^N_{j-1,k+K+1} + Q^N_{j-1}$
12: **end for**

(3) Decrease budgets by using Algorithm 6.2. Algorithm 6.6 describes the details for performing the decrease of period. It is illustrated in Fig. 6.22.

Theorem 6.14: (Service Guarantee during Decreasing the Period of the ASGs) *Reconfiguring a server from* (Q^O_i, P^O) *to* (Q^N_i, P^N) *in a schedule of* N *servers using Algorithm 6.6 provides at least a guaranteed service of:*

$$\widetilde{\beta}_i(\Delta) = \min\Big\{\beta_{Q^O_i,P^O}(\Delta),\ \beta_{Q^N_i,P^N}(\Delta),\ (\beta_{Q^O_i,P^O} \otimes \beta_{Q^N_i,P^N})(\Delta - K \cdot P^N + P^N - Q^N_i) + \beta_{Q^O_i,P^N}(K \cdot P^N)\Big\},$$

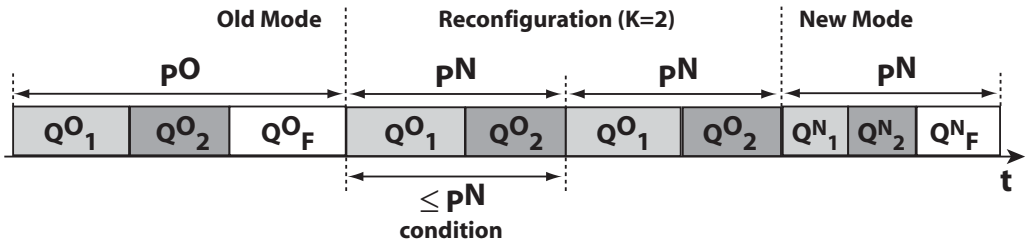

Fig. 6.22: Decrease of period with $K = 2$

which satisfies condition (6.6) when $K \geq 1$ *is found as:*

$$K = \max_{1 \leq i \leq N} \Big\{ \min\Big\{ \kappa \mid \forall \Delta \in \mathbb{R}^{\geq 0},\ \kappa \in \mathbb{Z}^{+},\ (\beta_{Q_i^O, P^O} \otimes \beta_{Q_i^N, P^N})(\Delta - \kappa \cdot P^N + P^N - Q_i^N) + \beta_{Q_i^O, P^N}(\kappa \cdot P^N) \geq \min\{\beta_{Q_i^O, P^O}(\Delta),\ \beta_{Q_i^N, P^N}(\Delta)\} \Big\} \Big\} .$$

Proof. Proof is analogous to the one for Theorem 6.12.

■

6.6 Case Study

Here, we consider a multi-mode real-time system that executes two applications. Application 1 can run in two modes denoted as mode 1 and mode 2. In mode 1, there is a single task which processes a single event stream described with a period $p = 5\,\text{ms}$, jitter $j = 10\,\text{ms}$, and minimum interarrival time between two events $d = 1\,\text{ms}$. Each event has a worst-case execution time of $c = 2\,\text{ms}$, and it needs to be processed within a relative deadline of $D = 9\,\text{ms}$. Similarly, in mode 2 there is a single task but it processes an event stream with parameters $p = 40\,\text{ms}$, $j = 20\,\text{ms}$, $d = 20\,\text{ms}$, $c = 7\,\text{ms}$, and $D = 25\,\text{ms}$. Application 2 is a single mode application, it has a single task that processes one event stream with parameters $p = 20\,\text{ms}$, $j = 15\,\text{ms}$, $d = 5\,\text{ms}$, $c = 1\,\text{ms}$, and $D = 30\,\text{ms}$. The system schedules the two applications using two servers (Q_1, P) and (Q_2, P). We suppose that each context switch takes 0.3 ms. The utilization of the system, U, can be computed as $U = (Q_1 + 0.3 + Q_2 + 0.3)/P$.

The designer of this system needs to select the configuration parameters of the ASGs such as the *minimum* required budgets that make the two applications schedulable, and the size of the period for the two servers. The design objective is to minimize utilization because other soft real-time applications use the unused resources while guaranteeing the real-time requirements. Then the solution depends on the mode

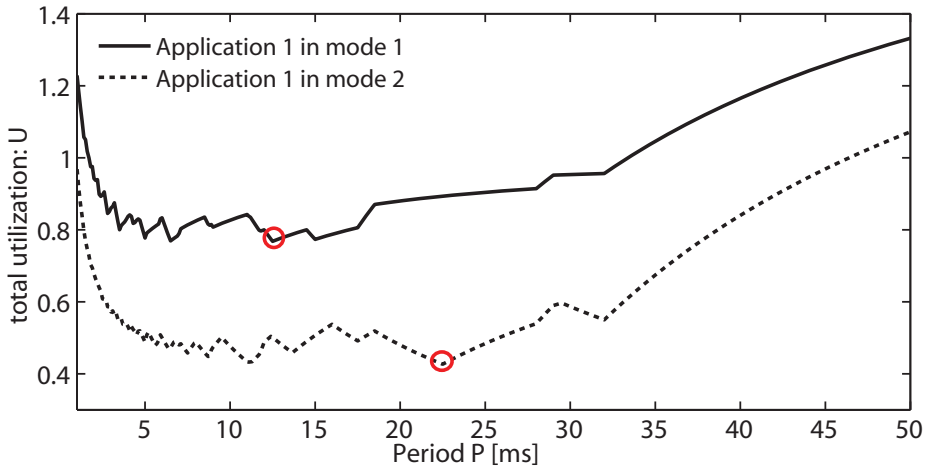

Fig. 6.23: Total utilization for period varying from 1 ms up-to 50 ms considering the two different modes of application 1. The circles on the graphs denote the points of minimum utilization

that application 1 is currently in. Figure 6.23 shows the total utilization of the system as a function of the period of the servers considering the two modes of application 1, where the period varies from 1 ms to 50 ms. When application 1 is in mode 1, the system has the minimum utilization ($U = 0.768$) with period $P = 12.5$ ms, and allocated budgets for application 1 and application 2, $Q_1 = 8$ ms and $Q_2 = 1$ ms, respectively. When application 1 is in mode 2, however, the system has the minimum utilization ($U = 0.427$) achieved for period $P = 22.5$ ms, and budgets $Q_1 = 7$ ms and $Q_2 = 2$ ms, respectively.

Since the mode of application 1 changes dynamically during run-time, it is not possible to fix the parameters of the scheduler at design time and achieve minimum utilization. If the parameters are set to the optimal ones for mode 1, when operating in mode 2 the system would have a 15 % utilization overhead. Similarly fixing the parameters optimally for mode 2, the utilization overhead would be 14 % when application 1 is in mode 1.

We can solve the above problem by using the algorithms proposed in this chapter. Let us consider the following two scenarios.

Scenario 1. When application 1 is in mode 1, we run the two ASG servers corresponding to the two applications with parameters $(8, 12.5)$ and $(1, 12.5)$ which give us the minimum system utilization. When application 1 switches to mode 2, it notifies the Server Manager (SM) and it requests a switch to the minimum budget for mode 2, $(4.7, 12.5)$. The SM can grant this budget using Algorithm 6.2. Afterwards the SM can reconfigure the two ASG servers and increase their period to the one which makes the system utilization the smallest. The SM can use

Algorithm 6.5 with $K = 1$ to reconfigure the system from $(4.7, 12.5)$ and $(1, 12.5)$ to $(7, 22.5)$ and $(2, 22.5)$.

Scenario 2. When application 1 has to switch back to mode 1, it first notifies the SM which by using Algorithm 6.6 with $K = 1$ reconfigures the two servers from $(7, 22.5)$ and $(2, 22.5)$ back to $(4.7, 12.5)$ and $(1, 12.5)$. Then the SM increases the budget for application 1 using Algorithm 6.4 from 4.7 to 8. Afterwards, application 1 is notified and can safely switch to mode 1.

Note that the SM takes advantage of the fact that mode 1 is more heavily loaded than mode 2 for application 1. Therefore, the SM optimizes the server period when the application is in the lightly loaded mode. This means that in Scenario 1, the application mode change is done before the resource optimization. And in Scenario 2, it is done after the resource optimization. This is feasible with our algorithms as they are completely deterministic and the time needed for a reconfiguration can be safely and accurately upper bounded in advance. It is also possible to perform the resource optimization when the system is more heavily loaded however, the reconfiguration process will take longer.

In summary, we can guarantee an optimal resource allocation in environments where applications are added or removed dynamically, or perform mode-changes. With the proposed algorithms, the schedulability of the applications is never compromised during the reconfiguration process.

Setup. The servers and applications have been modeled with the MATLAB Real-Time Calculus Toolbox [WT06c]. The exploration of the minimum required budgets for different periods as shown in Fig. 6.23 has been done with the Real-Time Interfaces methodology as described in Chapters 2 and 3. The exploration took less than 15 s to perform on a commodity laptop considering discretization of the period with steps of 0.1 ms. The search for the value of K took less than 1 s.

6.7 Discussion

The chapter considers the problem of adaptive resource reservations using servers in hard real-time systems. It classifies the possible problems that may occur during online server reconfigurations and establishes conditions of how to avoid them. It defines a statically TDMA scheduled adaptive server that provides resource guarantees not only during normal operation, but also during reconfigurations. The chapter identifies the possible reconfiguration scenarios for such a server and provides algorithms and schedulability analysis for each of them. The analysis is

based on Real-Time Calculus which even for the simplest case of TDMA scheduled servers is not trivial.

The server reconfiguration analysis and algorithms presented here guarantee schedulability of hard real-time applications even during reconfiguration phases. They are suitable for multi-mode real-time systems because the scheduling servers can be reconfigured during run-time in pace with the changing schedulability requirements of the applications. Therefore, multi-mode real-time systems that have been previously classified incorrectly as unschedulable, now can become schedulable as the analysis can accurately take into account that servers can always be run with optimal parameters that minimize system utilization with respect to the possible combinations of operating modes of the applications.

The general framework proposed here for reconfigurations of servers, and the respective algorithms and analysis, are applicable not only to TDMA-based servers, but also can be used as a methodology for developing similar results for other kinds of servers such as the CBS, the deferable, and others. The question whether there exists a server that is more suitable for online reconfigurations while providing scheduling guarantees is still open.

7

Conclusions

The aim of this thesis is to show that it is possible to develop accurate compositional system-level performance analysis methods for complex embedded real-time systems which use multiple distributed computational and communication resources, use complex resource sharing policies, have cycles in the event flows which can create backpressure, and can change their parameters at run-time due to changing system or environment conditions. Moreover, it shows that it is possible to develop a general framework for interface-based design of real-time systems which can support the early design phase by developing interfaces of system components that can take into account various constraints on time, resources, and memory usage.

7.1 Main Results

The thesis extends previous state-of-the-art results on Modular Performance Analysis with Real-Time Calculus and interface-based design with Real-Time Interfaces in the following several ways:

- **Interface-Based Design for Real-Time Systems.** We formalize and generalize the framework of Real-Time Interfaces by formally defining the concepts of abstract components, and their respective interfaces, and establishing properties for the successful composition of components and interfaces while respecting real-time constraints. The important aspects such as independent implementability, refinement, and incremental design are defined and corresponding conditions are derived. The proposed notion of interfaces supports the design by providing mechanisms to propagate real-time constraints. Interfaces are no longer static but adapt according to changes in the connected components

and system environment. This can be used to answer synthesis questions at design time, and to adapt a system at run-time when the requirements from the environment change.

- **Algebra for Interface-Based Design of Real-Time Systems.** We formulate the rate analysis problem in an interface-based design setting. It allows for compositional design of embedded systems whose components communicate through data or event streams. For this purpose, we develop the Rate Interfaces theory which can effectively check compatibility between components and guarantee various buffer and delay constraints. Rate Interfaces extend previous results on Real-Time Interfaces towards distributed systems. Therefore, we not only consider tasks that are executed on a single processing element but allow for data streams that are processed on several computing resources and communicated through different communication media. We consider variable execution demands of tasks in an interface-based design approach which improves the accuracy of the performance results. We consider not only component-wise constraints such as buffer underflow and overflow, but also constraints on networks of components such as end-to-end delays.

- **Compositional Performance Analysis of Cyclic Dataflow Real-Time Systems.** Component-based performance analysis methods are extended to the class of marked graphs. Unlike other known methods, the approach takes into account a general model for resource interaction based on the concept of service curves, and a general data stream model based on arrival curves. Performance bounds obtained with the newly described method have higher accuracy than the ones obtained with any of the previously known methods. The analysis can cover any system that can be modeled with a marked graph including ones with cyclic data dependencies, finite buffer sizes, non-deterministic resource behavior, and complex scheduling policies. It can be embedded into existing compositional frameworks such as the SymTA/S or the Modular Performance Analysis.

- **Performance Analysis of Multi-Mode Real-Time Applications.** We present a method for timing and buffer size analysis of uniprocessor multi-mode systems with fixed priority scheduling of tasks. It improves on accuracy and scope of previous mode change analysis methods since it supports complex task activation patterns by the use of arrival curves, and non-deterministic resource behaviors by the use of service curves. We show how the method

can be applied to transform a non-schedulable mode change into a schedulable one using an offset.

- **Performance Analysis of Run-Time Reconfigurable Resource Reservations in Real-Time Systems.** The problem of scheduler adaptations in resource partitioned architectures is considered from the perspective of adaptive servers that provide guarantees on real-time properties. It is the first discussion on the topic where not only schedulability of reconfigurable servers is considered, but also schedulability of applications during server reconfigurations. An adaptive scheduling server framework based on the TDMA partitioning paradigm is developed. Algorithms are developed for different reconfiguration scenarios that can guarantee meeting of real-time constraints of applications during server reconfigurations.

7.2 Future Directions

This thesis presents several substantial extensions of the scope and accuracy of previously existing compositional analysis and design frameworks for embedded real-time systems. However, their applicability and relevancy in modern industrial systems are not clear. Realistic real-time systems are likely to exhibit multiple complexities which can be any combination of multiple computational and communication resources, complex resource sharing policies, variable execution demands of applications, various memory and delay constraints, cyclic dataflow dependencies, and adaptive behavior in applications and resources.

The extensions presented here can deal with systems that only have memory and delay constraints, only cyclic dataflow dependencies, or only adaptive behavior. A method that can analyze accurately systems that exhibit a combination of such complexities does not exist yet. Development of such a method is not only driven by an academic need for a unified framework for compositional analysis and design of complex embedded real-time system, but also it is necessitated by the recent development in complexity and features of modern industrial embedded systems. It is common, nowadays, for mobile phones and tablets to have multiple processing cores connected over various buses with memory, caches, and specialized processors. Resource contentions and complex resource sharing policies are ubiquitous in such platforms. The devices need to perform various signal-, voice-, image-, and video-processing tasks which are usually modeled with formalisms with cyclic dataflow dependencies. Applications may be switched on and off by the user at any time which means that the analysis has to deal with the adaptive behavior

of the system that tries to maintain its real-time properties despite of the constant changes inside of it and in the environment.

Having one unified performance analysis and design framework for complex embedded real-time systems will make its adoption in industry much more likely. This will not only increase the quality and features of industrial embedded real-time systems, but will also support future research in this area.

Based on the conclusion drawn above, we can outline three main directions for continuing the work from this thesis:

- **Interface-Based Design for Real-Time Systems with Cyclic Dataflow.** Results presented in Chapters 2 and 3 focus on interface-based design frameworks for distributed embedded real-time systems that have various memory and delay constraints. However, they cannot deal with systems that have cyclic dataflow dependencies or finite buffer sizes. On the other hand, Chapter 4 develops a compositional performance analysis method specifically tailored to distributed systems with cycles in the event flows or with finite buffers that create backpressure. Combining these results would bring the advantages of interface-based design to the design flow of cyclic dataflow systems. Besides the questions already mentioned in Chapters 2 and 3, a designer would be able to efficiently answer questions such as: what is the design space of minimum buffer sizes in a system with cyclic data dependencies in order to meet a certain throughput constraint, or what is the minimum required CPU frequency by a component in a system with backpressure.

- **Compositional Performance Analysis of Distributed Multi-Mode Real-Time Applications.** Results presented in Chapters 5 and 6 focus on timing analysis of adaptive behavior in uniprocessor embedded real-time systems. Extending these results to multicore or multiprocessor systems is necessary because most realistic systems nowadays exhibit multiple processing and communication resources. Such an analysis will not be trivial as it has to accurately take into account that effects from adaptive behavior may propagate through the multiple processing elements in the system for indefinitely long during run-time.

- **Interface-Based Design for Real-Time Systems with Adaptive Behavior.** Results presented in Chapters 2 and 3 focus on interface-based design frameworks that are limited to systems whose parameters do not change during run-time. On the other hand, results in Chapters 5 and 6 focus on timing and buffer

analysis of adaptive systems without considering the features of interface-based design. It is desirable to combine them as it will allow the use of interface theory in online performance analysis for reconfigurations whose parameters are only known at run-time. It will also bring traditional benefits of interface-based design to multi-mode real-time systems such as being able to efficiently answer design questions. On the other hand, this will not be trivial to achieve as it will require designing interfaces that contain information not only about the bounds of certain parameters, but also information about how and when one can switch from one value of a parameter to another one during run-time without violating real-time constraints.

A

Min/Max Algebra

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

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

The duality between $\otimes$ and $\oslash$ states that:

$$f \oslash g \le h \iff f \le g \otimes h\ .$$

Isotonicity property of the min-plus convolution operator $\otimes$ is defined as:

$$\text{If } f \le g \text{ and } f' \le g' \text{ then } f \otimes f' \le g \otimes g'\ .$$

A function f is wide-sense increasing iff $f(s) \le f(t)$ for all $s \le t$.

The max-plus deconvolution operator $\overline{\oslash}$ is defined as:

$$(a \overline{\oslash} b)(\Delta) = \inf_{\lambda \ge 0} \{a(\Delta + \lambda) - b(\lambda)\}\ .$$

For details on the above operators and their properties see [LBT01].

B

Real-Time Calculus

The operator $RT(\beta, \alpha)$ for the remaining service curve of a GPC is defined as:

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

Its two inverses are defined as:

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

with solutions:

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

and

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

Bibliography

[AB98] Luca Abeni and Giorgio Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, RTSS '98, pages 4–13, Washington, DC, USA, 1998. IEEE Computer Society.

[AB99] Luca Abeni and Giorgio Buttazzo. Adaptive bandwidth reservation for multimedia computing. In *Proceedings of the Sixth International Conference on Real-Time Computing Systems and Applications*, RTCSA '99, pages 70–77, Washington, DC, USA, 1999. IEEE Computer Society.

[AB01] Luca Abeni and Giorgio Buttazzo. Hierarchical QoS management for time sensitive applications. In *Proceedings of the Seventh Real-Time Technology and Applications Symposium (RTAS '01)*, RTAS '01, pages 63–72, Washington, DC, USA, 2001. IEEE Computer Society.

[AB04] Luca Abeni and Giorgio Buttazzo. Resource reservation in dynamic real-time systems. *Real-Time Systems*, 27(2):123–167, 2004.

[ALE02] T. Austin, E. Larson, and D. Ernst. SimpleScalar: An infrastructure for computer system modeling. *IEEE Computer*, 35(2):59–67, 2002.

[And08] Björn Andersson. Uniprocessor EDF scheduling with mode change. In Theodore Baker, Alain Bui, and Sébastien Tixeuil, editors, *Principles of Distributed Systems*, volume 5401 of *Lecture Notes in Computer Science*, pages 572–577. Springer Berlin / Heidelberg, 2008.

[BA00] G. Buttazzo and L. Abeni. Adaptive rate control through elastic scheduling. In *Decision and Control, 2000. Proceedings of the 39th IEEE Conference on*, volume 5, pages 4883–4888 vol.5, 2000.

[Bar03] S. K. Baruah. Dynamic- and static-priority scheduling of recurring real-time tasks. *Real-Time Systems*, 24(1):93–128, 2003.

[BBLB03] Scott A. Brandt, Scott Banachowski, Caixue Lin, and Timothy Bisson. Dynamic integrated scheduling of hard real-time soft real-time and non-real-time processes. In *Proceedings of the 24th IEEE International Real-Time Systems Symposium*, RTSS '03, pages 396–407, Washington, DC, USA, 2003. IEEE Computer Society.

[BCGM99] Sanjoy Baruah, Deji Chen, Sergey Gorinsky, and Aloysius Mok. Generalized multiframe tasks. *Real-Time Systems*, 17(1):5–22, July 1999.

[BDN05] Enrico Bini and Marco Di Natale. Optimal task rate selection in fixed priority systems. In *Proceedings of the 26th IEEE International Real-Time Systems Symposium*, pages 399–409, Washington, DC, USA, 2005. IEEE Computer Society.

[BJLL06] Amit Bose, Xiaoyue Jiang, Bin Liu, and Gang Li. Analysis of manufacturing blocking systems with network calculus. *Performance Evaluation*, 63(12):1216–1234, December 2006.

[BLCA02] Giorgio C. Buttazzo, Giuseppe Lipari, Marco Caccamo, and Luca Abeni. Elastic scheduling for flexible workload management. *IEEE Transactions on Computers*, 51(3):289–302, March 2002.

[BML99] Shuvra S. Bhattacharyya, Praveen K. Murthy, and Edward A. Lee. Synthesis of embedded software from synchronous dataflow specifications. *Journal of VLSI Signal Processing Systems*, 21(2):151–166, 1999.

[BOQC92] F.L. Baccelli, G.J. Olsder, J.P. Quadrat, and G. Cohen. *Synchronization and Linearity: An Algebra for Discrete Event Systems*. Wiley Series on Probability and Mathematical Statistics: Probability and Mathematical Statistics, 1992.

[BPC09] Anne Bouillard, Linh T. X. Phan, and Samarjit Chakraborty. Lightweight modeling of complex state dependencies in stream processing systems. In *Proceedings of the 2009 15th IEEE Symposium on Real-Time and Embedded Technology and Applications*, RTAS '09, pages 195–204, Washington, DC, USA, 2009. IEEE Computer Society.

[CdAHS03] Arindam Chakrabarti, Luca de Alfaro, Thomas Henzinger, and Mariëlle Stoelinga. Resource interfaces. In *Embedded Software*, volume 2855 of *Lecture Notes in Computer Science*, pages 117–133. Springer Berlin / Heidelberg, 2003.

[CDQV85] Guy Cohen, Didier Dubois, Jean Pierre Quadrat, and Michel Voit. A linear-system-theoretic view of discrete-event processes and its use for performance avaluation in manufacturing. *IEEE Transactions on Automatic Control*, AC-30(3):210–220, 1985.

[Cha00] C.S. Chang. *Performance Guarantees in Communication Networks*. Springer-Verlag, London, UK, 2000.

[CKP+09] S.S. Craciunas, C.M. Kirsch, H. Payer, H. Rock, and A. Sokolova. Programmable temporal isolation through variable-bandwidth servers. In *Industrial Embedded Systems, 2009. SIES '09. IEEE International Symposium on*, pages 171–180, 2009.

[CKT+03a] S. Chakraborty, S. Künzli, L. Thiele, A. Herkersdorf, and P. Sagmeister. Performance evaluation of network processor architectures: Combining simulation with analytical estimation. *Computer Networks*, 41(5):641–665, April 2003.

[CKT03b] Samarjit Chakraborty, Simon Künzli, and Lothar Thiele. 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 - Volume 1*, DATE '03, pages 10190–10195, Washington, DC, USA, 2003. IEEE Computer Society.

[CPL08] Tommaso Cucinotta, Luigi Palopoli, and Giuseppe Lipari. FRESCOR Deliverable D-AQ2v2: control algorithms for coordinated resource-level and application-level adaptation v2, 2008.

[Cru91a] R.L. Cruz. A calculus for network delay. i. network elements in isolation. *Information Theory, IEEE Transactions on*, 37(1):114–131, January 1991.

[Cru91b] R.L. Cruz. A calculus for network delay. ii. network analysis. *Information Theory, IEEE Transactions on*, 37(1):132–141, January 1991.

[dAH01a] Luca de Alfaro and Thomas A. Henzinger. Interface automata. In *Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering*, ESEC/FSE-9, pages 109–120, New York, NY, USA, 2001. ACM.

[dAH01b] Luca de Alfaro and Thomas A. Henzinger. Interface theories for component-based design. In *Proceedings of the First International Workshop on Embedded Software*, EMSOFT '01, pages 148–165, London, UK, 2001. Springer-Verlag.

[dAH05] L. de Alfaro and T. A. Henzinger. Interface-based design. In M. Broy, J. Gruenbauer, D. Harel, and C.A.R. Hoare, editors, *Engineering Theories of Software-intensive Systems*, volume 195 of *NATO Science Series: Mathematics, Physics, and Chemistry*, pages 83–104. Springer, 2005.

[dAHS02] Luca de Alfaro, Thomas A. Henzinger, and Mariëlle Stoelinga. Timed interfaces. In *Proceedings of the Second International Conference on Embedded Software*, EMSOFT '02, pages 108–122, London, UK, 2002. Springer-Verlag.

[DB06] R. I. Davis and A. Burns. Resource sharing in hierarchical fixed priority pre-emptive systems. In *Proceedings of the 27th IEEE International Real-Time Systems Symposium*, pages 257–270, Washington, DC, USA, 2006. IEEE Computer Society.

[DP02] B.A. Davey and H.A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 2nd edition, 2002.

[ESSL06] Arvind Easwaran, Insik Shin, Oleg Sokolsky, and Insup Lee. Incremental schedulability analysis of hierarchical real-time components. In *Proceedings of the 6th ACM & IEEE International conference on Embedded software*, EMSOFT '06, pages 272–281, New York, NY, USA, 2006. ACM.

[Foh93] Gerhard Fohler. Changing operational modes in the context of pre run-time scheduling. *IEICE Transactions on Information and Systems*, 76(11):1333–1340, 1993.

[GGS+06] A. H. Ghamarian, M. C. W. Geilen, S. Stuijk, T. Basten, B. D. Theelen, M. R. Mousavi, A. J. M. Moonen, and M. J. G. Bekooij. Throughput analysis of synchronous data flow graphs. In *ACSD '06: Proceedings of the Sixth International Conference on Application of Concurrency to System Design*, pages 25–36, Washington, DC, USA, 2006. IEEE Computer Society.

[GLMS02] Thorsten Grötker, Stan Liao, Grant Martin, and Stuart Swan. *System Design with SystemC*. Springer US, 2002.

[Gua09] Qian Guangming. An earlier time for inserting and/or accelerating tasks. *Real-Time Systems*, 41:181–194, April 2009.

[Hai10] Wolfgang Haid. *Design and Performance Analysis of Multiprocessor Streaming Applications*. PhD thesis, ETH Zurich, 2010.

[HE05] Arne Hamann and Rolf Ernst. TDMA time slot and turn optimization with evolutionary search techniques. In *Proceedings of the conference on Design, Automation and Test in Europe - Volume 1*, DATE '05, pages 312–317, Washington, DC, USA, 2005. IEEE Computer Society.

[HE07] Rafik Henia and Rolf Ernst. Scenario aware analysis for complex event models and distributed systems. In *Proceedings of the 28th IEEE International Real-Time Systems Symposium*, RTSS '07, pages 171–180, Washington, DC, USA, 2007. IEEE Computer Society.

[HHK03] T.A. Henzinger, B. Horowitz, and C.M. Kirsch. Giotto: A time-triggered language for embedded programming. *Proceedings of the IEEE*, 91:84–99, 2003.

[HKL94] M. G. Harbour, M. H. Klein, and J. P. Lehoczky. Timing analysis for fixed-priority scheduling of hard real-time systems. *IEEE Transactions on Software Engineering*, 20(1):13–28, 1994.

[HM06] Thomas A. Henzinger and Slobodan Matic. An interface algebra for real-time components. In *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 253–266, Washington, DC, USA, 2006. IEEE Computer Society.

[HSdE09] Michael González Harbour, Daniel Sangorrín, and Miguel Tellería de Esteban. FRESCOR Deliverable D-AT2: schedulability analysis techniques for distributed systems, 2009.

[HT07] Wolfgang Haid and Lothar Thiele. Complex task activation schemes in system level performance analysis. In *CODES+ISSS '07: Proceedings of the 5th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, pages 173–178, New York, NY, USA, 2007. ACM.

[JE03] Marek Jersak and Rolf Ernst. Enabling scheduling analysis of heterogeneous systems with multi-rate data dependencies and rate intervals. In *DAC '03: Proceedings of the 40th conference on Design automation*, pages 454–459, New York, NY, USA, 2003. ACM Press.

[JPTY08] Bengt Jonsson, Simon Perathoner, Lothar Thiele, and Wang Yi. Cyclic dependencies in modular performance analysis. In *Proceedings of the 8th ACM international conference on Embedded software*, EMSOFT '08, pages 179–188, New York, NY, USA, 2008. ACM.

[JRE05] Marek Jersak, Kai Richter, and Rolf Ernst. Performance analysis for complex embedded applications. *International Journal of Embedded Systems*, 1(1/2):33–49, 2005.

[Kah74] Gilles Kahn. The semantics of a simple language for parallel programming. In *Information Processing 74: Proceedings of IFIP Congress 74*, pages 471–475, Stockholm, Sweden, August 1974. North-Holland.

[KTZ05] Simon Künzli, Lothar Thiele, and Eckart Zitzler. Modular design space exploration framework for embedded systems. *IEE Proceedings Computers & Digital Techniques*, 152(2):183–192, 2005.

[LBT01] Jean-Yves Le Boudec and Patrick Thiran. *Network calculus: a theory of deterministic queuing systems for the internet*, volume 2050 of *LNCS*. Springer-Verlag, 2001.

[LCM06] Yanhong Liu, Samararjit Chakraborty, and Radu Marculescu. Generalized rate analysis for media-processing platforms. In *Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, RTCSA '06, pages 305–314, Washington, DC, USA, 2006. IEEE Computer Society.

[LLB06] José L. Lorente, Giuseppe Lipari, and Enrico Bini. A hierarchical scheduling model for component-based real-time systems. In *Proceedings of the 20th international conference on Parallel and distributed processing*, IPDPS '06, pages 8–15, Washington, DC, USA, 2006. IEEE Computer Society.

[LM87] E.A. Lee and D.G. Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9):1235–1245, Sept. 1987.

[LP95] E.A. Lee and T.M. Parks. Dataflow process networks. *Proceedings of the IEEE*, 83(5):773–801, May 1995.

[LSS87] John P. Lehoczky, Lui Sha, and Jay K. Strosnider. Enhanced aperiodic responsiveness in hard real-time environments. In *Proceedings of the IEEE Real-Time System Symposium*, RTSS '87, pages 261–270, 1987.

[Max05] Alexandre Maxiaguine. *Modeling Multimedia Workloads for Embedded System Design*. PhD thesis, ETH Zurich, 2005.

[MDG98] Anmol Mathur, Ali Dasdan, and Rajesh K. Gupta. Rate analysis for embedded systems. *ACM Transactions on Design Automation of Electronic Systems*, 3(3):408–436, July 1998.

[MF01] Aloysius K. Mok and Xiang (Alex) Feng. Towards compositionality in real-time resource partitioning based on regularity bounds. In *Proceedings of the 22nd IEEE Real-Time Systems Symposium*, RTSS '01, pages 129–138, Washington, DC, USA, 2001. IEEE Computer Society.

[MKCT04] Alexander Maxiaguine, Simon Künzli, Samarjit Chakraborty, and Lothar Thiele. Rate analysis for streaming applications with on-chip buffer constraints. In *Proceedings of the 2004 Asia and South Pacific Design Automation Conference*, ASP-DAC '04, pages 131–136, Piscataway, NJ, USA, 2004. IEEE Press.

[MKT04] Alexander Maxiaguine, Simon Künzli, and Lothar Thiele. Workload characterization model for tasks with variable execution demand. In *Proceedings of the 7th Design, Automation and Test in Europe (DATE)*, page 21040, Washington, DC, USA, 2004. IEEE Computer Society.

[MRZ94] C. Mercer, R. Rajkumar, and J. Zelenka. Temporal protection in real-time operating systems. In *Proceedings 11th IEEE Workshop on Real-Time Operating Systems and Software (RTOSS)*, pages 79–83, 1994.

[Mur89] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, Apr 1989.

[MVB07] Orlando Moreira, Frederico Valente, and Marco Bekooij. Scheduling multiple independent hard-real-time jobs on a heterogeneous multiprocessor. In *Proceedings of the 7th ACM & IEEE international conference on Embedded software*, EMSOFT '07, pages 57–66, New York, NY, USA, 2007. ACM.

[MZCW04] Alexander Maxiaguine, Yongxin Zhu, Samarjit Chakraborty, and Weng-Fai Wong. Tuning SoC platforms for multimedia processing: Identifying limits and tradeoffs. In *Proceedings of the international conference on Hardware/Software Codesign and System Synthesis: 2004*, CODES+ISSS '04, pages 128–133, Washington, DC, USA, 2004. IEEE Computer Society.

[NGA09] Vincent Nelis, Joel Goossens, and Bjorn Andersson. Two protocols for scheduling multi-mode real-time systems upon identical multiprocessor platforms. In *Proceedings of the 2009 21st Euromicro Conference on Real-Time Systems*, ECRTS '09, pages 151–160, Washington, DC, USA, 2009. IEEE Computer Society.

[OCL09] Augusto Born de Oliveira, Eduardo Camponogara, and George Lima. Dynamic reconfiguration in reservation-based scheduling: An optimization approach. In *Proceedings of the 2009 15th IEEE Symposium on Real-Time and Embedded Technology and Applications*, RTAS '09, pages 173–182, Washington, DC, USA, 2009. IEEE Computer Society.

[PB98] P. Pedro and A. Burns. Schedulability analysis for mode changes in flexible real-time systems. In *Real-Time Systems, 1998. Proceedings. 10th Euromicro Workshop on*, pages 172 –179, June 1998.

[PBB+03] P. Poplavko, T. Basten, M. Bekooij, J. van Meerbergen, and B. Mesman. Task-level timing models for guaranteed performance in multiprocessor networks-on-chip. In *CASES '03: Proceedings of the 2003 international conference on Compilers, architecture and synthesis for embedded systems*, pages 63–72, New York, NY, USA, 2003. ACM.

[PEP02] Traian Pop, Petru Eles, and Zebo Peng. Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems. In *Proceedings of the tenth international symposium on Hardware/software codesign*, CODES '02, pages 187–192, New York, NY, USA, 2002. ACM.

[RC04] Jorge Real and Alfons Crespo. Mode change protocols for real-time systems: A survey and a new proposal. *Real-Time Systems*, 26:161–197, March 2004.

[Rei68] Raymond Reiter. Scheduling parallel computations. *Journal of the ACM*, 15(4):590–599, 1968.

[RJE03] Kai Richter, Marek Jersak, and Rolf Ernst. A formal approach to MpSoC performance verification. *Computer*, 36(4):60–67, April 2003.

[RvEP02] Martijn J. Rutten, Jos T. J. van Eijndhoven, and Evert-Jan D. Pol. Robust media processing in a flexible and cost-effective network of multi-tasking coprocessors. In *Proceedings of the 14th Euromicro Conference on Real-Time Systems*, pages 223–230, Washington, DC, USA, 2002. IEEE Computer Society.

[SB96] Marco Spuri and Giorgio Buttazzo. Scheduling aperiodic tasks in dynamic priority systems. *Real-Time Systems*, 10(2):179–210, 1996.

[SB00] Sundararajan Sriram and Shuvra S. Bhattacharyya. *Embedded multiprocessors: Scheduling and synchronization*. CRC Press, 2000.

[SBNN08] Insik Shin, Moris Behnam, Thomas Nolte, and Mikael Nolin. Synthesis of optimal interfaces for hierarchical scheduling with resources. In *Proceedings of the 2008 Real-Time Systems Symposium*, RTSS '08, pages 209–220, Washington, DC, USA, 2008. IEEE Computer Society.

[Sch11] Andreas Schranzhofer. *Efficiency and predictability in resource sharing multicore systems*. PhD thesis, ETH Zurich, 2011.

[SL03] Insik Shin and Insup Lee. Periodic resource model for compositional real-time guarantees. In *Proceedings of the 24th IEEE International Real-Time Systems Symposium*, RTSS '03, pages 2–13, Washington, DC, USA, 2003. IEEE Computer Society.

[SL04] Insik Shin and Insup Lee. Compositional real-time scheduling framework. In *Proceedings of the 25th IEEE International Real-Time Systems Symposium*, pages 57–67, Washington, DC, USA, 2004. IEEE Computer Society.

[SLR86] L. Sha, J. P. Lehoczky, and R. Rajkumar. Solutions for some practical problems in prioritized preemptive scheduling. In *IEEE Real-Time Systems Symposium*, RTSS '86, pages 181–191, 1986.

[SLS95] Jay K. Strosnider, John P. Lehoczky, and Lui Sha. The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments. *IEEE Transactions on Computers*, 44(1):73–91, January 1995.

[SPT09] Nikolay Stoimenov, Simon Perathoner, and Lothar Thiele. Reliable mode changes in real-time systems with fixed priority or EDF scheduling. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '09, pages 99–104, 3001 Leuven, Belgium, Belgium, 2009. European Design and Automation Association.

[SRLR89] L. Sha, R. Rajkumar, J. Lehoczky, and K. Ramamritham. Mode change protocols for priority-driven preemptive scheduling. *Real-Time Systems*, 1(3):243–264, 1989.

[SSE07] Simon Schliecker, Steffen Stein, and Rolf Ernst. Performance analysis of complex systems by integration of dataflow graphs and compositional performance analysis. In *DATE '07: Proceedings of the conference on Design, Automation and Test in Europe*, pages 273–278, San Jose, CA, USA, 2007. EDA Consortium.

[SSL89] B. Sprunt, L. Sha, and J. P. Lehoczky. Aperiodic task scheduling for hard real-time systems. *Real-Time Systems*, 1(1):27–60, 1989.

[TBW92] K.W. Tindell, A. Burns, and A.J. Wellings. Mode changes in priority pre-emptively scheduled systems. In *Real-Time Systems Symposium, 1992*, pages 100–109, December 1992.

[TC94] Ken Tindell and John Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and Microprogramming*, 40(2-3):117–134, 1994.

[TCGK02] Lothar Thiele, Samarjit Chakraborty, Matthias Gries, and Simon Künzli. A framework for evaluating design tradeoffs in packet processing architectures. In *Proceedings of the 39th annual Design Automation Conference*, DAC '02, pages 880–885, New York, NY, USA, 2002. ACM.

[TCN00] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on*, volume 4, pages 101–104 vol.4, 2000.

[TS09] Lothar Thiele and Nikolay Stoimenov. Real-Time Simulation (RTS) Toolbox.
`http://www.mpa.ethz.ch/Rtstoolbox`, 2009.

[TW06] Lothar Thiele and Ernesto Wandeler. Performance analysis of distributed embedded systems. In Richard Zurawski, editor, *Embedded Systems Handbook*, pages 15–1–15–18. CRC Taylor & Francis, 2006.

[VAdlP09] Marisol Garcia Valls, Alejandro Alonso, and Juan A. de la Puente. Mode change protocols for predictable contract-based resource management in embedded multimedia systems. In *Second International Conference on Embedded Software and Systems*, pages 221–230, 2009.

[VM04] Girish V. Varatkar and Radu Marculescu. On-chip traffic modeling and synthesis for MPEG-2 video applications. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 12(1):108–119, January 2004.

[Wan06] Ernesto Wandeler. *Modular Performance Analysis and Interface-Based Design for Embedded Real-Time Systems*. PhD thesis, ETH Zurich, 2006.

[WBJS06] Maarten Wiggers, Marco Bekooij, Pierre Jansen, and Gerard Smit. Efficient computation of buffer capacities for multi-rate real-time systems with back-pressure. In *CODES+ISSS '06: Proceedings of the 4th international conference on Hardware/Software codesign and system synthesis*, pages 10–15, New York, NY, USA, 2006. ACM.

[WRM+05] Shengquan Wang, Sangig Rho, Zhibin Mai, Riccardo Bettati, and Wei Zhao. Real-time component-based systems. In *Proceedings of the 11th IEEE Real Time on Embedded Technology and Applications Symposium*, pages 428–437, Washington, DC, USA, 2005. IEEE Computer Society.

[WT05a] Ernesto Wandele and Lothar Thiele. Abstracting functionality for modular performance analysis of hard real-time systems. In *Proceedings of the 2005 Asia and South Pacific Design Automation Conference*, ASP-DAC '05, pages 697–702, New York, NY, USA, 2005. ACM.

[WT05b] Ernesto Wandeler and Lothar Thiele. Real-time interfaces for interface-based design of real-time systems with fixed priority scheduling. In *Proceedings of the 5th ACM international conference on Embedded software*, EMSOFT '05, pages 80–89, New York, NY, USA, 2005. ACM.

[WT06a] Ernesto Wandeler and Lothar Thiele. Interface-based design of real-time systems with hierarchical scheduling. In *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 243–252, Washington, DC, USA, 2006. IEEE Computer Society.

[WT06b] Ernesto Wandeler and Lothar Thiele. 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, pages 479–484, Piscataway, NJ, USA, 2006. IEEE Press.

[WT06c] Ernesto Wandeler and Lothar Thiele. Real-Time Calculus (RTC) Toolbox. `http://www.mpa.ethz.ch/Rtctoolbox`, 2006.

[WTVL06] Ernesto Wandeler, Lothar Thiele, Marcel Verhoef, and Paul Lieverse. System architecture evaluation using modular performance analysis: a case study. *International Journal on Software Tools for Technology Transfer (STTT)*, 8(6):649–667, 2006.

[YNG10] P. Meumeu Yomsi, V. Nelis, and J. Goossens. Scheduling multi-mode real-time systems upon uniform multiprocessor platforms. In *The 15th IEEE International Conference on Emerging Technologies and Factory Automation*, number MF-002178 in ETFA '10. IEEE Computer Society Press, 2010.

List of Publications

The following list summarizes the publications which constitute the basis of this thesis. The corresponding chapters of this thesis are given in brackets.

Lothar Thiele, Ernesto Wandeler, and Nikolay Stoimenov. 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, NY, USA, 34–43.
(Chapter 2)

Samarjit Chakraborty, Yanhong Liu, Nikolay Stoimenov, Lothar Thiele, and Ernesto Wandeler. 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, USA, 25–34.
(Chapter 3)

Nikolay Stoimenov, Samarjit Chakraborty, and Lothar Thiele. 2010. An Interface Algebra for Estimating Worst-Case Traversal Times in Component Networks. In *Proceedings of 4th International Symposium on Leveraging Applications of Formal Methods, Verification, and Validation* (ISoLA '10). Springer-Verlag, Berlin, Heidelberg, Germany, 198–213.
(Chapter 3)

Lothar Thiele and Nikolay Stoimenov. 2009. Modular Performance Analysis of Cyclic Dataflow Graphs. In *Proceedings of the 9th ACM International Conference on Embedded Software* (EMSOFT '09). ACM, New York, NY, USA, 127–136.
(Chapter 4)

Nikolay Stoimenov, Simon Perathoner, and Lothar Thiele. 2009. Reliable Mode Changes in Real-Time Systems with Fixed Priority or EDF Scheduling. In *Proceedings of the Conference on Design, Automation and Test in Europe* (DATE '09). European Design and Automation Association, 3001 Leuven, Belgium, Belgium, 99–104.
(Chapter 5)

Nikolay Stoimenov, Lothar Thiele, Luca Santinelli, and Giorgio Buttazzo. Resource Adaptations with Servers for Hard Real-Time Systems. In *Proceedings of the 10th ACM International Conference on Embedded Software* (EMSOFT '10). ACM, New York, NY, USA, 269–278.
(Chapter 6)

It follows a list of publications that are not covered in this thesis.

Nikolay Stoimenov and Lothar Thiele. 2009. Interface-Based Design Approach for Analysis of Mode Changes in Distributed Real-Time Systems. In *Proceedings of the 15th IEEE Real-Time and Embedded Technology and Applications Symposium: Proceedings of Demos and Posters* (RTAS '09). San Francisco, CA, USA.

Jian-Jia Chen, Nikolay Stoimenov, and Lothar Thiele. 2009. Feasibility Analysis of On-Line DVS Algorithms for Scheduling Arbitrary Event Streams. In *Proceedings of the 30th IEEE Real-Time Systems Symposium* (RTSS '09). IEEE Computer Society, Washington, DC, USA, 261–270.

Simon Perathoner, Kai Lampka, Nikolay Stoimenov, Lothar Thiele, and Jian-Jia Chen. 2010. Combining Optimistic and Pessimistic DVS Scheduling: An Adaptive Scheme and Analysis. In *Proceedings of the 2010 IEEE/ACM International Conference on Computer-Aided Design* (ICCAD '10). IEEE, San Jose, CA, USA, 131–138.

Curriculum Vitæ

Name:	Nikolay Stoimenov
Date of birth:	April 25, 1979
Place of birth:	Varna, Bulgaria
Citizenship:	Bulgaria

Education

2005–2011 **ETH Zurich, Switzerland**
Research in Performance Analysis Methods for Embedded Real-Time Systems

Doctor of Sciences (Dr. sc. ETH Zurich)
Thesis: *Compositional Design and Analysis of Distributed, Cyclic, and Adaptive Embedded Real-Time Systems*
with Prof. Dr. Lothar Thiele

2001–2005 **University of Adelaide, Australia**
Studies in Computer Science, Mathematics, and Statistics

Bachelor of Computer Science Honours
First Class
Apple Prize for Best Honours Student
Aquinas College Gold Academic Medal
Thesis: *Investigating a Finite-State Machine Notation for Discrete-Event Systems*
with Dr. Robert Esser, Dr. Charles Lakos, and Mr. David Knight

Bachelor of Computer Science
Adelaide Achiever Scholarship International
Dean's Certificates of Merit

1993–1998	**Secondary School of Economics "G.S.Rakovski", Varna, Bulgaria** Maturas in Management, Mathematics, Literature, and English
	Secondary School Diploma in Management Award for Best Student

Professional Experience

2005–2011	Research and Teaching Assistant at the Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Switzerland
2001–2005	Head Network Administrator and Academic Tutor at Aquinas College, Adelaide, Australia
2004	Internship as Junior Researcher at the Robotic System Group, Australian National University, Australia
2003	Internship as Software Engineer at the School of Computer Science, University of Adelaide, Australia
2002	Internship as Software Engineer at the School of Computer Science, University of Adelaide, Australia