Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

MA-2006-05

Evaluation and Comparison of Performance Analysis Methods for Distributed Embedded Systems

Master's thesis
presented by

Simon Perathoner
Politecnico di Milano, Italy

Supervisors:

Prof. Dr. Lothar Thiele, Dipl. Ing. Ernesto Wandeler
Computer Engineering and Networks Laboratory
Swiss Federal Institute of Technology, Zürich

Prof. Dr. William Fornaciari
Dipartimento di Elettronica e Informazione
Politecnico di Milano

March 2006

to my family

Acknowledgements

First of all I would like to thank Prof. Dr. William Fornaciari for supporting me in writing this thesis abroad.

I would also like to express my sincere gratitude to Prof. Dr. Lothar Thiele for giving me the opportunity to write this thesis in his research group at the Computer Engineering and Networks Lab of the Swiss Federal Institute of Technology (ETH) Zürich. In particular, I would like to thank him for giving me the opportunity to take part in the ARTIST2 Workshop on Distributed Embedded Systems 2005 in Leiden, The Netherlands, which has considerably influenced the outcomes of this thesis.

Most of all I wish to thank Dipl. Ing. Ernesto Wandeler for his constant support throughout the whole project and for his ability to motivate me. Without his valuable input and advice this work would never have been possible.

I am also grateful to Dr. Alexandre Maxiaguine and Dipl. Ing. Simon Künzli for their help during this thesis work.

Finally, my warmest thanks go to my parents and my brother for their love and support during my studies.

Abstract

In this thesis we evaluate and compare a number of system level performance analysis methods for distributed embedded systems. We discuss different criteria for their classification and comparison and evaluate some concrete performance analysis techniques with respect to these criteria. In particular, we examine the modeling power and usability of various approaches, apply the performance analysis methods to a set of benchmark systems and compare the obtained results. We show that there are important differences between methods in terms of modeling effort and accuracy by highlighting some modeling difficulties and analyzing pitfalls of the different formal approaches. Further we present an extendible open-source library for performance simulation of distributed embedded systems based on SystemC and compare the hard performance bounds provided by formal analysis methods with the performance estimations obtained by simulation.

Table of Contents

Chapter 1

Introduction

1.1 Motivation

One of the major challenges in the design process of distributed embedded systems is to estimate performance characteristics of the final system implementation in the early design stages. In particular, during a system-level design process a designer faces several questions related to the system performance: Do the timing properties of a certain architecture meet the design requirements? What is the on-chip memory demand? What are the different CPU and bus utilizations? Which resource acts as a bottleneck? These and other similar questions are generally hard to answer for embedded system architectures that are highly heterogeneous, parallel and distributed and thus inherently complex.

Nevertheless, accurate performance predictions are essential for several reasons. First of all they are crucial in the domain of hard real-time applications, where provable guarantees of system performance are indispensable. In addition, higher standards of usability are now increasingly being applied to soft real-time systems, as well. Further, performance analysis plays a fundamental role in the design process of embedded systems. In particular, performance analysis is necessary to drive the design space exploration: different implementation variants in terms of partitioning, allocation and binding can only be evaluated on the basis of reliable performance predictions. And finally, the high market pressure to maximize the performance and minimize the price of embedded systems no longer permits designers to overallocate system resources in order to compensate for vague performance predictions.

Several different approaches to the performance analysis of distributed embedded systems can be found in the literature. However, the various approaches are very heterogenous in terms of modeling scope, modeling effort, tool support, accuracy and scalability and there is a lack of literature on their classification and comparison. It is difficult for a designer to ascertain which performance

analysis methods can be applied to a certain system and in particular which method is most suitable for his individual needs.

In this thesis we address this problem by evaluating, classifying and comparing different performance analysis methods.

1.2 Contributions

- We give an overview of approaches to the performance analysis of distributed embedded systems. We describe a number of concrete techniques that have been proposed so far and demonstrate their application.

- We discuss several criteria for the classification and comparison of performance analysis methods and evaluate a number of techniques with respect to these criteria. In particular we examine the modeling power, scalability and usability of various approaches.

- We then apply the different performance analysis methods to a number of benchmark systems defined by researchers of the ARTIST2 Network of Excellence on Embedded Systems Design[1]. We compare the results obtained in terms of accuracy and consider the required modeling and analysis effort. Further, we compare the hard performance bounds provided by formal analysis methods with worst-case estimations obtained by simulation.

- We present an extendible open-source library for performance simulation of distributed embedded systems based on SystemC, which we call PESIMDES. It is a repository of reusable modules that facilitates the system-level modeling and simulation of large distributed embedded systems using SystemC.

- We introduce a tool-independent and extendible data format for the description of distributed embedded systems and their performance analysis. This format is a first step towards an automated combination of several performance analysis tools.

[1] in the context of the ARTIST2 Workshop on Distributed Embedded Systems 2005 *(http://www.lorentzcenter.nl/lc/web/2005/20051121/info.php3?wsid=177)*

1.3 Overview

- In Chapter 2 we lay the foundations for the work presented. In particular we describe the principal characteristics of distributed embedded systems and the relevant performance metrics. Moreover, we identify the requirements for performance analysis methods.
- Chapter 3 gives an overview of approaches to performance analysis and describes a number of concrete analysis methods.
- In Chapter 4 we present PESIMDES, an extendible open-source library for performance simulation of distributed embedded systems based on SystemC.
- In Chapter 5 we discuss several criteria for the classification and comparison of performance analysis methods. Further, we evaluate and compare the techniques presented in Chapter 3 with regard to these criteria.
- In Chapter 6 we provide a number of case studies on performance analysis. In particular, we apply the performance analysis methods described in Chapter 3 to several benchmark systems and compare the results.
- Chapter 7 contains conclusions and perspectives for the future.

1.4 List of abbreviations

CAN	Controller area network
EDF	Earliest deadline first
FP	Fixed priority
GPS	Generalized processor sharing
MAST	Modeling and Analysis Suite for Real-Time Applications
PESIMDES	Performance Simulation of Distributed Embedded Systems
RM	Rate monotonic
SymTA/S	Symbolic Timing Analysis for Systems
TA	Timed automata
TDMA	Time division multiple access
WCET	Worst case execution time
WCRT	Worst case response time

Table 1.1: List of abbreviations used in this thesis

Chapter 2

Performance analysis of distributed embedded systems

2.1 Distributed embedded systems

Embedded systems are special purpose computing systems that are closely integrated into their environment. Typically, they are embedded into a larger product. In contrast to general purpose computing systems, embedded systems are dedicated to a specific application domain. This means that they execute only few applications, which are entirely known at design time. In particular, they are typically not programmable by the end user.

In general embedded systems must be efficient in terms of power consumption, size and cost. In addition, they usually have to be highly dependable, as a malfunction or breakdown of the device they control is not acceptable.

For many embedded systems not only the availability and the correctness of the computations is relevant, but also the timeliness of the computed results. Such systems with precise timing requirements are called real-time embedded systems. Their temporal behavior in terms of computation times, latencies and end-to-end delays is a functional system requirement. This means that a right answer arriving too late (or even too early) is wrong. Thus, real-time does not simply mean 'fast', but is a synonym for 'timely' and 'predictable'. In particular, real-time systems are called *hard*, if a missed deadline is not acceptable. All other embedded systems with timing constraints are called *soft*.

In most cases the timing constraints concern the reaction of the embedded system to events generated by the environment. In particular, embedded systems that must execute at a pace determined by the environment are called reactive systems. Examples are the control system of a nuclear power plant or the video processing system of a set-top box.

The embedding into a larger product and the constraints imposed by a

special application domain very often lead to distributed implementations. In particular physical, performance, safety and modularity constraints require distributed solutions. Distributed embedded systems consist of several hardware components that communicate via an interconnection network. Often the network is a shared resource that is contended for by several processing units. For instance the network can be a data bus that handles various data streams. Usually, the individual computing nodes are not synchronized and communicate via message passing. They have a high degree of independency and make autonomous decisions concerning resource access and scheduling. Therefore, it is a particularly complex task to maintain global state information.

In addition, in many cases each computing node has an individual functionality and a particular local environment. Thus, each node is provided with specific and adapted hardware resources, which leads to highly heterogeneous implementations. For instance in a car the embedded control units responsible for engine control, power train control and stability control have a highly heterogeneous composition.

Based on the above description, it becomes clear that heterogeneous and distributed embedded systems are inherently complex to design and analyze.

2.2 Performance metrics

In [15] Marwedel indicates the following five metrics for the evaluation of the efficiency of an embedded system:

- Power consumption
- Code-size
- Run-time efficiency[1]
- Weight
- Cost

All these metrics can be subject to design requirements of the system. Therefore, in order to evaluate admissible system implementations and drive the state space exploration, appropriate predictions of these system characteristics are required in early design stages. Hence, in a broad sense, they can all be considered objectives of early performance analysis.

[1] comprehends the amount of allocated hardware resources, clock frequencies and supply voltages

However, in most of the performance analysis methods for real-time embedded systems the focus is on the analysis of timing aspects. In particular, the designer is interested in best-case and worst-case delays, as he needs to know if the end-to-end latencies of the designed system meet the real-time requirements.

Further, in most embedded systems it is not acceptable that portions of a data stream are lost or retained due to buffer overflows. Therefore, it is essential for the designer to know if the allocated data buffers between the various computing nodes of the distributed system are large enough to contain the worst-case amount of data. In other words, he is interested in the worst-case buffer fill levels of the designed system.

On the basis of the above requirements, most of the performance analysis methods that can be found in the literature for real-time systems are targeted at one or both of the following performance metrics:

- End-to-end delays
- Memory requirements (in terms of buffer spaces)

2.3 Requirements for performance analysis methods

An ideal performance analysis method is accurate, fast and easy to use. There are, however, also other requirements that a performance analysis technique should fulfill. In this section we pick up and extend the list of requirements for performance analysis methods given in [25].

Correctness

A performance analysis approach should produce correct results. In the case of hard real-time embedded systems correct means that the determined results are hard upper (lower) bounds for the worst-case (best-case) performance of the system. In other words, there are no reachable system states such that the determined bounds are violated.

Accuracy

The calculated upper (lower) performance bounds should be close to the actual worst-case (best-case) performance of the system.

Analysis effort

The analysis of a distributed embedded system should be cheap in terms of analysis time and computational effort. This is particularly important if the performance analysis is used to drive a design space exploration. In this case the performance analysis is executed repeatedly in order to

evaluate several different system implementations and a short analysis time for a single evaluation is essential.

Modeling effort

A designer should be able to set up a system model for performance analysis with little effort. Also, an existing model should be easy to reconfigure, which is fundamental for design space exploration.

Reusability

An ideal performance analysis method should be reusable across different abstraction levels. This means that it should permit designers to model and analyze a system with different levels of detail. In particular, it should be easy to refine an existing system model.

Modularity

Performance analysis methods are not necessarily required to be modular (in the next chapter we will also consider holistic approaches), but modularity can have several advantages. For instance modular performance analysis methods are typically more scalable and easier to reconfigure than holistic approaches, because they permit designers to model and analyze a system by composing several smaller, ideally pre-built system modules. However, often the modularity is paid for by reduced analysis accuracy.

We would like to point out that the discussed requirements form a sort of "wish list" for attributes of performance analysis methods, i.e. they concern an ideal method. As we will show in the following chapters, none of the approaches considered satisfies all these requirements. In particular, different requirements may be conflicting for some performance analysis methods. For instance, a performance analysis method might be able to achieve more accurate results when performing a more time-consuming analysis.

Moreover, depending on the application domain and the design approach, the different requirements are usually not equally relevant.

Chapter 3

Approaches to performance analysis

There are several different approaches to analyzing or estimating the performance of distributed embedded systems. In this chapter we describe a number of methods for performance analysis that can be found in the literature. In particular we focus on system level performance analysis in early design stages.

3.1 Classification

An important distinction is drawn between formal performance analysis methods and simulation based approaches. The former determine hard guarantees of the performance of a system, while the latter cannot usually provide hard performance bounds due to insufficient corner case coverage. There are also stochastic methods for performance analysis which we will not consider in the context of this thesis.

Further, one can distinguish modular and holistic performance analysis methods. The modular approaches analyze the performance of single components of the system and propagate the results in order to determine the performance of the entire system. In contrast, the holistic approaches consider the system as a whole. Holistic methods can generally provide more accurate results, as the behavior of the entire system is analyzed at once. However, this can also considerably increase the complexity of the analysis.

3.2 Simulation based methods

The use of simulation for performance verification of distributed embedded systems is state of the art in industry. Several tools are available for cycle accurate hardware-software co-simulation; see e.g. [7, 31, 24]. In addition, the use of SystemC [19], a widespread platform for system-level modeling and simulation, is very common. Many simulation methods are trace-based, i.e. the

system designer provides traces of input stimuli that drive the simulation of the modeled system.

The main advantage of simulation is the large modeling scope. In contrast to formal analysis methods, basically every system can be modeled, as many dynamic and complex interactions can be taken into account for the simulation. Moreover, in most of the cases the functioning and performance of a system can be verified with the same simulation environment, simulation traces and benchmarks.

Another advantage of simulation based methods is that usually they are reusable over different abstraction levels, as the simulation models can be refined. In other words, the level of abstraction for the simulation can be adapted to the required degree of accuracy.

However, hardware-software co-simulations are often computationally complex and have long running times. Therefore, performance estimation quickly becomes a bottleneck in the design process, especially if it is used to drive a design space exploration.

The fundamental problem of simulation based performance analysis methods is the insufficient corner case coverage. Usually an exhaustive simulation of all the system states is not feasible due to the enormous state space. Thus, in order to determine the best-case or worst-case performance of a system through simulation, the system designer must provide use cases in the form of simulation traces that lead to the corresponding corner cases. However, this becomes practically impossible with increasing system complexity.

The following example adapted from [10] shows that finding simulation patterns that lead to corner cases is already challenging for apparently trivial distributed embedded systems. The system architecture is represented in Figure 3.1. In application A1 a task P1 of the CPU reads periodically data bursts from the sensor and stores the data in the memory. A second task P2 reads the data from the memory, processes it and transfers it to an output device via the shared bus. The task P2 has a best case execution time BCET and a worst case execution time WCET. We suppose that the CPU implements static fixed priority scheduling and that P1 has higher priority than P2. In the second application A2 a task P4 running on the input interface periodically sends data packets to the DSP over the shared bus. Task P5 on the DSP stores the data packets into the buffer. A second task P6 periodically removes data packets from the buffer, e.g. for playback. We suppose that the bus uses a first come first serve scheme for arbitration. As the two data streams of A1 and A2 interfere on the shared bus, there will be a jitter in the packet stream received

by the DSP that may lead to an underflow or overflow of the buffer.

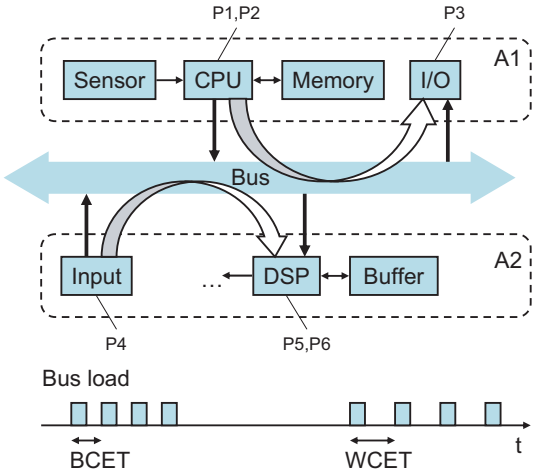


Figure 3.1: Interference of two data streams on a shared communication resource

The interesting property of this system is that the DSP experiences the worst case input jitter when P2 executes continuously with its BCET. The reason is that in this case the distance between the packets of A1 on the bus is shortest and thus the transient bus load is highest. In other words, the worst case execution of A2 coincides with the best case execution of A1.

The designer must perceive this system particularity in order to provide a simulation trace that reaches the corner case. In case of larger and more realistic systems, several computation and communication resources will be shared simultaneously, there may be different scheduling policies for the various resources and data/control dependencies will play a role. In short, the corner cases will be extremely difficult to find.

Hence, simulation based methods are not suited to determining hard performance bounds of a general distributed embedded system. Nevertheless, simulative approaches can be useful to estimate the average system performance.

Moreover, it can be advantageous to combine simulation with formal performance analysis approaches: if the performance estimates provided by simulation are close to the results determined by an analytical method, it means that the calculated performance bound is accurate. In other words, simulation may be helpful to evaluate the accuracy of formal performance analysis. However, note that if the simulation and analysis results are distant, no conclusion about the accuracy is possible: either a too pessimistic performance analysis or a too optimistic performance simulation can be the cause.

3.3 Holistic scheduling

There is a large body of literature on scheduling of tasks on shared computing resources. In particular, in the real-time domain the research is focused on the analysis of schedulability and worst case response times of tasks. Examples of scheduling algorithms are fixed priority, rate monotonic, earliest deadline first, round robin and TDMA. Detailed information about the various scheduling policies and the corresponding analyses can be found in [5], as well as in many other books on the topic.

Several proposals have been made to extend concepts of the classical scheduling theory to distributed systems. In such systems the applications are executed on several computing nodes and the delays caused by the shared use of communication resources cannot be neglected. In particular the integration of process and communication scheduling is often denoted as holistic scheduling. Rather than a specific performance analysis method, holistic scheduling is a group of techniques for the analysis of distributed embedded systems.

Each analysis technique is focused on a particular input event model and resource sharing policy. This permits a detailed analysis of the temporal behavior of a system and leads to accurate performance predictions. Nevertheless, the modeling scope of the analysis techniques is restricted to a particular class of systems, i.e. the holistic approaches do not scale to general distributed architectures. For every new kind of input event model, communication protocol, resource sharing policy and combinations thereof, a new analysis method needs to be developed.

A number of holistic analysis techniques can be found in the literature. For instance in [27] Tindell and Clark combine fixed priority scheduling on the processing resources of a distributed system with TDMA scheduling on communication resources. In [22] Pop, Eles and Peng analyze mixed event triggered and time triggered task sets that communicate over protocols with both static and dynamic phases (e.g. FlexRay).

In this section we briefly describe the holistic analysis approach presented by Tindell and Clark as well as improvements of the analysis for systems with data dependencies (Yen, Wolf) and control dependencies (Pop, Eles, Peng). Moreover, we describe the MAST tool (Gonzalez Harbour *et al.*), an analysis suite that implements several holistic techniques.

3.3.1 Schedulability analysis for distributed systems

In [27] Tindell and Clark propose an extension of the static priority preemptive scheduling analysis to address the wider problem of scheduling analysis for distributed systems. In particular, they derive an analysis for systems in which tasks with arbitrary deadlines communicate via message passing over a communication network implementing the TDMA protocol.

The starting point is the equation to compute the worst-case response time of a given task i on a shared processor, assuming periodic task activations and static priority preemptive scheduling [13, 11]:

$$r_i = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{r_i}{T_j} \right\rceil C_j \tag{3.1}$$

where r_i is the worst-case response time of a given task i, $hp(i)$ is the set of all tasks of higher priority than task i, C_i ist the worst-case execution time of task i and T_j is the period of task j. This equation is valid under the assumption that the deadline of a task i is less than its period T_i and can be solved by iteration (a suitable initial value for r_i is 0).

Tindell and Clark contribute to two extensions of the above analysis:

1. They extend the analysis to the case with arbitrary deadlines
2. They take into account the release jitter J_i of processes, i.e. the worst-case time between the arrival of a process and its release

The resulting analysis of the worst-case response time of a task i is given by the following equations:

$$r_i = \max_{q=0,1,2,\ldots} (J_i + w_i(q) - qT_i) \tag{3.2}$$

$$w_i(q) = (q+1)C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{J_i + w_i(q)}{T_j} \right\rceil C_j \tag{3.3}$$

The sequence of values of q in the first equation is finite since only values of q where $w_i(q) > (q+1)T_i$ need to be considered.

The major achievement of Tindell and Clark is the adaptation of the above processor schedulability analysis to a communication system. In particular, they apply the same family of analysis to the bounding of message delays across a TDMA broadcast bus. For the sake of conciseness, we do not report the corresponding equations and refer the reader to [27].

Finally, Tindell and Clark integrate the analysis of the worst-case timing of tasks with the analysis of the worst-case timing of messages. The basic concept

is to interpret the message delay induced by the communication system as release jitter of the receiver task. The result is a holistic analysis method for distributed systems.

However, the holistic scheduling equations cannot usually be trivially solved due to mutual dependencies. For instance the release jitter of a receiver task depends on the arrival time of the corresponding message, which in turn depends on the interference from higher priority messages, which in turn depends on the release jitter of sender tasks.

This example points out an important property that holds for most of the holistic analysis methods: in general the complexity of the model grows with the size of the system.

3.3.2 Performance analysis for systems with data dependencies

In [32] Yen and Wolf introduce an analysis algorithm for the execution time of an application on a distributed system. In particular they extend the classical analysis for static priority preemptive scheduling (equation 3.1) in order to exploit data dependencies that exist among the tasks of a task graph. This permits to determine tighter performance bounds. Basically, the extended analysis takes into account that the delays through a path of tasks forming a task graph are not independent.

The example of Figure 3.2 adopted from [32] illustrates the effects of data dependencies on the execution delay[1]: if the data dependency between T2 and T3 is ignored, their worst-case response times are $35\,ms$ and $45\,ms$, respectively. Thus, the classical analysis assumes a worst-case delay of $80\,ms$ for the execution of the task sequence T2-T3. However, the worst-case delay for this task sequence is actually $45\,ms$, because

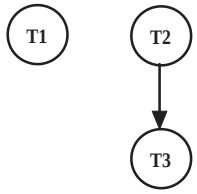
1. T1 can only preempt either T2 or T3, but not both in a single execution

2. T2 cannot preempt T3

as can be easily traced considering the periods and WCETs of the system specification.

To detect and exploit properties like the first one in the above example, Yen and Wolf introduce the concept of *phases* among task activations and extend

[1]The example is considered also in Section 6.7 where a more detailed analysis can be found



Task	Period	WCET	Priority
T1	80	15	high
T2	50	20	medium
T3	50	10	low

Figure 3.2: Effect of data dependencies on the execution delay. The three tasks form two task graphs and share the same CPU which implements preemptive fixed priority scheduling.

the analysis of equation 3.1 as follows[2]:

$$r_i = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{r_i - \phi_{ij}}{T_j} \right\rceil C_j \tag{3.4}$$

where the phase ϕ_{ij} is the smallest interval for the next activation of a preempting task j relative to the activation of a task i. In particular the phases among the tasks are computed iteratively by a fixed point iteration: starting with initial phase values the response time analysis is used to derive better phase values which in turn allow a more accurate response time analysis etc.

To handle conditions like the second one in the above example, Yen and Wolf introduce a so called separation analysis that allows to verify whether the executions of two tasks can overlap or not.

For the sake of conciseness we do not report the two algorithms to compute the phases and separations of tasks and refer the interested reader to [32].

3.3.3 Performance analysis for systems with control dependencies

In [21] Pop, Eles and Peng present an extension of the previous analysis that takes into account the control dependencies among the tasks of an application. In systems with control dependencies, depending on conditions, only a subset of the tasks is executed during a system invocation. As for data dependencies, the consideration of control dependencies can significantly reduce the pessimism of the performance analysis.

In particular, in [21] the authors introduce so-called conditional process graphs (CPG) as models for applications and analyze their delay. Figure 3.3 shows an example of a system model consisting of two CPGs.

Three approaches are proposed to analyze the delay of a CPG:

[2] In [26] Tindell presents a similar concept of *time offsets* to exploit data dependencies

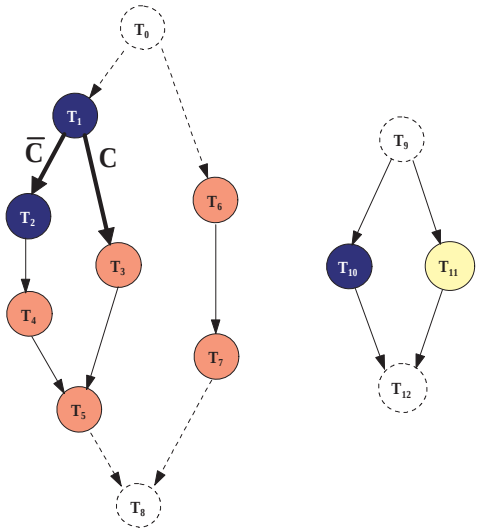


Figure 3.3: Example of system model consisting of two CPGs. The execution of T2/T4 and T3 depends on the condition C determined by T1. The tasks are mapped on three different processors as indicated by the shading.

Brute force solution

The CPG is decomposed into all its constituent unconditional subgraphs and each of these subgraphs is analyzed as presented in the previous section. This approach provides a tight bound on the delay, but can be very expensive in terms of analysis effort, as in general the number of unconditional subgraphs can grow exponentially with the number of tasks.

Condition separation

The knowledge about the conditions is used only in order to refine the separation analysis of the previous section. This analysis is more pessimistic than the brute force solution, however it reduces the analysis effort significantly.

Relaxed tightness analysis

This approach is similar to the brute force solution, as the same number of unconditional subgraphs must be analyzed. However, the analysis effort is reduced significantly by the substitution of the original analysis algorithms with less complex approximation variants. Of course, this simplification is paid for by reduced analysis accuracy.

For the corresponding analysis algorithms we refer the reader to [21].

3.3.4 The Modeling and Analysis Suite for Real-Time Applications (MAST)

An important contribution to enhance, implement and aggregate several scheduling analysis techniques has been made by the research group of Gonzalez Harbour at the University of Cantabria. This group has realized the MAST suite[3][14], an open source set of software tools for the schedulability analysis of real-time applications. It aggregates several scheduling analysis techniques for mono-processor and distributed systems. In particular the MAST tool

- implements offset-based scheduling analysis techniques
- can model complex dependence patterns among the tasks of an application (for instance multiple event task activation)
- supports hierarchical scheduling
- supports several input event models (for instance periodic, sporadic, or bursty event streams)
- can compute optimal priority assignments to tasks
- can compute priority ceilings and preemption levels for shared resources
- can analyze the resource load and compute several slack values

For the evaluation of performance analysis methods conducted in the next chapters we will use the MAST tool as representative of holistic performance analysis methods.

3.4 Compositional scheduling analysis using standard event models

In [23, 10] Richter *et al.* propose a modular performance analysis approach for distributed embedded systems based on the results of classical real-time scheduling. The approach is denominated SymTA/S, which stands for Symbolic Timing Analysis for Systems.[4]

[3] *http://mast.unican.es/*

[4] The SymTA/S approach is fully implemented in a software tool distributed under the same name. In this thesis we will use the term *SymTA/S* for both the analysis approach itself and the corresponding tool.

In the following subsections we first report a number of well known abstractions for event arrival patterns on which the SymTA/S analysis is based. Then, we briefly describe the analysis approach itself and some recent extensions.

3.4.1 Standard event models

The behavior of the environment of a distributed embedded system is often modeled using common abstractions for event arrival patterns. These abstractions include periodic or sporadic event arrivals with potential jitters or bursts. In the context of SymTA/S these models are denoted as standard event models.

Periodic event stream

In a periodic event stream with period P the events arrive at intervals of exactly P time units. Figure 3.4 depicts a periodic arrival pattern.

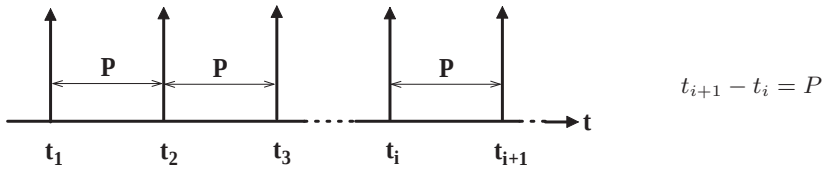


Figure 3.4: Periodic event stream

Periodic event stream with jitter

In the periodic event stream model with jitter the events arrive at an average time interval of P time units, but may have a local deviation around the ideal periodic arrival. The deviation is bounded by an interval of length $J \leq P$. This is represented in Figure 3.5, where the intervals of admissible arrival times are represented as shaded rectangles.

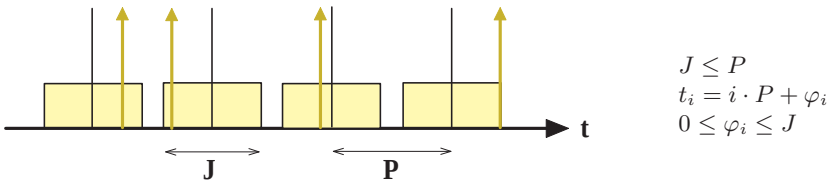
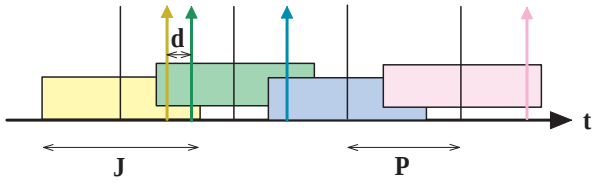


Figure 3.5: Periodic event stream with jitter. In each jitter interval (shaded rectangle) exactly one event arrives.

Periodic event stream with burst

Events may arrive in bursts, if the deviation from the ideal periodic arrival time is larger than the period. In this case the admissible arrival time intervals of adjacent periods overlap, as depicted in Figure 3.6. However, events cannot overtake each other: the arrival time of an event is restricted by the arrival time of previous events. In particular an event may arrive only d time units (d = minimum event inter-arrival time, $d \geq 0$) after the arrival of the event belonging to the previous period.



$J > P$
$t_i = i \cdot P + \varphi_i$
$0 \leq \varphi_i \leq J$
$t_{i+1} - t_i \geq d$

Figure 3.6: Periodic event stream with burst

The standard event models also include the sporadic variants of the three models described above. Basically they are identical to the periodic variants with the difference that single event arrivals may be left out.

3.4.2 The SymTA/S analysis approach

The main goal of the SymTA/S analysis approach is to exploit the host of work on mono-processor real-time scheduling analysis for the performance analysis of distributed embedded systems. While holistic methods attempt to extend classical scheduling analysis to special classes of distributed systems, SymTA/S applies existing analysis techniques in a modular manner: the single modules of a distributed system are analyzed with classical algorithms and the local results are propagated among the system through appropriate interfaces.

The advantage of this approach is that it does not require the development of new scheduling analysis algorithms. However, all the event streams in the system must fit the basic models for which scheduling analysis techniques are available. In particular, the output event stream of a component must be converted to an event stream model that is compatible with the scheduling analysis performed on the next component. SymTA/S provides interfaces to convert standard event stream models among each other. These interfaces can be grouped into two types:

Event Model Interfaces (EMIFs)

Event Model Interfaces do not change the actual timing properties of an

event stream. Only the mathematical representation of the stream, i.e. the underlying model is converted. Such transformations require that the parameters of the target model encompass the timing of any possible event sequence in the source model.

Event Adaptation Functions (EAFs)

Event Adaptation Functions need to be used in cases where an EMIF transformation is not possible. In this case the timing properties of the stream must be adapted to fit the requested model. In particular this requires to change the implementation of the system, e.g. by adding appropriate event buffers.

For instance it is possible to convert a periodic event stream with jitter X to a sporadic event stream Y by a simple EMIF: If X is characterized by a period P_X and a jitter J_X and Y is characterized by a minimum interarrival time t_Y, the corresponding EMIF is represented by the equation $t_Y = P_X - J_X$. However, this transformation comports a loss of information, as the stream Y also comprehends event sequences that cannot occur according to the event stream model X.

An example of an interface that requires an EAF is the conversion of a periodic event stream with burst to a pure periodic event stream. In particular it is necessary to add an appropriate buffer to smooth out the bursts.

Figure 3.7 gives an overview of the event model interfaces adopted by SymTA/S.

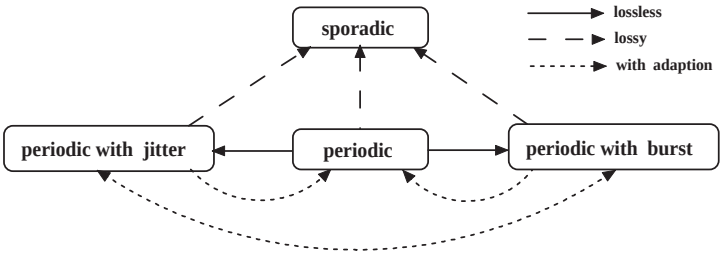


Figure 3.7: Event model interfaces in SymTA/S

The event stream interface technology described above permits to analyze the performance of a distributed embedded system by applying classical scheduling analysis algorithms to local components. Figure 3.8 illustrates the overall analysis principle. First the environmental timing assertions are applied to components connected to the system inputs. Then, these components are analyzed to derive local delays and buffer requirements, as well as the corresponding output event models. These output event models are mapped to the input event

models of the connected components using appropriate EMIFs or EAFs. In pure feed-forward systems this procedure is simply repeated for all the components until the event streams are propagated through the whole system and global end-to-end delays and buffer sizes can be determined.

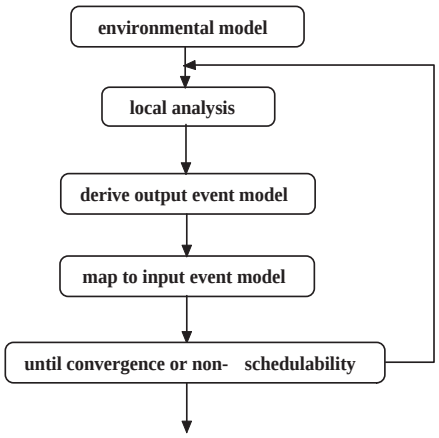


Figure 3.8: Analysis principle of SymTA/S

For systems with functional cycles (i.e. systems with feedback) or systems with non-functional cyclic dependencies the timing of two or more components is mutually dependent. In this case the event streams are propagated iteratively until the event stream parameters converge or the tasks of a resource are no longer schedulable.

3.4.3 Extensions

Several extensions have been worked out for the analysis approach described above. For instance the SymTA/S analysis approach can deal with multiple task activation. This means that the tasks of a system can be triggered by multiple inputs in AND- or OR-combination. Moreover, the approach is able to take into account system context information in order to reduce analysis pessimism. In particular, the analysis methods support the exploitation of two kinds of context information:

Intra-event stream context

This kind of context information considers the correlations between successive computation or communication requests in an event stream. In particular, the events of a stream can have different types and impose different workloads on the activated tasks according to their type. Correlations within a sequence of different activating events can be described by means of appropriate intra-context information. This information can ei-

ther describe exactly the sequence of activation events (e.g. by specifying a periodically recurring pattern of event types) or be partially incomplete (e.g. by specifying minimum and maximum number of occurrences for a certain event type in an event sequence of a given length).

Inter-event stream context

This kind of context information considers timing correlations between events in different event streams. In particular, while context-blind analysis assumes that all tasks sharing a resource are independent and can all be activated simultaneously, this might not be possible due to timing correlations among event streams. For instance, such correlations may result from data dependencies (see Section 3.3.2) and can be expressed by appropriate activation offsets.

More detailed information about the various extensions to the SymTA/S performance analysis approach can be found in [10].

3.5 Modular Performance Analysis with Real Time Calculus

Modular Performance Analysis with Real Time Calculus (MPA-RTC) [6] is a framework for performance analysis of distributed embedded systems that has its roots in network calculus [4], a theory of deterministic queuing systems for communication networks. MPA-RTC analyzes the flow of event streams through a network of computation and communication resources in order to derive performance characteristics of a distributed embedded system.

MPA-RTC is a modular approach to performance analysis. It permits to analyze large systems by composing basic analysis components to performance models. In contrast to the SymTA/S approach, MPA-RTC is not restricted to a few classes of input event models. In particular it permits to model any event stream using so-called arrival curves. A similar abstraction, the so-called service curves, is used to model the availability of computation or communication resources. Service curves are first-class citizen in the MPA-RTC approach and permit to model basically any form of resource availability. This differentiates MPA-RTC from other performance analysis methods, that are usually restricted to a few common models of resource availability.

In the following subsections we first describe the concepts of arrival and service curves (denoted together as variability characterization curves) on which MPA-RTC is based. Subsequently, we describe the analysis approach itself and some recent extensions.

3.5.1 Variability characterization curves

In MPA-RTC the timing characterization of event and resource streams is based on variability characterization curves which basically generalize the classical representations such as sporadic, periodic or periodic with jitter.

Event streams are described using arrival curves $\bar{\alpha}^u(\Delta)$, $\bar{\alpha}^l(\Delta) \in \mathbb{R}^{\geq 0}$, $\Delta \in \mathbb{R}^{\geq 0}$ which provide upper and lower bounds on the number of events in *any* time interval of length Δ. In particular, if $R[s,t)$ denotes the number of events that arrive in the time interval $[s,t)$, then the following inequality is satisfied:

$$\bar{\alpha}^l(t-s) \leq R[s,t) \leq \bar{\alpha}^u(t-s) \quad \forall s < t \tag{3.5}$$

where $\bar{\alpha}^l(0) = \bar{\alpha}^u(0) = 0$. The timing information of the standard event models can easily be represented by appropriate pairs of upper and lower arrival curves [6]. For instance Figure 3.9 depicts the upper and lower arrival curves of the class of event streams with period P and jitter J.

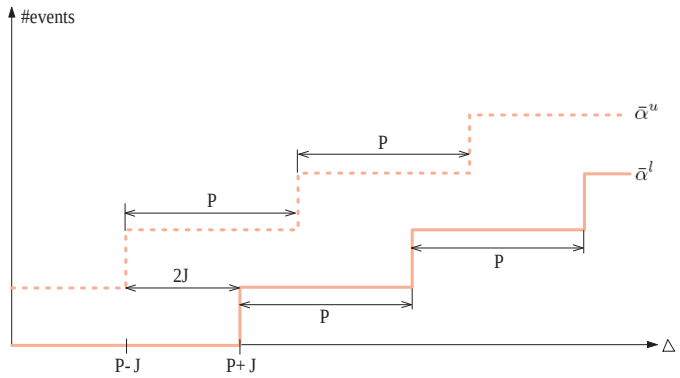


Figure 3.9: The upper and lower arrival curves of an event stream with period P and jitter J

The arrival curves are much more general than the standard event models: any deterministic event stream can be modeled by an appropriate pair of arrival curves. The curves can be constructed analytically, if the event stream pattern is completely defined. Alternatively they can be derived from a finite set of event traces. This can be done easily by using a sliding window of size Δ and determining the minimum and maximum number of events within the window.

In a similar way, resource streams are described using service curves $\beta^u(\Delta)$, $\beta^l(\Delta) \in \mathbb{R}^{\geq 0}$, $\Delta \in \mathbb{R}^{\geq 0}$ which provide upper and lower bounds on the available service in any time interval of length Δ. The service is expressed in an appropriate unit, for instance number of cycles for computing resources or bytes for communication resources. In particular, if $C[s,t)$ denotes the number of

processing or communication units available from the resource over the time interval $[s,t)$, then the following inequality holds:

$$\beta^l(t-s) \leq C[s,t) \leq \beta^u(t-s) \quad \forall s < t \tag{3.6}$$

Again, there are no restrictions for the representable resource models. With an appropriate pair of service curves, any deterministic resource availability can be modeled. For instance Figure 3.10 depicts the upper and lower service curves for a slot of a time units that permits the transmission of b bytes on a TDMA resource with period Q. Moreover, service curves enable the modeling of hierarchical scheduling.

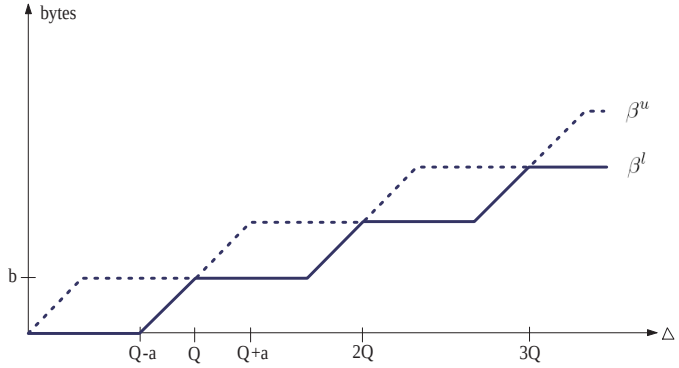


Figure 3.10: The upper and lower service curves for a slot on a TDMA resource

Note that in the above definitions $\bar{\alpha}^l(\Delta)$ and $\bar{\alpha}^u(\Delta)$ are expressed in terms of events (this is marked by a bar on the α), while $\beta^l(\Delta)$ and $\beta^u(\Delta)$ are expressed in terms of workload/service units. However, the analysis described in the next subsection requires the arrival and service curves to be expressed in the same unit. The transformation of event-based curves into workload/resource-based curves and vice versa is done by means of so called workload curves. Basically these define the minimum and maximum workload imposed on a resource by a given number of succeeding events, i.e. they capture the variability in execution demands. The interested reader can find more information about workload curves in [16].

3.5.2 Analysis and resource sharing

In this subsection we describe how MPA-RTC models the processing of event streams by computation and communication resources. In particular we describe how the outgoing event and resource streams of a processing component are derived from the ingoing event and resource streams.

Figure 3.11 shows a so called Real Time Calculus abstract processing component that models the processing of an event stream by an application process. In particular, an incoming event stream represented as a pair of arrival curves α^l and α^u, flows into a FIFO buffer in front of the processing component. The component is triggered by these events and will process them in a greedy manner while being restricted by the availability of resources, which are represented by a pair of service curves β^l and β^u. On its output, the component generates an outgoing stream of processed events, represented by a pair of arrival curves $\alpha^{l'}$ and $\alpha^{u'}$. Resources left over by the component are made available again on the resource output and are represented by a pair of service curves $\beta^{l'}$ and $\beta^{u'}$.

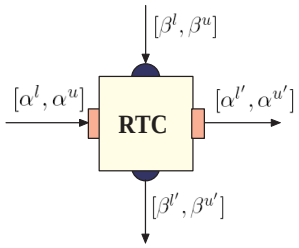


Figure 3.11: Real Time Calculus processing component

The transformation of input arrival and service curves to output arrival and service curves is described by the following set of equations:

$$\alpha^{l'}(\Delta) = \min\{\inf_{0\le\mu\le\Delta}\{\sup_{\lambda\ge 0}\{\alpha^l(\mu+\lambda)-\beta^u(\lambda)\}+\beta^l(\Delta-\mu)\}, \beta^l(\Delta)\} \quad (3.7)$$

$$\alpha^{u'}(\Delta) = \min\{\sup_{\lambda\ge 0}\{\inf_{0\le\mu<\lambda+\Delta}\{\alpha^u(\mu)+\beta^u(\lambda+\Delta-\mu)\}-\beta^l(\lambda)\}, \beta^u(\Delta)\} \quad (3.8)$$

$$\beta^{l'}(\Delta) = \sup_{0\le\lambda\le\Delta}\{\min\{\inf_{0\le\mu\le\Delta}\{\sup_{\lambda\ge 0}-\alpha^u(\lambda)\}\} \quad (3.9)$$

$$\beta^{u'}(\Delta) = \max\{\inf_{\lambda\ge\Delta}\{\beta^u(\lambda)-\alpha^l(\lambda)\}, 0\} \quad (3.10)$$

The processing components can be freely combined to form performance models of distributed embedded systems. For instance in order to model the sequential processing of an event stream by two tasks, it is sufficient to connect two processing components in series so that the outgoing event stream of the first one is the ingoing event stream of the second one.

Scheduling policies on shared resources can be modeled by the way processing components are linked and resource streams are distributed among them. For instance Figure 3.12(a) shows how to connect two performance components in order to model a resource that implements preemptive fixed priority scheduling: the task T_B has lower priority than T_A and thus gets only the resource

service that is left after $T_A$has been served. Figure 3.12(b) shows the modeling of a proportional share policy. Many other scheduling strategies as for instance FCFS, TDMA or EDF can be modeled by distributing resource streams properly.

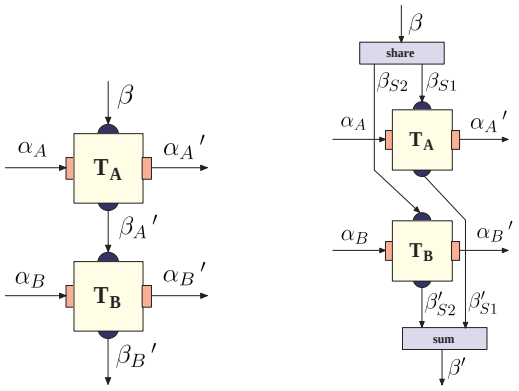


Figure 3.12: Real Time Calculus models for Fixed Priority and Proportional Share scheduling

The performance analysis of a distributed embedded system is done by combining the analysis of the single processing components of a performance model. In particular, the maximum delay experienced by an event at a system module and the maximum number of events that are waiting to be processed can be bounded by the following inequalities:

$$delay \leq \sup_{t \geq 0} \{ \inf \{ \tau \geq 0 : \alpha^u(t) \leq \beta^l(t+\tau) \} \} \tag{3.11}$$

$$backlog \leq \sup_{t \geq 0} \{ \alpha^u(t) - \beta^l(t) \} \tag{3.12}$$

The maximum delay and backlog experienced at a processing component correspond to the maximal horizontal and vertical distance between α^u and β^l, respectively, as depicted in Figure 3.13.

The end-to-end delay experienced by an event at the complete system is computed as the sum of the single delays at the various processing components. However, the analysis does not necessarily need to be strictly modular. For instance a holistic delay analysis that considers the combined action of several processing components in series is also feasible.

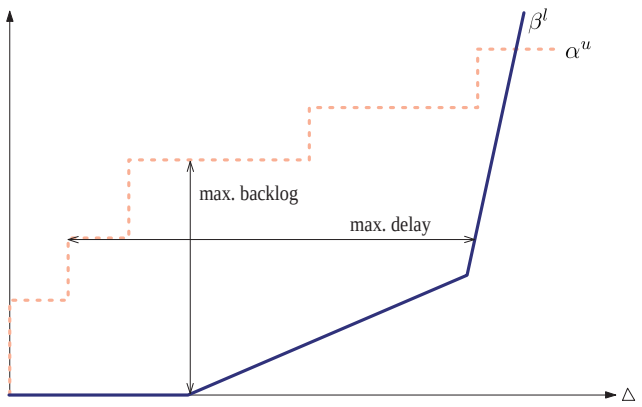


Figure 3.13: Graphical interpretation of maximum delay and backlog

3.5.3 Extensions

Several extensions have been worked out to refine the MPA-RTC analysis approach. The following list cites three examples:

- [28] proposes an abstract stream model for the characterization of streams with different event types that impose different workloads on the system. It permits considerable improvements of the worst-case performance analysis for systems with type related workload.

- [29] presents abstract models for system components, which permit to capture complex functional properties of systems, as for example caches, variable resource demands and arbitrary up- and down-sampling of event streams in a system component.

- [30] introduces a model to characterize and capture the correlation of different resource demands that events of a given type cause on different system components. The exploitation of such so-called workload correlations can lead to considerably improved analysis results (see case study in Section 6.6).

Finally, we would like to point out that the described modular performance analysis framework is not necessarily bound to the use of Real Time Calculus. Instead, any abstraction of event streams and resource characterization can be used. It is sufficient to change the computations that are done within the processing components appropriately.

3.6 Timed automata based performance analysis

The use of formal methods for the design and analysis of real-time systems has driven research for many years. Several different formal approaches can be used to specify a system and verify its correctness. [8] gives an overview of available formalisms for the design and analysis of real-time computing systems.

Timed automata [1] are one popular formalism for the specification of real-time systems. They can be used in combination with a logic language to verify system properties by model checking. In particular the UPPAAL tool environment[5] [3] allows users to validate and verify real-time systems modeled as networks of timed automata.

In [17] Yi *et al.* have shown that the schedulability analysis of an event-driven system can be represented as a reachability problem for timed automata and thus can be tackled with model checking. In particular, timed automata based schedulability analysis is implemented in the TIMES tool[6] [2]. TIMES permits users to analyze systems that are described as a set of tasks which are triggered either periodically or by external event streams modeled through appropriate timed automata. However, the TIMES tool is limited to the schedulability analysis of single processors. Thus, it is not suited for performance analysis of distributed systems.

Recently Hendriks and Verhoef have presented an approach to performance analysis of distributed embedded systems based on the model checking of timed automata networks [9]. In this section we briefly describe the fundamental concept and the application of their approach.

Basically, the idea is to model the environment and the resources of a system as timed automata. The various components are then composed into a network of timed automata that models a distributed embedded system. The performance properties of the system are verified through exhaustive model checking. In particular, UPPAAL is used for the modeling and verification of timed automata networks.

In the following subsections we describe some timed automata models for input event streams and hardware resources that have been proposed so far in the context of this analysis approach. Afterwards we show how the different components can be aggregated to model a distributed embedded system and how the performance analysis is realized. We would like to point out that the analysis approach is not restricted to the component models described. In

[5] available at *http://www.uppaal.com*
[6] available at *http://www.timestool.com*

particular, new timed automata models for other types of event streams and resource sharing policies can be designed, making extensibility one of the major benefits of this approach.

3.6.1 Modeling the environment

Several timed automata models have been proposed to represent different input event streams. In particular, for all the standard event models (see Section 3.4.1) corresponding timed automata templates have been designed. For instance Figure 3.14 shows a timed automaton that models a periodic event stream with period P. After an undefined initial offset the automaton generates events at intervals of exactly P time units. The generation of an event is modeled by the increment of the global variable *req*. Figure 3.15 depicts a timed automaton presented in [20] that models a periodic event stream with jitter $J \leq P$.

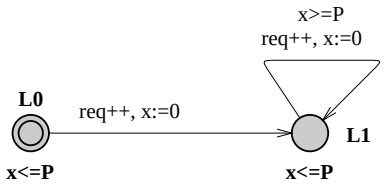


Figure 3.14: Timed automata model for a periodic event stream

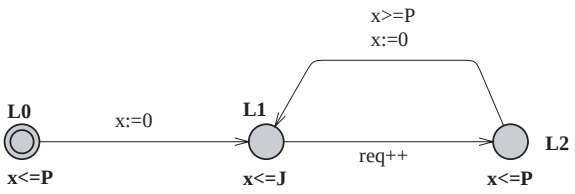


Figure 3.15: Timed automata model for a periodic event stream with jitter

The automaton for a periodic event stream with burst can be found in [9]. As we stated above, new event stream models can be designed easily. Basically any deterministic event stream can be modeled.

3.6.2 Modeling the hardware resources

Each processing component is modeled as a separate timed automaton. A processing component is either idle or busy computing some function. Similarly, each communication link is modeled as a timed automaton. Each link is either idle or transporting some data. For shared resources the adopted scheduling

policy determines the structure of the model. For instance Figure 3.16 shows a timed automaton that models a hardware resource with two tasks implementing preemptive fixed priority scheduling. The resource can either be idle or process T1 or process T2. The location *pre_T1* models the fact that T1 can preempt T2. The *hurry!* synchronization models a so-called urgent edge (see [3] for details) and makes sure that the corresponding edge is taken as soon as it is enabled.

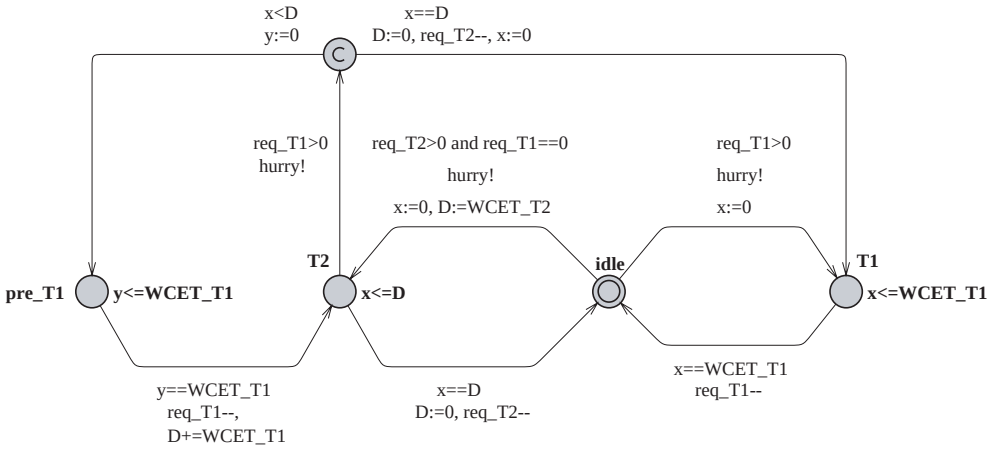


Figure 3.16: Timed automata model for a preemptive FP resource with two tasks

Several other resource sharing strategies can be modeled with appropriate timed automata. For instance in [20] we have presented a solution for a TDMA policy.

3.6.3 Performance analysis

The timed automata models of the single system components are aggregated into a timed automata network that represents a distributed embedded system. The single components interact via global variables and channels. For instance suppose that the timed automaton of an input event generator increments a global variable *req* to model the request of a task activation on a certain resource. The timed automaton that models the corresponding resource is sensitive to increments of the variable *req* and immediately starts the execution of the corresponding task if no higher priority task has to be executed. The completion of the task execution is modeled by the decrement of the variable *req*. Let's suppose that the corresponding output event triggers a second task. This can be modeled by incrementing a second global variable *req2* simultaneously with the decrement of *req*. Again, another automaton will be sensitive to

the increments of *req2*, start the corresponding task and so on. In this way the propagation of events through the distributed system can be easily modeled.

The performance attributes of a distributed embedded system are derived by verifying properties of the corresponding timed automata network. For instance, to ensure that the maximum backlog of a certain task does not exceed a given value b, it is sufficient to verify the following property by model checking:

$$AG \quad (req \leq b)$$

where 'AG' stands for 'always generally' (= invariantly) and *req* is the global variable that counts the activation requests of the corresponding task. In particular it is possible to derive the *exact* maximum backlog by finding the smallest b that satisfies the above property. This can be done by using a binary search strategy.

The verification of end-to-end delays is a little more involved as it requires to adapt the timed automata models of the corresponding input event generators. For instance Figure 3.17 shows the variant of a periodic event stream generator that permits to verify end-to-end latencies.

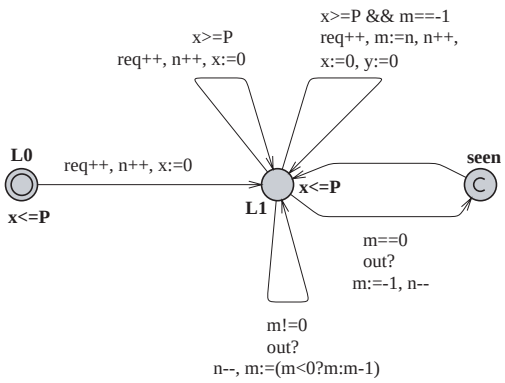


Figure 3.17: Timed automata model for a periodic input generator that measures the end-to-end delay

The automaton is synchronized with the system output over the global channel *out* and can keep track of the amount of time that passes between the generation of an event and its output from the system. Basically, the automaton can generate input events in the same way as the automaton of Figure 3.14 (left upper transition), but it can also arbitrarily choose to measure the end-to-end delay of an event (right upper transition). In particular, the variable n (initially 0) keeps track of the number of events that have been fed into the system and for which no response (a synchronization over the channel*out*) has been received yet. The clock y measures the response time and m (initially -1)

equals the number of responses that must be discarded before the one used for the measurement is seen. At most one measurement can be in progress and $m = -1$ if no measurement is in progress. For more details we refer the reader to [9].

Similar 'measuring' automaton variants are available also for other event streams. To ensure that the worst-case end-to-end delay of an event does not exceed a given value d it is sufficient to verify the following property by model checking:

$$AG \quad (IG.seen \Rightarrow IG.y < d)$$

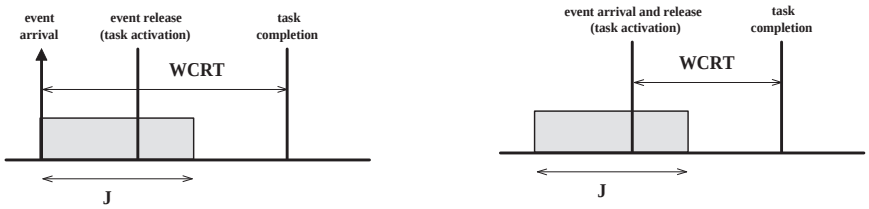
where we assume that 'IG' is the name of the measuring automaton. Again, the *exact* worst-case end-to-end delay can be determined by finding the smallest d that satisfies the property.

The described method for performance analysis based on model checking has an important benefit with respect to the approaches considered previously: it permits to derive not only hard but also exact bounds for performance properties of a distributed system. However, the price to pay is a potential high analysis effort due to the exhaustive model checking performed. In particular, the modeling of a distributed embedded system as a network of timed automata can easily lead to a state space explosion which makes the verification of system properties infeasible.

3.7 Remarks

In this section we would like to point out a relevant difference in the interpretation of periodic task activation with jitter adopted by the various performance analysis methods. In particular, the holistic methods interpret the jitter in the activation of a task as release jitter, while all the other considered methods interpret it as arrival jitter. In the former interpretation the arrival and release of an event are distinguished: the events are assumed to arrive exactly at intervals of one period but their release may be delayed up to the maximum jitter value J. In the latter interpretation the maximum jitter value J defines an interval of admissible arrival times.

Although in both cases the interval of admissible task activation times is the same, this leads to a different analysis of the worst-case response time as depicted in Figure 3.18: the holistic methods (a) consider the release jitter already as part of the delay and refer the WCRT to the ideal periodic arrival time of the event, while the other performance analysis methods (b) refer the



(a) interpretation adopted by the holistic methods

(b) interpretation adopted by the other methods

Figure 3.18: Two different interpretations of activation jitter and WCRT

WCRT to the actual task activation time.

Usually it is not possible to convert the WCRT determined by a holistic method to the second interpretation, as the actual task activation instant leading to the worst-case response is unknown. However, the two interpretations of WCRT differ at most for the maximum jitter value J. Thus, the impact of this interpretation difference on the performance analysis results depends on the relative size of the WCRT in comparison with J: if the WCRT is much larger than J, then the two different interpretations will not lead to significantly different results. However, if the actual worst-case delay from the task activation to its completion is considerably smaller than J, then the holistic methods will provide poor performance analysis results compared to methods that adopt the second interpretation.

Chapter 4

PESIMDES - An extendible performance simulation library

PESIMDES (**Pe**rformance **Sim**ulation of **D**istributed **E**mbedded **S**ystems) was developed as part of this thesis and is an extendible open-source library for performance simulation of distributed embedded systems based on SystemC.

In this chapter we briefly review the motivations for developing PESIMDES, describe its features and explain the most important concepts of its implementation. A user guide to PESIMDES can be found in Appendix B.

4.1 Motivation

Whereas the use of formal approaches for performance analysis is still rare in industry, simulation can be considered the current state of the art in MpSoC performance verification. There are various commercial simulation environments for the simulation of distributed embedded systems. They differ from each other mainly in the level of abstraction of the simulation models. The range extends from cycle-accurate simulators for low-level models to discrete event simulators for system-level models.

While there are many different (mostly proprietary) software tools for low-level simulations, we have not found an adequate simulation tool focused on performance estimation of distributed embedded systems on the system-level, i.e. one which abstracts systems to an aggregation of event generators, processing resources and tasks with BCET/WCET.

SystemC [19], a widespread platform for system-level modeling and simulation, can be used to describe and simulate a distributed embedded system on the requested level of abstraction. However, this requires a substantial set-up effort, as all the necessary components of the system model must be implemented from scratch.

Because on a high level of abstraction all distributed embedded systems are composed of the same basic components, we have decided to collect these components in a common repository in order to reduce the set-up effort required for a SystemC performance simulation. The result is PESIMDES, a library for performance estimations of distributed embedded systems build on top of SystemC. PESIMDES is intended to be a pool of reusable modules which are designed to facilitate the system-level modeling and simulation of large distributed embedded systems in early design stages.

4.2 Performance metrics and modeling scope

PESIMDES provides estimations for the two most important performance metrics of a distributed embedded system: latencies and memory requirements. On the one hand the simulation allows to record the maximum observed end-to-end delay for the processing of event streams. On the other hand the maximum observed activation backlog of every single task can be monitored.

These results can be compared to the timing/memory requirements of the system and possible deadline misses or buffer overflows may be detected. The sequence of events leading to such a requirement violation can easily be traced back as all generated input stimuli are stored into trace-files which can be used to replicate the simulation.

Like any other simulation based approach, PESIMDES can only analyze single input instances out of all possible system inputs. Thus the tool itself cannot provide hard bounds for the worst-case performance of a system. It is up to the designer to provide a set of appropriate simulation stimuli which cover all relevant corner cases.

Table 4.1 summarizes the performance metrics supported by the current PESIMDES version. Regarding the modeling scope, PESIMDES offers various components to model the environment, the computation and communication resources, the tasks and the buffers of a distributed embedded system. To create a new system model, it is sufficient to select the proper components, instantiate them and link them together. Table 4.2 gives an overview of the current modeling power of PESIMDES. The modeling scope can easily be extended by adding new components to the library.

Figure 4.1 gives an example of a simple distributed embedded system that

[1]By non-preemptive TDMA we mean that the execution of a task starts only if it can be concluded within the remaining time of the corresponding TDMA-slot, i.e. if the task can be executed without preemption.

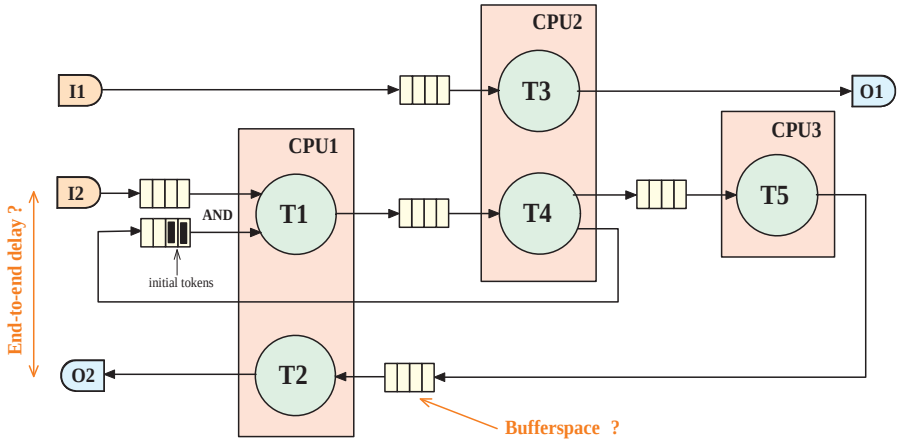
Performance metric	**Supported**
End-to-end delays (latencies)	yes
Memory requirement (buffer dimensions)	yes
Resource utilization	not yet

Table 4.1: Performance metrics supported by the current PESIMDES version

Input Event Models	periodic periodic with jitter periodic with burst sporadic sporadic with jitter sporadic with burst input from tracefile
Resource Models & Scheduling	FP resource (preemptive/non-preemptive) EDF resource (preemptive/non-preemptive) TDMA resource (preemptive/non-preemptive)[1]
Processing Components	task with single activation task with multiple activation (AND) task with multiple activation (OR)

Table 4.2: Modeling scope of the current PESIMDES version

can be simulated with PESIMDES. The corresponding PESIMDES model description is provided in Appendix B.



Input streams	I1: periodic with burst (P=20ms, J=55ms, d=2ms) I2: periodic (P=10)
Resource sharing	CPU1: FP preemptive CPU2: EDF preemptive
Task WCETs	T1: 2ms, T2: 1ms, T3: 7ms, T4: 3ms, T5: 8ms
Scheduling param.	priority T1: high, priority T2: low, rel. deadline T3: 18ms, rel. deadline T4: 35ms

Figure 4.1: Example system

4.3 Implementation concepts

This section describes the key concepts behind the implementation of PESIMDES. In particular we will briefly address the realization of event propagation/buffering, input stream generation and resource sharing. The source code of PESIMDES and more detailed documentation is available online.[2]

All basic components of the PESIMDES library (event sources, event sinks, tasks and resources) are implemented as SystemC modules. These modules are linked among each other with channels which are necessary to propagate the event streams in the system.

[2] *http://www.mpa.ethz.ch*

4.3.1 Event tokens and task activation buffers

The processing of an event stream in a distributed embedded system can be represented as propagation of tokens among modules of the system. The tasks in the system are triggered by incoming tokens (i.e. events) which are processed for a certain amount of time and then forwarded to the next task in the task graph. For performance analysis only the timing aspects of the system are relevant and the actual functionality of the task does not matter. Thus, in PESIMDES the processing of an event by a task is implemented by simply delaying the corresponding token for the execution time of the task.

To permit performance estimation for event streams, tokens need to carry some information with them while being propagated through the system. In particular the data structure representing a token contains:

- a list of its generators with corresponding generation timestamps
- the timestamp of the last task activation request
- a list of tasks by which the token has been processed

The generation timestamp permits to keep track of the end-to-end delay experienced by an event. This delay is simply calculated as the difference between the arrival time at the event sink and the generation time at the event source.

A token can have more than one generator because it may actually be composed of several tokens. This occurs in the context of an AND-activated task: several events (one on each input) are necessary to trigger the task. The token produced at the output of the task must keep the generation information of every input token and thus a list of generators is necessary.

The timestamp of the last task activation is necessary for the implementation of the EDF scheduling protocol. EDF scheduled tasks must meet a deadline relative to their activation request. If the timestamp of the activation request is stored, the observance of the corresponding deadline can be easily verified.

The recording of all the processing tasks of a token is necessary for the detection of cycles in the path from the event source to the event sink. A token finds out that it is following a cyclic path, if it reaches a task by which it has already been processed.

Every task has an activation buffer where incoming activation requests are queued up. This buffer is necessary because a new activation request may arrive before the previous request has been processed. It is implemented as a

FIFO-queue and thus preserves the order of activation requests. In particular, for the implementation of the activation buffers in PESIMDES we use a slightly extended version of the SystemC primitive channel `SC_FIFO`: our channel behaves like `SC_FIFO` but also keeps track of the maximum observed queue length during the simulation of the system. This permits to calculate the maximum backlog for every task and thus the memory requirement of the system.

4.3.2 Input stream generators

This subsection briefly describes the realization of the different input event generators for the periodic event models listed in Table 4.2. The implementations of the sporadic event generators are not reported here for the sake of conciseness, but can be obtained from the corresponding periodic variants with minor changes.

Periodic input

Figure 4.2 shows the desired behavior of a periodic event generator with period P: the first event is generated somewhen in the interval [0,P] (initial phase f) and the following events succeed at intervals of exactly P time units.

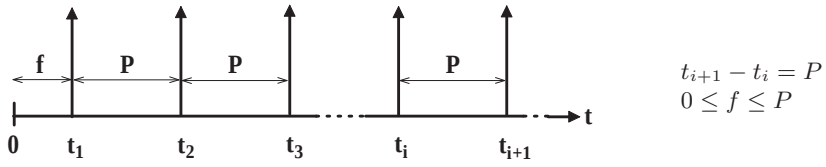


Figure 4.2: Behavior of a periodic event stream generator

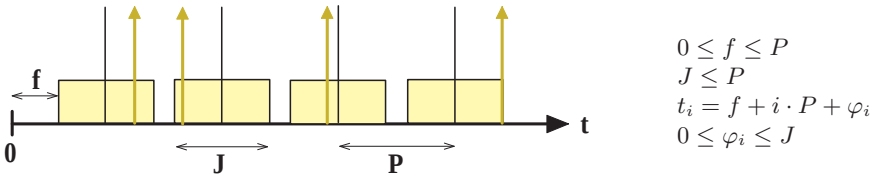
The implementation of the periodic event generator is straightforward - the corresponding pseudo-code is reported in Listing 4.1.

```
periodic_input_generator (period) {
  offset = random number in [0,period];
  wait (offset);
  while (true) {
    generate event;
    wait (period);
  }
}
```

Listing 4.1: Pseudo-code of the periodic event generator

Periodic input with jitter

Figure 4.3 depicts the behavior of a periodic input generator with jitter (see Section 3.4.1 for the description of the corresponding event stream model).



$0 \leq f \leq P$
$J \leq P$
$t_i = f + i \cdot P + \varphi_i$
$0 \leq \varphi_i \leq J$

Figure 4.3: Behavior of a periodic event stream generator with jitter. In each jitter interval (shaded rectangle) exactly one event is generated.

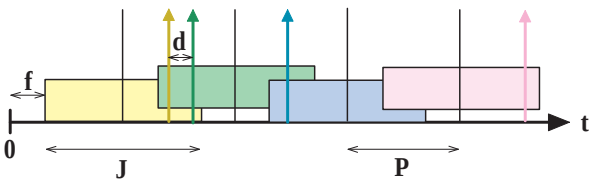
Listing 4.2 shows a simple implementation of the described behavior, where in every period the event generation time is chosen at random among the admissible values.

```
periodic_input_generator_with_jitter (period, jitter) {
  offset = random number in [0,period];
  wait (offset);
  while (true) {
    current_jitter = random number in [0,J];
    wait (current_jitter);
    generate event;
    wait (period - current_jitter);
  }
}
```

Listing 4.2: Pseudo-code of the simple periodic event generator with jitter

Periodic input with burst

The behavior of a periodic input generator with burst is shown in Figure 4.4. The description of the corresponding event stream model can be found in Section 3.4.1.



$0 \leq f \leq P$
$J > P$
$t_i = f + i \cdot P + \varphi_i$
$0 \leq \varphi_i \leq J$
$t_{i+1} - t_i \geq d$

Figure 4.4: Behavior of a periodic event stream generator with burst

The concept behind the implementation of the event stream generator with

burst is represented in Figure 4.5. The arrival time of an event belonging to a preceding period can restrict the admissible arrival interval for the next event. In particular, if an event 'disturbs' the next period then the generation probability for the next event is no longer distributed over the whole jitter interval, but only over the remaining time (denoted as RT in the figure). We call this implementation URT, which stands for 'Uniform over Remaining Time'.

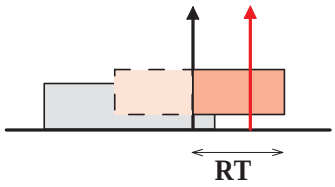


Figure 4.5: Restriction of admissible generation interval

The implementation of the described behavior is shown as pseudo-code in Listing 4.3. It is a simple extension of the periodic input stream generator with $J \leq P$.

```
periodic_input_generator_with_burst_URT (period, jitter,
min_inter_arr) {
  offset = random number in [0,period];
  wait (offset);
  i = 0;
  while (true) {
    if (i-th period not yet started)
       wait until start of i-th period;
    RT = min (now - previous generation, J);
    additional_wait = random number in [0,RT];
    wait (additional_wait);
    generate event;
    wait (min_inter_arr);
    i++;
  }
}
```

Listing 4.3: Pseudo-code of the periodic event generator with burst (URT)

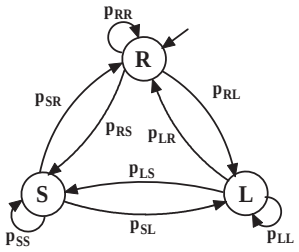
PESIMDES contains also a second implementation variant for the periodic event stream generator with burst which we do not describe here for the sake of conciseness. Basically, it distributes the generation probability of an event over the whole jitter interval and defers events that would be generated 'too early' (i.e. before the arrival of the previous event) to the first admissible generation time. Compared to the first implementation described above, this variant leads to a burstier event generation on average.

Increasing the corner case coverage

In many distributed embedded systems the worst-case performance occurs when some events arrive as soon or as late as possible. However, the implementations of the input generators for periodic event streams with jitter and periodic event streams with burst presented above generate the events at random somewhen in the admissible (remaining) jitter intervals. Thus it is very unlikely that an event is generated exactly as soon or as late as possible and often this leads to performance estimations that are far from the worst case.

For this reason we have included in PESIMDES an implementation of input event streams with jitter/burst that allows users to easily configure the frequency of earliest/latest possible event generations.

In particular we have implemented the simple stochastic finite state machine depicted in Figure 4.6. This input generator can be in one of three states: it either generates the events as soon as possible, as late as possible or at random somewhen in the admissible interval. After each generated event the generator may change state according to state change probabilities set by the user of the module. We will show in chapter 6 that with proper values for the state change probabilities this implementation can lead to much tighter worst-case performance estimations for many distributed embedded systems.



R: generate events at random in admissible interval
S: generate events as soon as possible
L: generate events as late as possible
p_{xy} = probability of state change from x to y

Figure 4.6: The stochastic FSM at the bottom of the event generator

4.3.3 Resource sharing

This subsection describes briefly how the sharing of computation and communication resources among several tasks is simulated in PESIMDES. In particular the key concepts behind the implementation of the scheduling algorithms listed in Table 4.2 are addressed.

All the implementations are parameterized by the number of tasks which share a resource. Thus, in PESIMDES an arbitrary number of tasks can be assigned to each shared resource.

For conciseness we describe only the preemptive variants of the different scheduling algorithms. The non-preemptive algorithms are obtained directly from the corresponding preemptive ones with minor changes.

Static Fixed Priority scheduling (FP)

To simulate a shared resource implementing FP scheduling for n tasks, n+1 parallel running threads are generated: one thread for every task plus an additional scheduler thread. These threads interact via SystemC events. Listing 4.4 summarizes the behavior of the task threads and the scheduler thread as pseudo-code. Upon receiving of an activation request a task interrupts the currently running task and notifies the scheduler. The scheduler assigns the resource to the task with the highest priority among all the tasks that require it. When a task completes its execution, it again produces an interrupt to notify the scheduler that the resource can be assigned to some other task.

```
scheduler () {
   while (true) {
      wait for interrupt;
      assign resource to the busy task with the highest priority;
   }
}


task () {
   while (true) {
      wait for activation request;
      notify interrupt;
      while (execution is not terminated) {
         wait for resource;
         process request until execution is terminated or interrupted;
         if (execution is terminated)
            notify interrupt;
      }
   }
}
```

Listing 4.4: Pseudo-code of the FP implementation

Earliest Deadline First scheduling (EDF)

The realization of the EDF scheduling algorithm is similar to the FP implementation and is reported as pseudo-code in Listing 4.5. Again, the tasks interrupt the current execution and notify the scheduler upon receiving an activation request. However, in this case the scheduler must be aware of the task's deadlines

and assigns the resource to the most urgent task, i.e. the task with the closest deadline.

```
scheduler () {
   while (true) {
      wait for interrupt;
      assign resource to the most urgent task;
   }
}


task () {
   while (true) {
      wait for activation request;
      notify interrupt;
      while (execution is not terminated) {
         wait for resource;
         process request until execution is terminated or interrupted;
         if (execution is terminated)
            notify interrupt;
      }
   }
}
```

Listing 4.5: Pseudo-code of the EDF implementation

Time Division Multiple Access scheduling (TDMA)

With the TDMA scheduling policy the tasks dispose of the resource in time slots that are are periodically repeated. These slots are assigned to the tasks for exclusive resource utilization. For simplicity we assume that at most one task is assigned to every TDMA-slot.

Listing 4.6 describes the behavior of the task threads and the scheduler thread as pseudo-code. The scheduler does simply clock the single TDMA-slots. At every slot change it interrupts the current execution and assigns a new value to the turn variable. The tasks wait for their turn to use the resource and occupy it until the execution is terminated or interrupted by a slot change. Unused slot parts are not left to other tasks and cause the resource to be idle.

```
scheduler () {
   while (true) {
      for (i = 0; i < slot_no; i++) {
         turn = i;
         notify interrupt;
         wait (length of slot i);
      }
   }
}


task () {
   while (true) {
      wait for activation request;
      while (execution is not terminated) {
         if (turn == myturn)
            process request until execution is terminated or interrupted;
         else
           wait for interrupt;
      }
   }
}
```

Listing 4.6: Pseudo-code of the TDMA implementation

4.4 Future extensions

Currently PESIMDES is a library of SystemC components and the user is required to write a SystemC main routine that instantiates and connects appropriate modules from the PESIMDES library in order to simulate the system. In the near future we plan to implement a PESIMDES stand-alone tool that can be used without any knowledge of SystemC. In particular, the tool will take as input the specification of a distributed embedded system according to the XML-format described in Appendix A, enriched with an appropriate PESIMDES tag for the configuration of simulation parameters. Therefore, in order to run a simulation, the user will only be required to pass a file containing the XML description of the system to the PESIMDES executable.

Other future work will deal with the extension of the modeling scope of PESIMDES. In the following we give two examples of extensions that we plan to implement in near future.

Arrival curves

Arrival curves are a much more general model to describe event streams than classical representations such as sporadic or periodic event models (see Section 3.5.1 for details).

We plan to develop a component for the PESIMDES library that generates an input stream according to a given pair of arrival curves (upper and lower arrival curve). In particular we plan to employ the algorithm presented by Künzli *et al.* in [12] to generate event traces from arrival curves.

Hierarchical scheduling

In several distributed embedded systems computing or communication resources are shared among tasks using scheduling strategies in a hierarchical manner. We plan to add the support for hierarchical scheduling in PESIMDES by introducing so called pseudo-tasks. These are tasks that are scheduled on a resource like common tasks but act themselves as schedulers towards other tasks by forwarding the assigned resource share.

Chapter 5

Comparison of performance analysis methods

The need for reliable and accurate performance analysis in early design stages has driven research for many years. The increasing relevance of tight performance prediction is reflected in a growing literature about approaches to performance analysis. However, the various approaches are very heterogenous in terms of modeling scope, modeling effort, tool support, accuracy and scalability and there is a lack of literature on their classification and comparison.

In this chapter we first address several criteria that we regard as relevant for the classification and comparison of performance analysis approaches (Section 5.1). We then classify, compare and evaluate the performance analysis methods described in chapter 3 with respect to several of these criteria (Sections 5.2 and 5.3).

5.1 Comparison criteria

In the following we discuss possible comparison and classification criteria for performance analysis methods. Most of them are not directly quantifiable, but play an important role in the distinction of performance analysis approaches.

Performance metrics

A first classification criterion for performance analysis methods is given by the set of analyzable performance metrics (see section 2.2). In particular, a performance analysis approach may support the analysis of system characteristics like timing aspects, memory requirement, resource utilization or power consumption. The analysis of timing aspects includes the determination of best-case and worst-case latencies and end-to-end delays. The analysis of the memory requirement is often related to the determination of worst-case buffer fill levels.

Modeling scope

A fundamental comparison criterion for performance analysis methods is the modeling scope. By the modeling scope of a certain approach we mean the set of distributed embedded systems that can be represented and analyzed using the modeling power of the method. For instance the capability to model several particular system characteristics, such as hierarchical scheduling, blocking times, multiple task activation etc., differentiates the modeling scopes of the various performance analysis methods.

Correctness and accuracy

A worst-case analysis is said to be correct if the result is a hard upper bound for the real worst-case performance of the considered system. In other words, there are no reachable system states which would allow the calculated bound to be violated.

The accuracy of a performance analysis is usually not quantifiable because the exact worst-case performance of the considered system is unknown. However, a performance analysis method is more accurate than another for a certain system if it provides a tighter upper (lower) bound for the worst-case (best-case) performance.

Modularity

Performance analysis methods can be classified into modular and holistic approaches (see Chapter 3). The modular approaches analyze the performance of single components of the system and propagate the results in order to determine the performance of the entire system. In contrast, the holistic approaches consider the system as a whole. Modular performance analysis methods are typically less complex and easier to reuse than holistic ones.

Modeling effort and tool support

An important criterion for the comparison of performance analysis methods is the effort that it costs the designer to create system models. The modeling effort can be largely alleviated by appropriate software tools.

Analysis effort

This criterion considers the computational effort that is necessary to obtain performance analysis results. For instance one could compare the running times of the tools that implement the different performance analysis approaches.

Scalability

A relevant comparison criterion for performance analysis approaches is scalability. In particular, this point is pertinent to several of the previous criteria: the modeling and analysis efforts as well as the accuracy of the results may be greatly influenced by the dimension of the analyzed system.

End-user complexity and learning curve

Other aspects that can be considered for the comparison of performance analysis approaches are the complexity experienced by the end-user that applies a certain method or tool, as well as the progression of its learning curve. In particular, these points are largely influenced by the amount of background knowledge that a user must acquire about a certain performance analysis approach in order to be able to apply it.

5.2 Comparison of modeling scope and performance metrics

In this section we compare the performance analysis methods described in Chapter 3 with respect to their modeling scope and the analyzable performance metrics.

The multitude of heterogeneities among the performance analysis approaches and the different levels of abstraction in the modeling of particular system attributes make this task very complex. We therefore restrict the comparison of the various modeling scopes to a number of key attributes and show the result in the form of an overview in Table 5.2.

In particular, we base the comparison on the modeling capabilities of concrete implementations of the various performance analysis approaches. Moreover, we would like to point out that a 'no' in a cell of the table does not mean that the modeling of the corresponding system characteristic is conceptually impossible for the corresponding performance analysis approach. Rather, it denotes that no significant research has so far been conducted to integrate this particular aspect.

The performance metrics which are analyzable by the different approaches are listed in Table 5.1.

	Real Time Calculus	SymTA/S	Holistic scheduling (MAST-tool)	Timed automata based analysis
End-to-end delays	yes	yes	yes	yes
Buffer spaces	yes	yes	no	yes
Resource utilization	yes	yes	yes	no

Table 5.1: Analyzable performance metrics

5.3 Comparison of usability

In this section we compare the performance analysis approaches described in Chapter 3 with respect to their usability. In particular we comment briefly on the modeling effort, tool support, end-user complexity and learning curve for each of the methods.

Real Time Calculus

The modular performance analysis approach based on real time calculus has been implemented in form of a MATLAB toolbox.[6] Basically, the toolbox offers performance modules that the user can combine to produce performance networks (see Section 3.5). These performance modules implement the equations that describe the processing of event and resource streams through tasks, i.e. they abstract the user from the actual Real Time Calculus.

Nevertheless, the user must be familiar with the concepts of arrival and service curves. Moreover, he must learn how to connect the performance modules properly in order to model given scheduling strategies for shared resources. However, once this knowledge is acquired, it is quite straight forward to apply this performance analysis method.

[1] only schedulability test
[2] only for mono-processor systems
[3] see case study in section 6.4
[4] see case study in section 6.5
[5] see case study in section 6.6
[6] available at *http://www.mpa.ethz.ch*

	Real Time Calculus	**SymTA/S**	**Holistic scheduling (MAST-tool)**	**Timed automata based analysis**
Input event model	any (arrival curves)	periodic periodic+jitter periodic+burst sporadic sporadic+jitter sporadic+burst	periodic sporadic singular bursty unbounded	periodic periodic+jitter [20] periodic+burst sporadic sporadic+jitter sporadic+burst
Resource model	any (service curves)	Full service with speed factor	Full service with speed factor	Full service
Scheduling	FP TDMA GPS RM Round robin EDF[1]	FP TDMA RM EDF Round robin Ercosek CAN	FP EDF[2]	FP TDMA [20]
Hierarchical scheduling	yes	no	yes	no
Shared resources (mutually exclusive)	no, but can consider blocking time	no, but can consider blocking time	yes	no
Workload model	any (workload curves)	BCET/WCET interval (variable according to context)	BCET/WCET interval	BCET/WCET interval
Task activation	single activation mult. activation (AND/OR)	single activation mult. activation (AND/OR)	single activation mult. activation (AND/OR)	single activation mult. activation (AND/OR)[3]
Intra-event contexts	yes	yes	no	yes[4]
Workload correlations	yes	no	no	yes[5]

Table 5.2: Comparison of modeling scope

To analyze a distributed embedded system with the RTC Toolbox, the user is required to write a MATLAB program that invokes appropriate commands for the creation and analysis of performance networks. For large systems this may result in a considerable modeling effort. However, a prototype of a graphical user interface for the creation of performance networks has recently been developed. The modeling effort can be drastically reduced by using such a tool.

SymTA/S

Among the evaluated performance analysis methods, the Compositional Scheduling Analysis based on Standard Event Models is by far the easiest approach for the end user to apply. This because it is fully implemented in SymTA/S, a powerful and user-friendly software tool.[7]. Figure 5.1 shows a screenshot of the tool in action.

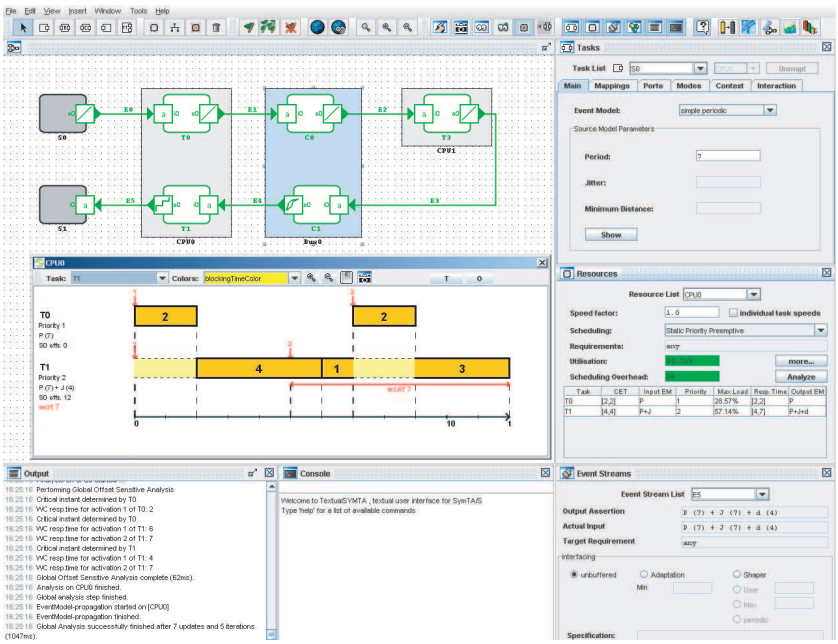


Figure 5.1: SymTA/S screenshot

The intuitive graphical user interface reduces the modeling effort to a minimum and permits the user to model large distributed embedded systems. The tool abstracts completely from the underlying analysis theory and only basic knowledge of embedded systems is necessary for its use.

[7]developed and distributed by Symtavision GmbH *(http://www.symtavision.com)*

Holistic scheduling

As described in Section 3.3 several different holistic performance analysis approaches can be found in the literature. However, there is no tool support for most of them . The consequence is poor scalability in terms of modeling effort. Moreover, the complexity of the analysis theories based on holistic scheduling entails a very steep learning curve for non-specialists.

There is, however, one software tool that implements holistic performance analysis, the so called Modeling and Analysis Suite for Real-Time Applications (MAST)[8][14]. Rather than the implementation of a single performance analysis method, MAST is an open source set of tools for timing analysis of real-time applications.

In order to analyze a distributed embedded system with MAST it is necessary to feed the tool with a model specified in a predefined textual format. Currently, there is no automatic support for the modeling process. Thus, the user faces a considerable modeling effort for large systems. Moreover, the advantage that MAST can consider several details of a system in order to analyze it (for instance resource overheads, mutual exclusions, synchronizations) is paid for by a rather complex input format.

Timed automata based analysis

The performance analysis approach described in Section 3.6 uses the Uppaal tool as support for the design and verification of timed automata networks. However, the construction of the timed automata that represent a distributed embedded system is still done manually, which requires a huge modeling effort. The use of predefined modeling templates may help[9], but the major issue is scalability: with growing system dimensions the modeling effort may increase dramatically. For instance, the timed automaton that is necessary to model a preemptive FP resource with 3 tasks is already considerably larger than the corresponding automaton for 2 tasks, as shown in Figures 5.2 and 5.3.

Nevertheless, as for other performance analysis methods, the modeling process can potentially be automated.

In principle, basic knowledge of timed automata is sufficient to apply this performance analysis method. However, exhaustive model checking may easily lead to a state space explosion and in such cases advanced knowledge about Uppaal and state space reduction may be helpful.

[8] available at *http://mast.unican.es/*
[9] see [9] and [20]

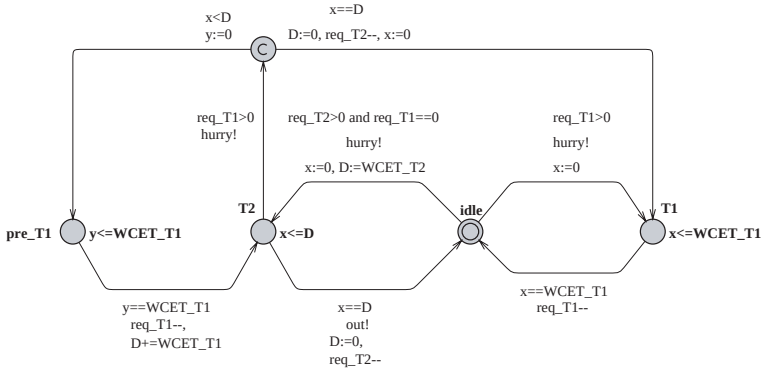


Figure 5.2: Timed automata model for a preemptive FP resource with 2 tasks

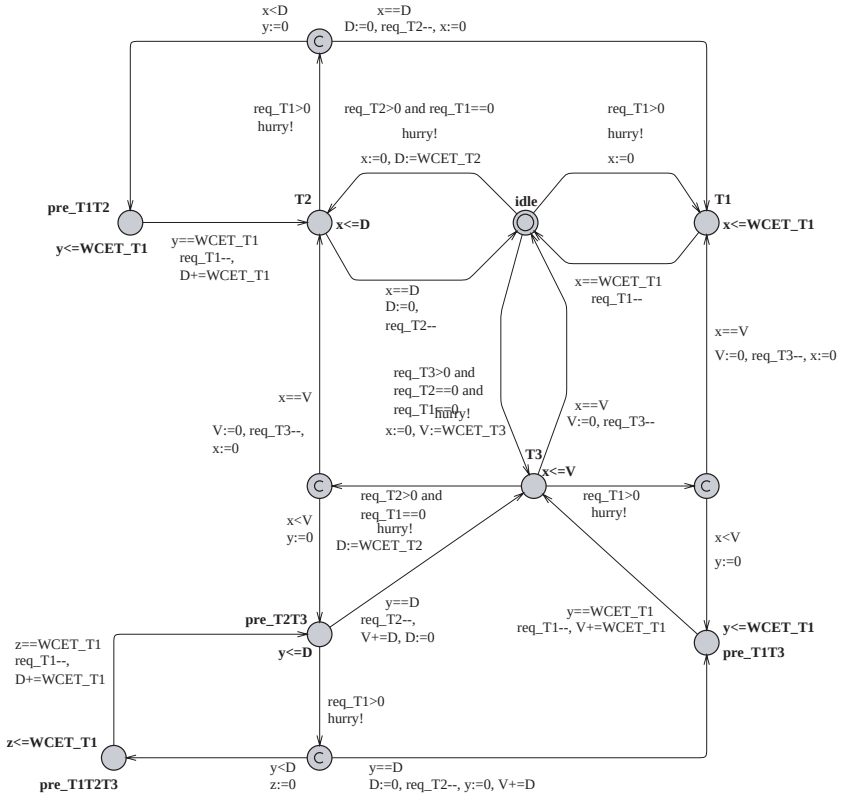


Figure 5.3: Timed automata model for a preemptive FP resource with 3 tasks

Chapter 6

Case studies - Comparison in numbers

In this chapter we present a number of performance analysis case studies. In particular, we apply the performance analysis methods considered in Chapter 3 to some simple distributed embedded systems and compare the results obtained.

The comparison is mainly focused on the accuracy of the different performance analyses, but modeling effort and running times of the tools are also considered in some cases.

Most of the analyzed systems derive from a pool of performance analysis problems defined at the ARTIST2[1] Workshop on Distributed Embedded Systems 2005.[2] At this workshop the originators of several performance analysis approaches met to define a set of benchmark problems for performance analysis of distributed embedded systems[3]. The authors of most of the approaches considered in this thesis were present at this workshop, in particular there were representatives for:

- Real Time Calculus (Swiss Federal Institute of Technology ETH Zurich, Switzerland)
- SymTA/S (TU Braunschweig & Symtavision GmbH, Germany)
- Holistic scheduling (Linkoping University, Sweden)
- Timed automata based analysis (Radboud University Nijmegen & Chess Information Technology B.V., The Netherlands)
- Uppaal (University of Uppsala, Sweden)

[1] ARTIST2 Network of Excellence on Embedded Systems Design *(http://www.artist-embedded.org/FP6/)*

[2] held at the Lorentz Center in Leiden, The Netherlands. Official website of the workshop: *http://www.lorentzcenter.nl/lc/web/2005/20051121/info.php3?wsid=177*

[3] available at *http://www.tik.ee.ethz.ch/∼leiden05/*

The authors of the MAST tool were not present at the workshop; however the research conducted by the group from Linkoping University is tightly coupled to the MAST approach.

At the workshop two kinds of benchmark problems were defined. A first set of benchmarks forms the so called pool of 'small' performance analysis problems. These are basic distributed systems that are focused on a particular analysis problem. The systems are intentionally kept as small as possible (involving only a few tasks and resources) in order to isolate a single analysis difficulty and expose the analysis behavior of the different methods with respect to that particular system characteristic. A second set of benchmarks forms the so called pool of 'large' performance analysis problems. These systems were defined to examine the scalability of the various performance analysis methods.

In this chapter we pick up some of the 'small' problems defined at the ARTIST2 workshop and analyze them with the performance analysis methods described in Chapter 3. Moreover, we also simulate the systems with PESIMDES, the SystemC simulation library presented in Chapter 4. The intention is to compare the upper and lower bounds for the worst-case performance provided respectively by formal analysis methods and simulation.

We do not consider system specifications out of the pool of large performance analysis problems because most of them are restricted to one single analysis method which impedes a direct comparison with other approaches.

For every considered case study we

- describe briefly the system composition and its particularities
- specify the performance characteristics to determine
- discuss potential modeling difficulties of single performance analysis methods
- compare the performance bounds provided by the different methods
- interpret the obtained results

In order to produce meaningful comparisons we do not restrict the analysis problems to a single system configuration but repeat the performance analysis for changing values of a relevant parameter in the system. For instance we vary the values of input jitters or worst case execution times in order to figure out the influence of these parameters on the analysis results and to individuate trends.

All the models constructed and used with the various tools to analyze the worst-case performance of the systems described in this chapter are included in Appendix C.

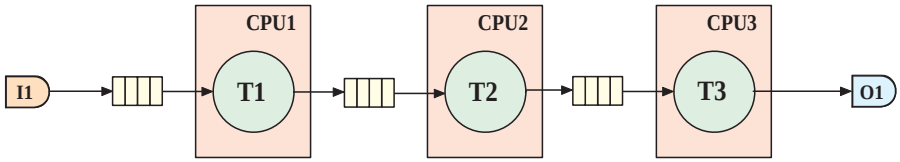
6.1 Case study 1: Pay burst only once

The intention of this case study is to compare the different performance analysis approaches with respect to the sequential processing of a periodic input stream with jitter/burst on independent resources.

This case study is denoted as problem No. 9 in the pool of small performance analysis benchmarks defined at the ARTIST2 Workshop on Distributed Embedded Systems 2005 [18].

Specification

Figure 6.1 depicts the topology of the system. An event stream with burst is processed by a row of three tasks running on independent CPUs. The performance characteristics to determine are the worst-case end-to-end delay from I1 to O1 and the maximum backlog of task T3.



Input stream I1	periodic with burst (P=10ms, J=50ms, d=1ms)
Task WCETs	T1: 1ms, T2: 4ms, T3: 8ms

Figure 6.1: Specification of the case study 'Pay burst only once'

The name 'Pay burst only once' denotes an important property of the system: the WCETs of the tasks are in increasing order and therefore a potential burst at the input I1 is gradually smoothed out while being processed by the system. A consequence of this fact is that no event experiences the worst-case response time on all three processors: an event that has 'paid' a heavy burst on one processor will not experience the worst-case response time on another processor. In other words, the worst-case end-to-end delay of the system is smaller than the sum of the worst-case response times of the three CPUs.

This effect becomes more apparent with increasing jitter values. In order to analyze its impact on the performance bounds calculated by the various

methods we repeat the performance analysis for jitter values of the input event stream from $J = 0\,ms$ (no jitter) to $J = 70\,ms$ (burst).

Modeling

The modeling of the system with the various performance analysis tools is straightforward. The corresponding models can be found in the Appendix Section C.1. There is, however, one point to consider: the MAST tool cannot model bursty event streams with a minimum inter-arrival time. Thus, for the MAST analysis we use the following parameters for I1: $P = 10\,ms, J = 50\,ms, d = 0\,ms$. This will still produce a valid upper bound for the end-to-end delay of the original system, but comports an overapproximation and thus a loss of accuracy.

Regarding the PESIMDES simulation, we model and simulate the system twice using two different library components for the periodic event stream generator with burst. First we use the 'random' burst generator and then we try to improve the corner case coverage by using the burst generator with configurable frequency of earliest/latest possible event generations. The behavior of both components is explained in detail in Section 4.3.2. In particular the state change probabilities used for the stochastic FSM shown in Figure 4.6 are $p_{RS} = p_{SR} = p_{RL} = p_{LR} = p_{SL} = p_{LS} = 0.05$, $p_{RR} = p_{SS} = p_{LL} = 0.9$. We call this model PESIMDES '90-5-5' for short.

Results and Discussion

Figure 6.2 shows the worst-case end-to-end delay provided by the different tools. The values derived with Uppaal are verified through model checking and represent the exact worst-case end-to-end latencies of the system. As can be seen in the chart, the MPA-RTC bound corresponds to the exact worst-case latencies of the system. This shows that MPA-RTC is able to cope with the particular system property described above. For SymTA/S this is not the case: the analysis becomes worse for increasing jitter values. The pessimism of the MAST tool has two causes: first the overapproximation due to the different input event stream with $d = 0$ and second the fact that MAST considers input jitters as delays (this issue is explained in detail in Section 3.7).

Regarding the PESIMDES simulations, the chart shows that the estimations provided by the random burst implementation do not cover the worst case in general and become worse (too optimistic) with increasing jitter values, this despite the simulation of 10000 consequent input events per estimation. In contrast, the PESIMDES 90-5-5 simulation (based on the stochastic FSM)

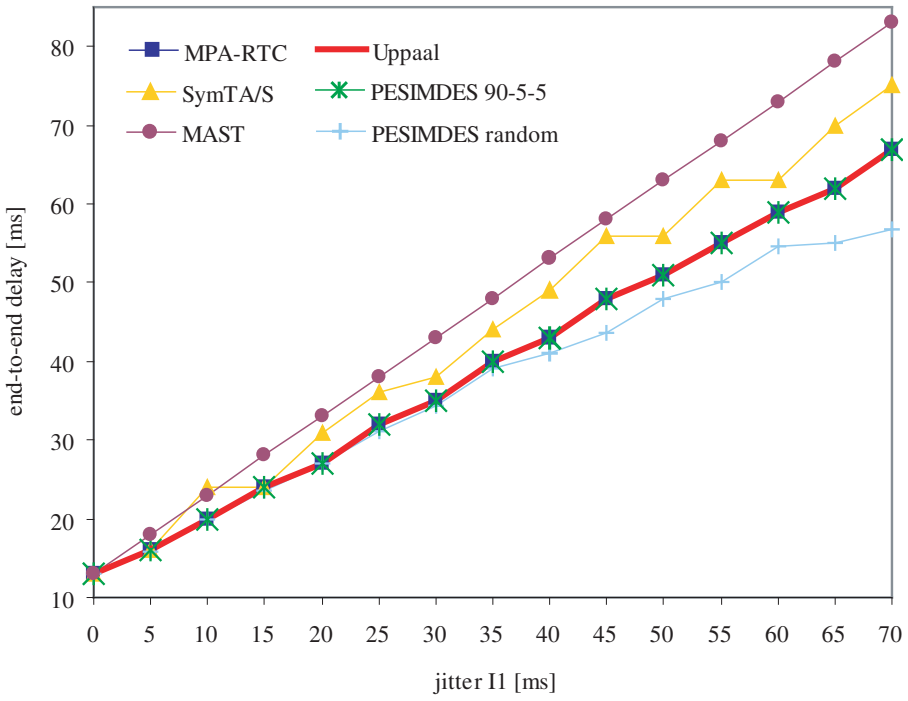


Figure 6.2: Worst-case end-to-end delay

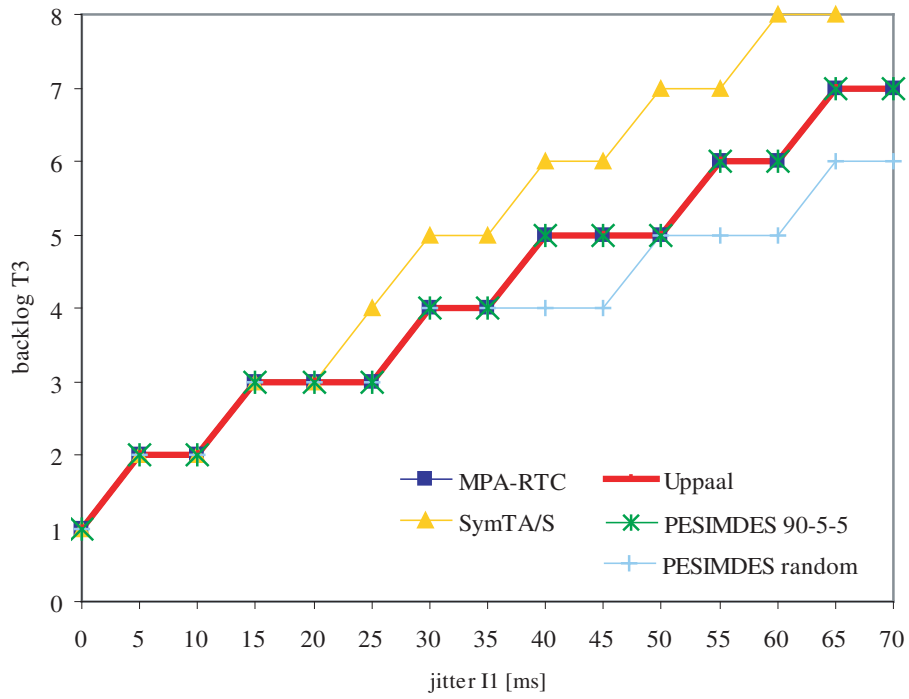


Figure 6.3: Worst-case backlog for T3

covers the corner cases which lead to the worst-case performance of the system and thus provides accurate performance estimations.

The execution times of the various analysis/simulation runs are negligible on a standard PC ($< 1s$) except for the Uppaal model checker: the verification time grows considerably with increasing jitter, from $< 1s$ for $J = 0\,ms$ to $100s$ for $J = 70\,ms$ (for a single step of binary search).

The estimated maximum backlog values are depicted in Figure 6.3. As for the end-to-end delays, Uppaal, MPA-RTC and PESIMDES 90-5-5 provide the exact worst-case backlogs, while the results of SymTA/S and PESIMDES become worse with increasing jitter values. As indicated in the last chapter, the MAST tool does not support the estimation of backlogs.

6.2 Case study 2: Cyclic dependencies

The purpose of this case study is to examine the capability of the different performance analysis methods to deal with feedback loops and cyclic dependencies.

This case study is denoted as problem No. 10 in the pool of small performance analysis benchmarks defined at the ARTIST2 Workshop on Distributed Embedded Systems 2005 [18]. Differently from the original problem specification we omit the minimum interarrival time for the input event stream in order to avoid the penalization of the MAST tool that cannot take into consideration this parameter (see previous case study).

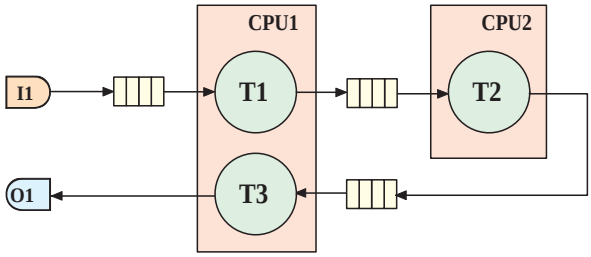
Specification

Figure 6.4 represents the system to be analyzed. An event stream with burst is processed by a sequence of three tasks running on two independent resources. On CPU1, a fixed-priority scheduler is used to schedule T1 and T3. The performance characteristic to be determined is the worst-case end-to-end delay from I1 to O1.

We consider two scenarios for this system. In scenario 1, T1 has higher priority than T3. In this case correlation effects may occur. For instance, depending on the input stream properties it may happen that T1 is never running when T3 has to run and that therefore T3 is never preempted by T1.

In scenario 2, T3 has higher priority than T1. In this case the same correlation effects as in scenario 1 may occur, but additionally, there is a cyclic dependency in the system.

In order to examine the influence of the input jitter on the behavior of the system, we repeat the performance analysis for jitter values from $J = 0\,ms$ to



Input stream I1	periodic with burst (P=10ms, J=50ms, d=0ms)
Resource sharing	CPU1: FP preemptive
Task WCETs	T1: 1ms, T2: 4ms, T3: 4ms
Scheduling param.	1) priority T1: high, priority T2: low 2) priority T1: low, priority T2: high

Figure 6.4: Specification of the case study 'Cyclic dependencies'

$J = 50\,ms$.

Modeling

The two scenarios can be modeled with all the considered performance analysis tools. The corresponding models are enclosed in Appendix Section C.2. However, the modeling of the second scenario requires particular attention. The reason is the cyclic dependency on CPU1: the output behavior of T1 depends on the CPU availability left over after the activity of T3 while at the same time the activity of T3 depends on the output behavior of T1. This dependency can be handled through a fixed-point calculation. However, while in some tools (for instance in SymTA/S) the fixed-point calculation is handled automatically and the user does not need to worry about it, in others (for instance in MPA/RTC) it must be implemented explicitly.

Results and Discussion

The worst-case end-to-end delays determined by the different methods for scenario 1 are shown in Figure 6.5. The values provided by the model checker Uppaal correspond to the exact worst-case end-to-end latencies of the system. The chart shows that the results of the MPA-RTC and SymTA/S analyses slightly exceed the exact worst-case latencies for small jitter values and that the pessimism grows with increasing jitter values.

Regarding the results of the MAST tool, we have described in Section 3.7 that this approach considers input jitters already as delays because the calcula-

tion of the worst-case end-to-end latency is referred to the ideal periodic arrival time of the input event (and not to its actual arrival time as in other methods). This explains why the analysis results of the MAST tool become worse with increasing jitter values.

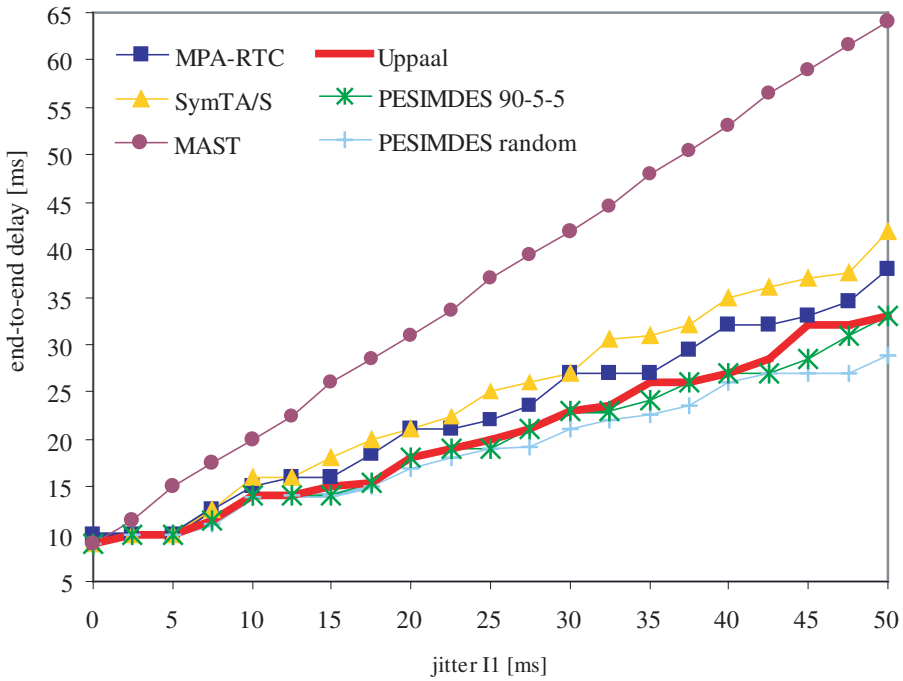


Figure 6.5: Worst-case end-to-end delay for scenario 1

Figure 6.6 shows the worst-case end-to-end delays provided by the different tools for scenario 2. Again, the Uppaal values correspond to the real end-to-end latencies of the system. The chart shows clearly that the cyclic dependency represents a serious difficulty for the analytic methods: the fixed point iteration produces very poor results in the case of MPA-RTC and even worse results in the case of SymTA/S.

In scenario 1 the running times to compute the worst-case delays are not relevant for most methods, except for Uppaal. The exhaustive model checking quickly becomes computationally expensive for large jitter values (up to 50s for one single step of binary search[4]).

In contrast, in scenario 2 SymTA/S has the longest running times (up to 30s) due to the fixed point calculation performed.

[4]On a standard PC (Intel Pentium 4 with 512MB RAM)

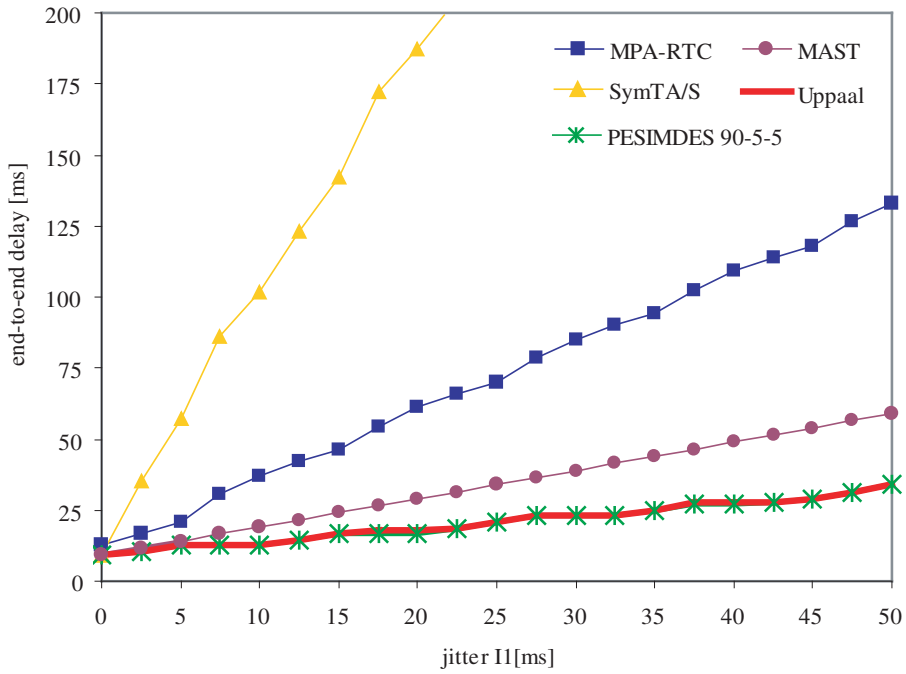


Figure 6.6: Worst-case end-to-end delay for scenario 2

6.3 Case study 3: Variable Feedback

The intention of this case study is to compare the various performance analysis methods with respect to the analysis of buffer requirements and latencies in feedback loops.

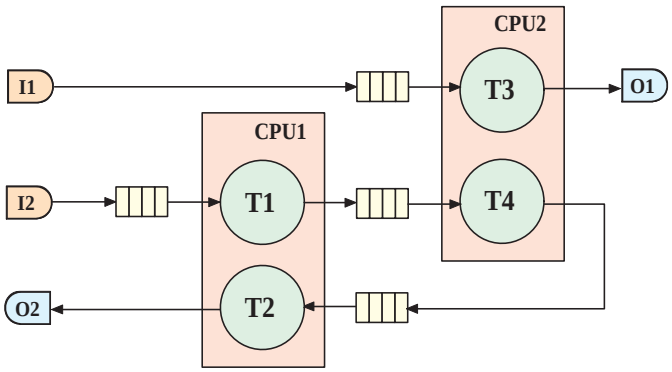
This case study is very similar to the problem No. 5 in the pool of small performance analysis benchmarks defined at the ARTIST2 Workshop on Distributed Embedded Systems 2005 [18]. The following specification differs from the original problem definition in the WCET of T2 (see system definition below). We have modified this parameter in order to shift the system to a design point that is more sensitive to variations of the input event streams.

Specification

Figure 6.7 depicts the topology of the system. An event stream with burst I2 is processed serially by three tasks running on two independent CPUs and forming a feedback loop. Both CPUs implement fixed-priority scheduling and the event stream I2 is disturbed by the upper event stream I1 as task T3 may preempt task T4. The performance characteristics to be determined are the maximum backlog of task T2 and the worst-case end-to-end delay from I2 to O2. The feedback behavior of stream I2 depends strongly on the period of the event stream I1. Altering the period of I1 may cause the correlation effects

between T1 and T2 to vary considerably; thus the name 'Variable Feedback' for the case study.

In order to examine the influence of I1 on the feedback behavior of I2, we repeat the performance analysis for period values of I1 from $P = 4\,ms$ to $P = 30\,ms$.



Input streams	I1: periodic (P=4ms) I2: periodic with burst (P=100ms, J=200ms, d=0ms)
Resource sharing	CPU1: FP preemptive, CPU2: FP preemptive
Task WCETs	T1: 20ms, T2: 15ms, T3: 3ms, T4: 20ms
Scheduling param.	priority T1: high, priority T2: low priority T3: high, priority T4: low

Figure 6.7: Specification of the case study 'Variable Feedback'

Modeling

The modeling of the system with the various performance analysis methods is straightforward and does not present particular difficulties. The corresponding models can be found in Appendix Section C.3

Results and Discussion

Figure 6.8 shows the worst-case backlogs for task T2, determined by the different methods. Uppaal provides the exact backlog values through exhaustive model checking. The curve of the exact backlog values shows an interesting behavior: with increasing period values of stream I1 the influence on stream I2 (in terms of preemptions of task T4) diminishes and thus the events of stream I2 tend to accumulate less in the buffer of T4 and more in the buffer of T2. In other words, the backlog of T2 increases with increasing period values of

I1. However, if the period of I1 exceeds a certain value (22 ms) the correlation effect between T1 and T2 changes and the backlog of T2 diminishes.

The chart in Figure 6.8 demonstrates that the analytic performance evaluation methods are not able to detect this behavior: instead of the exact maximum backlog of 2 processing requests MPA-RTC provides a backlog of 3 and SymTA/S of 4 processing requests. On the other hand the PESIMDES simulation does not always cover the worst-case and underestimates the maximum backlog.

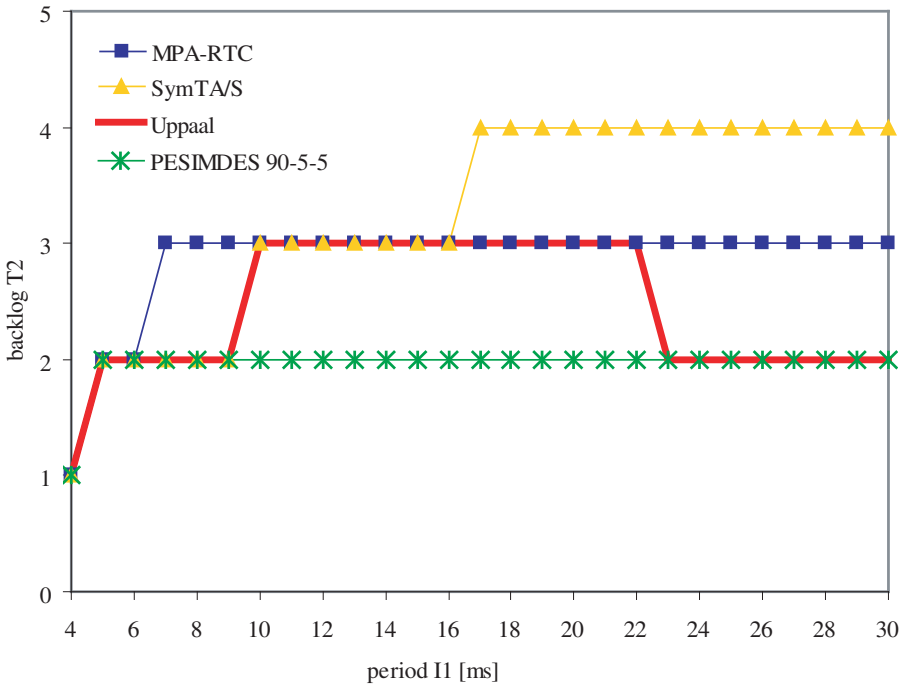


Figure 6.8: Worst-case backlog for T2

Figure 6.9 shows the analysis results for the worst-case end-to-end delay from I2 to O2. The model checker Uppaal permits to verify the exact worst-case delays. However, the large jitter of I2 leads to a huge state space and extremely time-consuming verifications. While for large period values of I1 the verification time is just a few seconds, for small period values the Uppaal verification time becomes unbearable due to the state space explosion. For instance for $P_{I1} = 6\,ms$ the verification time for one single step of the binary search is nearly one hour. For $P_{I1} = 4\,ms$ the verifier runs out of memory after a few hours of verification on a PC with 512MB main memory.

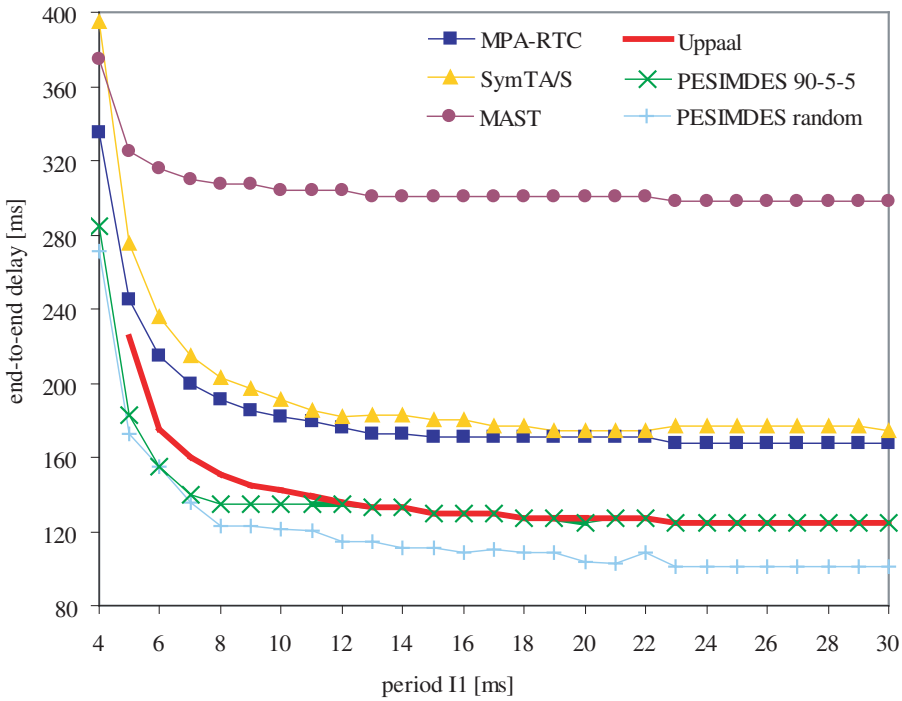


Figure 6.9: Worst-case delay I2-O2

6.4 Case study 4: AND/OR task activation

The purpose of this case study is to compare the different performance analysis approaches with respect to modeling effort and analysis accuracy for tasks with multiple activation. In particular we consider tasks with multiple input ports with AND-activation and OR-activation, respectively.

This case study is denoted as problem No. 3 in the pool of small performance analysis benchmarks defined at the ARTIST2 Workshop on Distributed Embedded Systems 2005 [18].

Specification

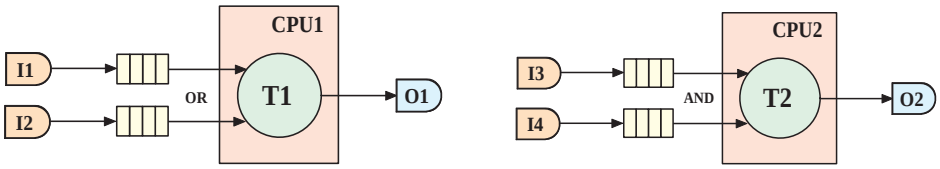
We consider two different scenarios. The first one is depicted in Figure 6.10(a). The system consists of two event streams I1 and I2 and an OR-activated task T1. OR-activation means that each event from any of the two inputs I1 or I2 triggers T1. The two activation queues are served by T1 in FCFS-order[1] and can actually be replaced by one single buffer in which the events from both input stream generators are queued up. The performance characteristics

[1] First-Come-First-Served

to be determined are the worst-case latencies I1-O1 and I2-O1 as well as the maximum backlog of T1.

The second scenario is represented in Figure 6.10(b). The system consists of two event streams I3 and I4 and an AND-activated task T2. AND-activation means that one event from both inputs must be available in order to activate T1. The performance characteristics to be determined are the worst-case latencies I3-O2 and I4-O2 as well as the worst-case total buffer size.

In order to examine the influence of the WCET of T1 and T2 on the latencies and buffer sizes, we repeat the performance analysis for WCET values of T1 and T2 from $5\,ms$ to $60\,ms$.



(a) OR-activation (b) AND-activation

Input streams	I1: periodic with jitter (P=100ms, J=20ms) I2: periodic with jitter (P=150ms, J=60ms) I3: periodic with jitter (P=100ms, J=10ms) I4: periodic with burst (P=100ms, J=190ms, d=20ms)
Task WCETs	T1: 40ms, T2: 40ms

Figure 6.10: Specification of the case study 'AND/OR task activation'

Modeling and Analysis

The modeling of the two simple systems turns out to be straightforward only with the SymTA/S tool. With all the other approaches either modeling difficulties are encountered or some particular procedures or implementations are necessary:

MAST

We were not able to analyze the AND-activation with the current version of the MAST tool (v1.3.6). Although MAST provides a so called barrier event handler which allows to model the AND-activation, the tool interrupts the analysis with the following exception: `Feasible_Processing_Load not yet implemented for Multiple-Event systems.`

For the OR-activation the same problem is encountered; however, the latencies can be analyzed with a workaround model: instead of modeling one

single transaction with an OR-activation (using a so called concentrator event handler), two distinct transactions I1-O1 and I2-O1 that contend the task T1 can be used.

UPPAAL

The AND-activation can be easily modeled by performing minor adjustments to the timed automata network described in Section 3.6. The modeling of the OR-activation is, however, more involved. The problem is that in order to analyze the latencies I1-O1 and I2-O1 the activations of T1 caused by the input streams I1 and I2 respectively must be distinguishable. Moreover, the FCFS order must be respected for the activation requests. Hence, the activation buffer can no longer be represented by a counter variable and must be modeled explicitly. Figure 6.11 shows the timed automaton used to model the activation buffer. Basically, the automaton has a location for every possible buffer state.

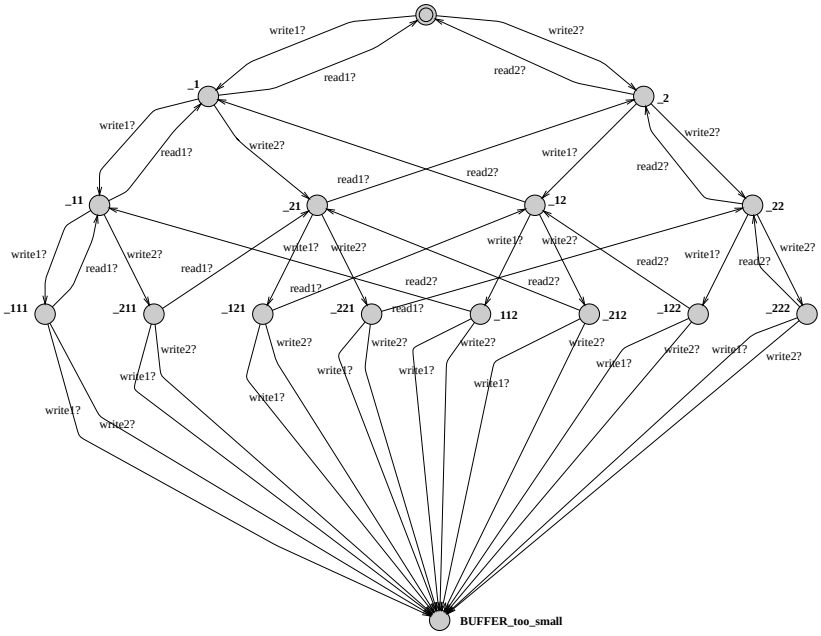


Figure 6.11: TA model for the activation buffer

In the current case study the backlog of T1 does not exceed 3 activation requests and thus the automaton has 'only' 16 locations. However, the number of locations grows exponentially with the maximum backlog: for an OR-activated task with m inputs and a maximum backlog of n activation requests, an automaton with $\frac{m^{n+1}-1}{m-1} + 1$ locations is required. Therefore, in general, this way of modeling the OR-activation involves an impractical modeling and verification effort.

MPA-RTC

The analysis of the system in Figure 6.10(b) with MPA-RTC requires some extra attention. In particular, in MPA-RTC the AND-activation is implemented as shown in Figure 6.12. Input events remain in the buffer B_1 only as long as the buffer B_2 is empty and vice versa. As soon as a 'partner' event arrives on the other input, the two events merge and pass to buffer B_T. Thus, either B_1 or B_2 is always empty.

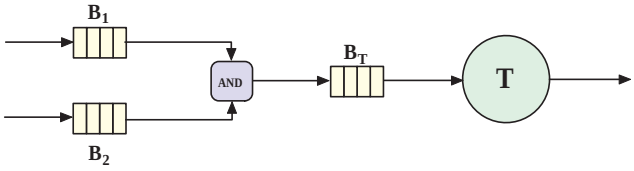


Figure 6.12: AND-activation in MPA-RTC

If the default MPA-RTC functions for delay and backlog analysis are called on the processing component modeling the task T, the corresponding analysis considers only the activation buffer B_T. However, in the current case study buffers B_1 and B_2 also need to be considered.

Therefore, in order to calculate upper bounds for the worst-case end-to-end delays we add to the worst-case response time of T the maximum time that an event may spend in B_1 or B_2 respectively. This time can be easily computed as the maximum horizontal distance between the upper arrival curve of one input and the lower arrival curve of the other input. Upper bounds for the buffer sizes of B_1 and B_2 can be calculated in an analogous way using the maximum vertical distance between the arrival curves. Moreover, in order to draw a fair comparison with the other approaches, the buffer size of B_T is counted twice when estimating the total buffer size of the system. This is necessary to compensate the merging of events before the queue B_T which does not occur in the original case study.

These simple adjustments guarantee upper bounds for the required performance quantities, but may in general provide pessimistic results.

PESIMDES

The PESIMDES simulator provides the maximum observed fill level for every single buffer of the system. However, it cannot trace the progression of the sum of two or more buffer sizes. To estimate the total required buffer size for a task with multiple activation, we simply sum the maximum observed fill levels of all the input buffers. However, this can easily

lead to an overestimation, as the total worst-case buffer size of a task with multiple inputs is in general not the sum of the worst-case sizes of the single input buffers.

All the corresponding modeling solutions are included in Appendix Section C.4.

Results and Discussion

Figure 6.13 and Figure 6.14 respectively show the worst-case delays from I1 to O1 and the worst-case backlogs for T1 determined by the different methods for the first scenario (OR-activation). The values calculated by the Uppaal model checker represent the exact system performance. As suspected, by adding the two worst-case buffer sizes observed by PESIMDES, the actual worst-case backlog of T1 is overestimated in some cases. This shows that in these cases the two input buffers do not reach their worst-case filling contemporarily.

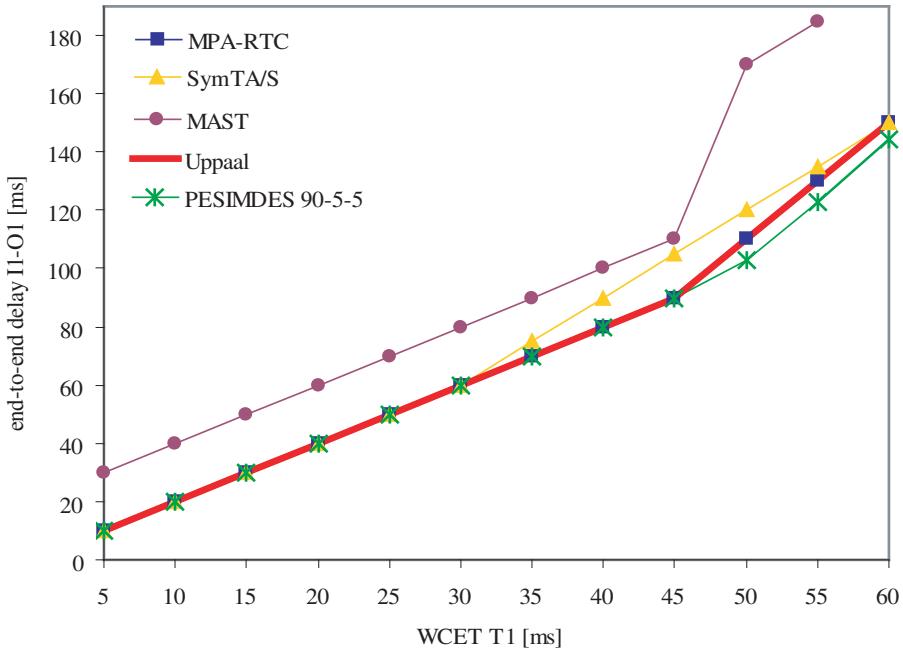


Figure 6.13: Worst-case delay I1-O1 (OR-activation)

Figure 6.16 and Figure 6.17 depict the estimated worst-case end-to-end delays from I3 to O2 and from I4 to O2 respectively. The exact delays determined by Uppaal show that the worst-case latency I4-O2 is much lower than the worst-case latency I3-O2. This is because events generated by I3 may have to wait much longer for a 'partner' event than events generated by I4, which results from the different parameters of the two input streams: while the maximum

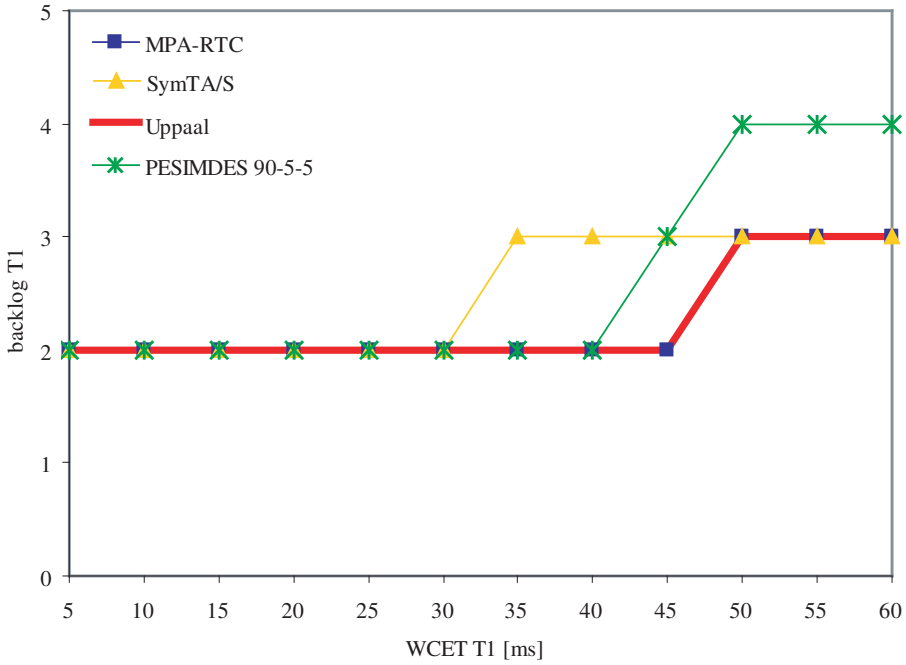


Figure 6.14: Worst-case backlog for T1 (OR-activation)

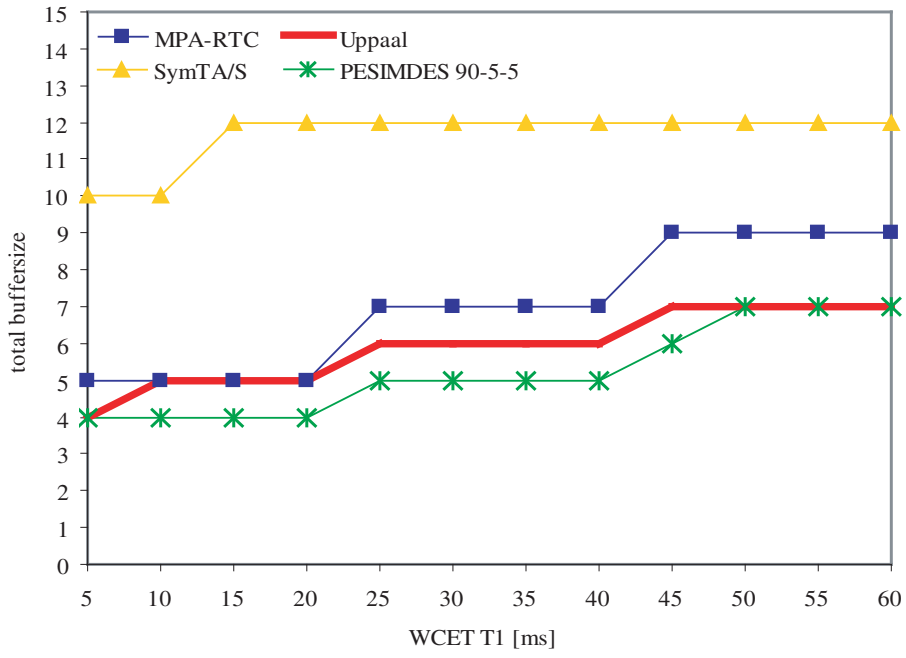


Figure 6.15: Worst-case total buffer size (AND-activation)

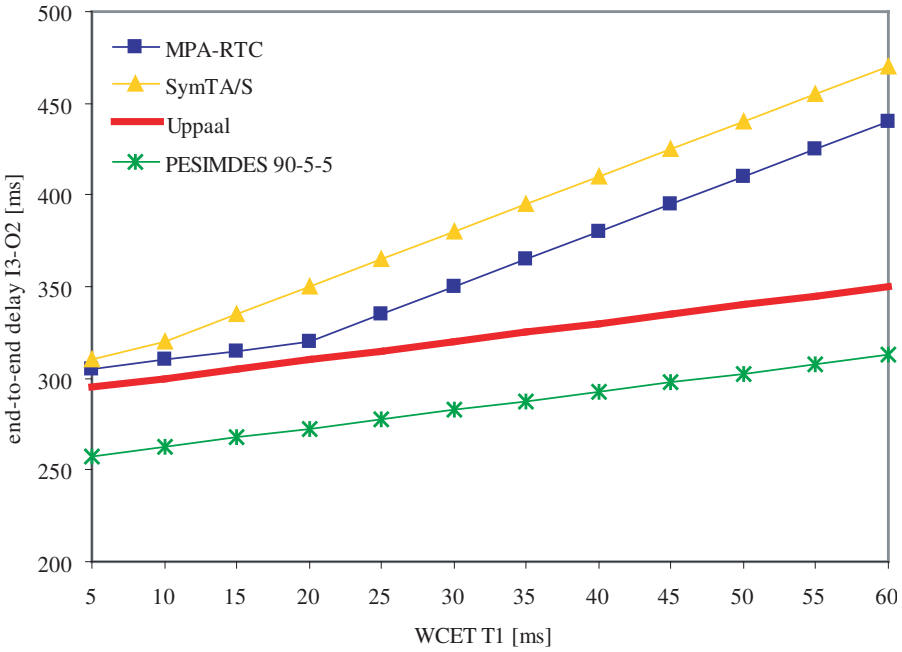


Figure 6.16: Worst-case delay I3-O2 (AND-activation)

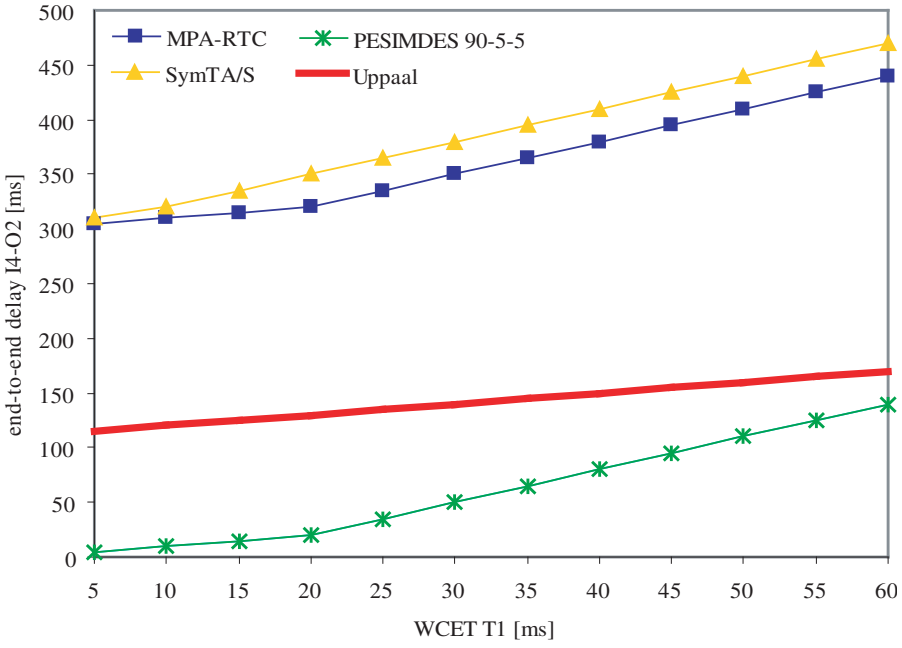


Figure 6.17: Worst-case delay I4-O2 (AND-activation)

distance between two events generated by I3 is 110 ms, two events generated by I4 may have a maximum distance of 290 ms.

However, the charts show that the analytic performance evaluation approaches are not able to detect this difference: both MPA-RTC and SymTA/S respectively provide the same bounds for the two end-to-end delays. The result is a very pessimistic upper bound for the worst-case latency I4-O2.

Figure 6.15 shows the results provided by the different approaches for the total buffer size required to handle the AND-activation.

6.5 Case study 5: Intra-context information

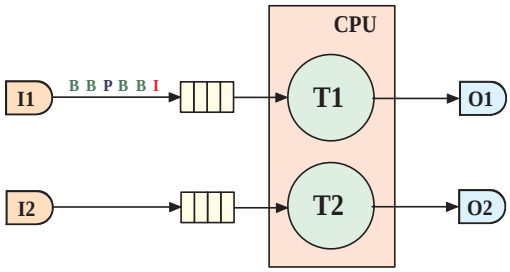
The intention of this case study is to compare the various approaches with respect to the ability to exploit event type correlations within the activation sequence of a task. In particular, tasks can be activated in many real systems by different types of events yielding to different WCETs. Often information about the sequence of activating event types is available from the system context. This so called intra-context information can be exploited to determine tight bounds for the worst-case system performance.

This case study is very similar to the problem No. 4 in the pool of small performance analysis benchmarks defined at the ARTIST2 Workshop on Distributed Embedded Systems 2005 [18]. The following specification differs from the original problem definition in the period of the input event streams. We have modified these parameters in order to allow the analysis with all the methods considered. With the original parameters a numerical comparison would not have been possible as for context-blind analysis methods the original system results not schedulable.

Specification

Figure 6.18 depicts the topology of the system under consideration. Two event streams are processed by two tasks on a CPU that implements fixed-priority scheduling. In particular, T1 decodes an MPEG stream generated by I1. The worst-case execution time of T1 varies according to the decoded frame type I, P or B. We assume that I1 repeatedly generates the following frame pattern: I B B P B B. The performance characteristic to be determined is the worst-case response time of T2.

In order to point out the influence of the varying workload of T1 on the response time of T2, we repeat the performance analysis for several different ratios between the WCETs of T1. We start with equal WCETs for I, P and B



Input streams	I1: periodic (P=200ms), I2: periodic (P=1000ms)
Resource sharing	CPU: FP preemptive
Task WCETs	T1: 80ms (mode I), 40ms (mode P), 20ms (mode B) T2: 200ms
Scheduling param.	priority T1: high, priority T2: low

Figure 6.18: Specification of the case study 'Intra-context information'

frames. In this case the intra-event correlations do not affect the performance of the system. Then, we gradually diminish the WCETs of T1 for P and B frames in proportion to the WCET for I frames. In this way we expect to observe an increasing advantage for analysis methods that exploit the intra-context information.

Modeling

Not all the performance analysis methods considered support the modeling of intra-context information. For instance with the current version of the MAST tool only a context-blind analysis is feasible.

In MPA-RTC the specified system can be easily modeled by representing the cyclic frame pattern with an appropriate arrival curve. Also in Uppaal the cyclic frame pattern and the influence of the event types on the workload can be modeled with some changes to the original timed automata models described in Section 3.6. The respective models are included in the Appendix Section C.5.

While in Uppaal and MPA-RTC customized models are necessary to take intra-context information into account, in SymTA/S there is no additional modeling effort: the user just needs to describe the context characteristics of the typed event stream in appropriate input fields.

The current modeling scope of the PESIMDES stand-alone tool does not yet cover typed events and intra-event correlations. However, as described in Chapter 4, the simulator can be easily extended and the integration of intra-context information is planned as future work.

Results and Discussion

Figure 6.19 shows the various values provided for the worst-case response time of T2. The Uppaal curve represents the exact worst-case response times of T2, verified by exhaustive model checking. As expected, ignoring the intra-context information leads to increasingly pessimistic results with decreasing execution demands for P and B frames.

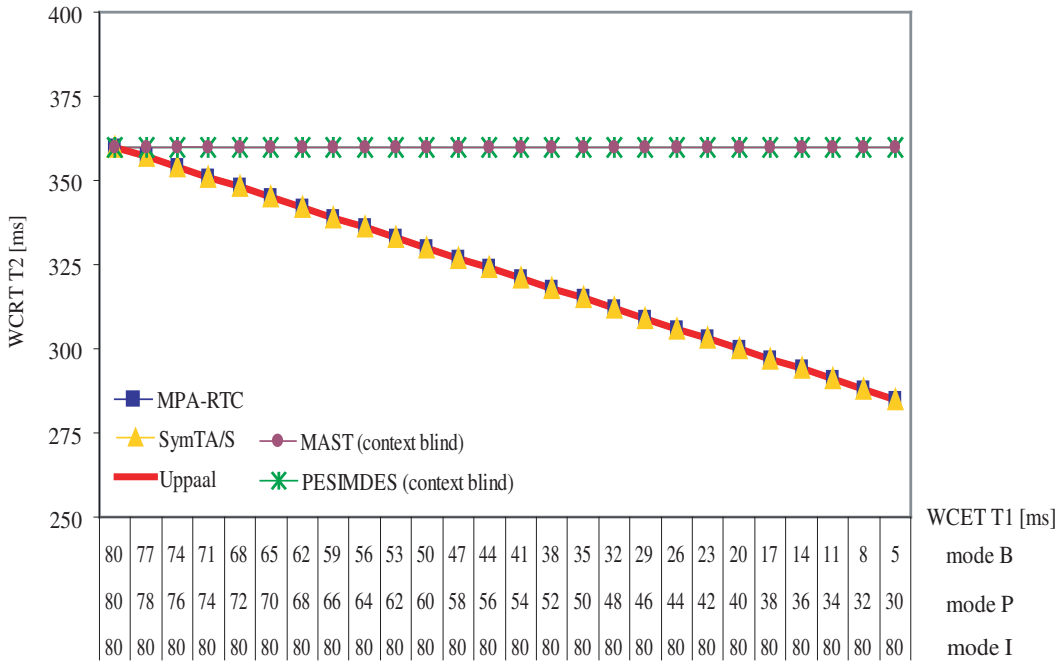


Figure 6.19: Worst-case delay I2-O2

6.6 Case study 6: Workload correlations

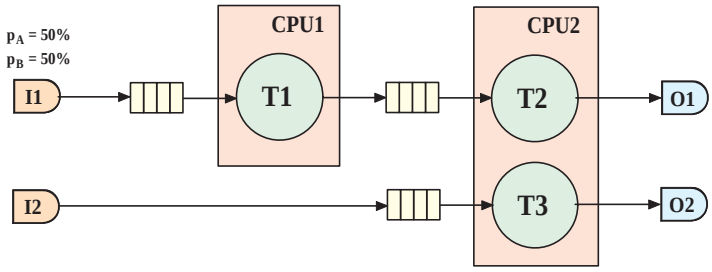
The purpose of this case study is to compare the various performance analysis methods with respect to the results provided for systems with workload correlations. In particular, in many embedded systems different events cause different execution demands on several tasks and often these execution demands are highly correlated. For instance in many data processing systems the execution demand of most tasks depends on the payload of the triggering event: an event with a small payload will cause a small execution demand on most tasks whereas an event with a large payload will impose a heavy workload on most tasks. The consideration of such workload correlations can lead to a much more accurate performance analysis.

The example system considered in this case study was first presented by Wandeler and Thiele in [30], where an abstract model to characterize and capture workload correlations is introduced.

Specification

Figure 6.20 shows the topology of the system considered. The event generator I1 generates a periodic event stream with burst. The generated events can be either of type A or B. Both event types are equally probable and there are no intra-event correlations, i.e. the arrival pattern is random. The events A and B impose different workloads on T1 and and T2. In particular, there is a clear correlation between the workloads of T1 and T2: an event of type A creates a high workload on both tasks, while an event of type B creates a low workload on both tasks.

The system also processes a second event stream generated by I2. The events of this stream are of a single type and all have the same worst-case execution demand for T3. The performance attribute sought is the worst-case latency form I2 to O2, i.e. the worst-case response time of T3.



Input streams	I1: periodic with burst (P=4ms, J=15ms, d=1ms) I2: periodic with jitter (P=6ms, J=1ms)
Resource sharing	CPU2: FP preemptive
Execution demands	T1(A): 20000 cycles , T1(B): 5000 cycles T2(A): 15000 cycles, T2(B): 5000 cycles T3: 5000 cycles
Scheduling param.	priority T2: high, priority T3: low
CPU speeds	CPU1: 6 MHz, CPU2: 6 MHz

Figure 6.20: Specification of the case study 'Workload correlations'

In order to examine the accuracy of the determined delay for several different system configurations, we repeat the performance analysis for increasing processor speeds of CPU2.

Modeling

Not all the considered performance analysis methods can model the workload correlation described in the above specification properly. In some cases the modeling is possible but requires particular adjustments.

SymTA/S and MAST

With these two tools the different execution demands of the event types A and B can only be considered as the worst-case execution demand or the best-case execution demand for a task. However, there is no way to model the fact that events with a low execution demand for T1 also have a low execution demand for T2. Therefore we expect overly pessimistic analysis results for these approaches.

Uppaal

The timed automata networks described in Section 3.6 can be extended to allow the modeling of typed events and variable task workload. However, the same obstacle as in Case study 4 is encountered: the activation buffers of T1 and T2 can no longer be represented anymore as simple counter variables but need to be modeled explicitly, as the events of type A and B must be distinguishable.

Figure 6.21 shows the timed automaton used to model the activation buffer of T1. Again, the automaton has a location for every possible buffer state up to a buffer size of four events, which is sufficient for the particular case study. Nevertheless, the modeling effort is already enormous.

The number of locations grows exponentially with the required buffer size. Thus, this modeling solution is inapplicable in general.

MPA-RTC

The workload correlations can be easily modeled with the MPA-RTC toolbox. The corresponding model is included in Appendix Section C.6. More details about the integration of workload correlations in the modular performance analysis framework can be found in [30].

PESIMDES

The PESIMDES stand-alone tool does not yet support typed events and can therefore not be used to simulate the specified system. However, starting from the PESIMDES library we have implemented an ad-hoc SystemC simulation for the described system with very little effort.

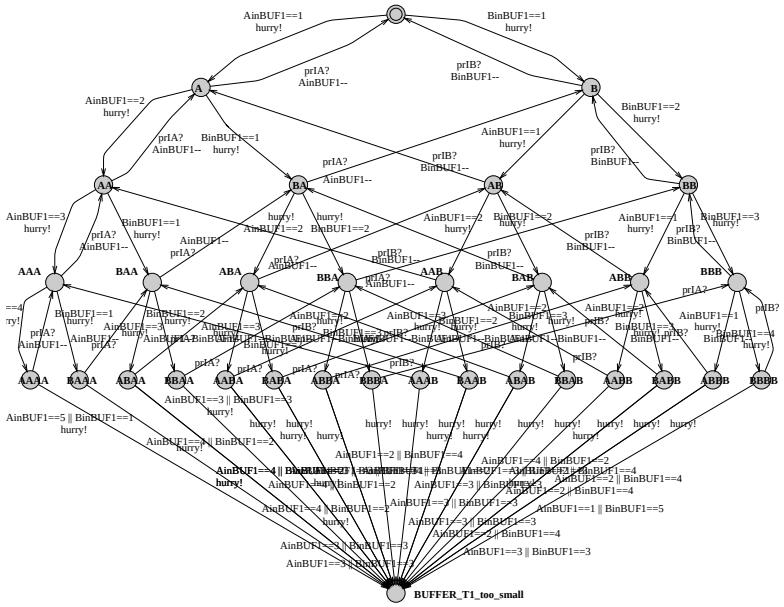


Figure 6.21: TA model for the activation buffer of T1

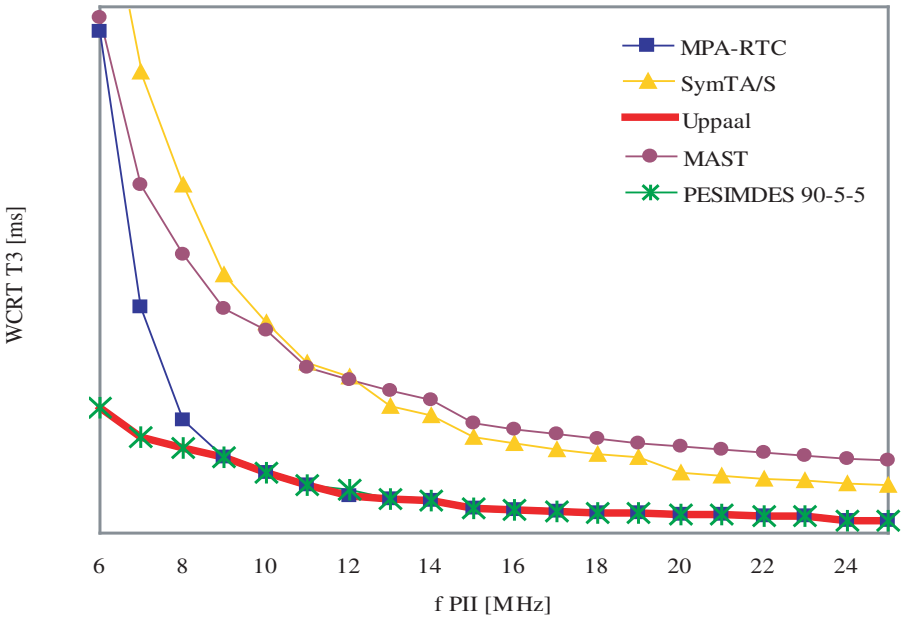


Figure 6.22: Worst-case delay I2-O2

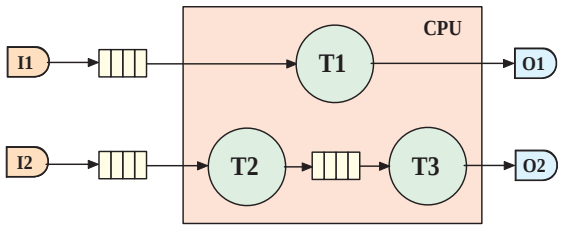
Results and Discussion

Figure 6.22 shows the results provided by the different performance analysis methods. As usual, the Uppaal data series has been determined with exhaustive model checking and represents the exact worst-case response time of T3. The chart shows clearly that by ignoring the workload correlations the worst-case performance of the system is considerably overestimated. In particular, as expected, the SymTA/S and MAST methods provide overly pessimistic bounds for the worst-case performance.

6.7 Case study 7: Data dependencies

The intention of this case study is to illustrate the ability of the various performance analysis approaches to deal with data dependencies that may exist among the tasks of a distributed embedded system. In particular, the activation times of various tasks are not independent in many systems: data dependencies force the tasks to be executed in a certain order and impose temporal offsets between their activation. These dependencies can be exploited to determine tight bounds for the worst-case system performance.

The system analyzed in this case study is an example provided by Yen and Wolf in [32], where an algorithm to determine tight bounds on the execution time of systems with data dependencies is presented.



Input streams	I1: periodic (P=80ms), I2: periodic (P=50ms)
Resource sharing	CPU: FP preemptive
Task WCETs	T1: 15ms, T2: 20ms, T3: 10ms
Scheduling param.	priority T1: high, priority T2: medium, priority T3: low

Figure 6.23: Specification of the case study 'Data dependencies'

Specification

Figure 6.23 depicts the considered system. Two periodic event streams are processed by three tasks on a CPU that implements preemptive fixed priority scheduling. The data dependency is given by the processing sequence T2-T3. The performance characteristic to be determined is the worst-case end-to-end delay from I2 to O2.

We repeat the performance analysis for WCETs of T1 between $15\,ms$ and $30\,ms$.

Modeling

The system can be easily modeled with all the considered performance analysis tools. The corresponding models are included in Appendix Section C.7.

Results and Discussion

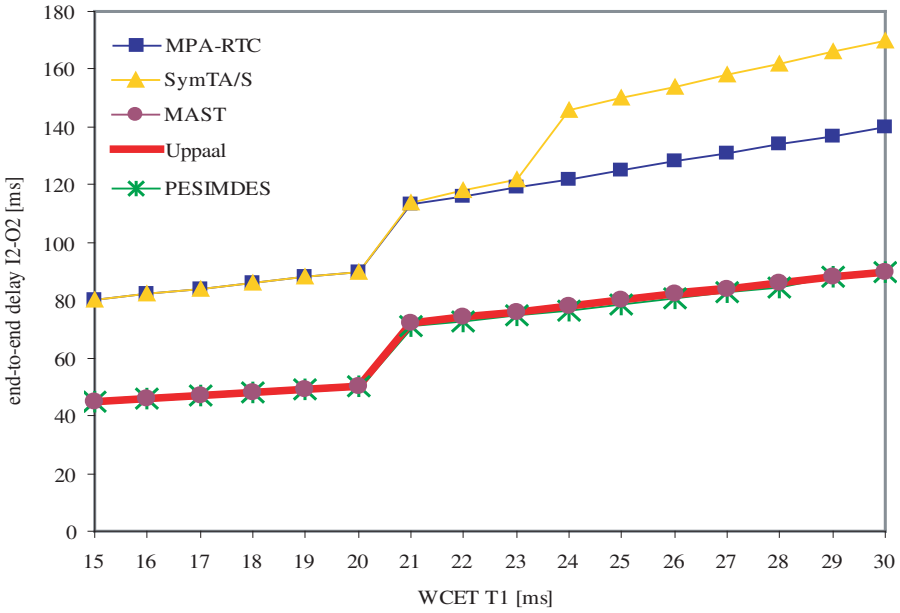


Figure 6.24: Worst-case delay I2-O2

Figure 6.24 represents the worst-case latencies I2-O2 determined by the various methods. The Uppaal curve indicates the exact worst-case latencies verified by model checking. The chart shows that MPA-RTC and SymTA/S provide very pessimistic bounds for the worst-case system performance. The overly pessimistic analysis results from the disregard of the data dependencies. Let us consider, for instance, the system configuration with $WCET_{T1} = 15\,ms$. It is easily traceable that in this configuration

1. T1 can only preempt either T2 or T3, but not both in a single execution
2. T2 cannot preempt T3

Hence, the worst-case latency I2-O2 is $45\,ms$. However, MPA-RTC and SymTA/S ignore the data dependencies between T2 and T3 and consider their activation times independently. Thus, they suppose a worst-case response time of $35\,ms$ for T1 and $45\,ms$ for T2 and estimate the worst-case latency I2-O2 with $80\,ms$, the sum of the two delays.

In contrast, the MAST tool implements an offset based analysis that detects the data dependencies described. The figure shows that the MAST results correspond precisely to the exact worst-case latencies.

6.8 Overview

In this section we summarize the results obtained in terms of accuracy by the different performance analysis methods for the case studies described in this chapter. In particular we give an overview of the degree of pessimism of the various performance analyses in Figures 6.25 and 6.26. For every case study and every method we indicate the interval of the analysis results obtained relative to the real performance characteristic of the system (100%). The intervals are represented as vertical bars: the lower edge of a bar indicates the result of the least pessimistic analysis and the upper edge the result of the most pessimistic analysis performed for the different parameters. For instance, Figure 6.25 shows that for the analysis series of scenario 2 in the case study *Cyclic dependencies* (see Section 6.2) the performance analysis method MPA-RTC produced results in the interval between 144% (best analysis result) and 407% (worst analysis result) of the actual system performance. A dot on the 100%-mark indicates that the corresponding analysis method provided exact performance results for all the parameter values of the corresponding case study.

The two charts also show the results obtained by simulation with the PESIMDES-tool. In this case the bars are normally below the 100%-mark which indicates over-optimistic performance estimations. There are, however, a few exceptions where the bars are above the 100%-mark, i.e. the simulation overestimates the system performance. This happens if the simulation model is not able to capture particularities of the system specification. For instance in the case study *Intra-context information* (see Section 6.5) PESIMDES overestimates the system performance because it cannot model variable execution demands and thus assumes the worst-case execution demand for every task activation.

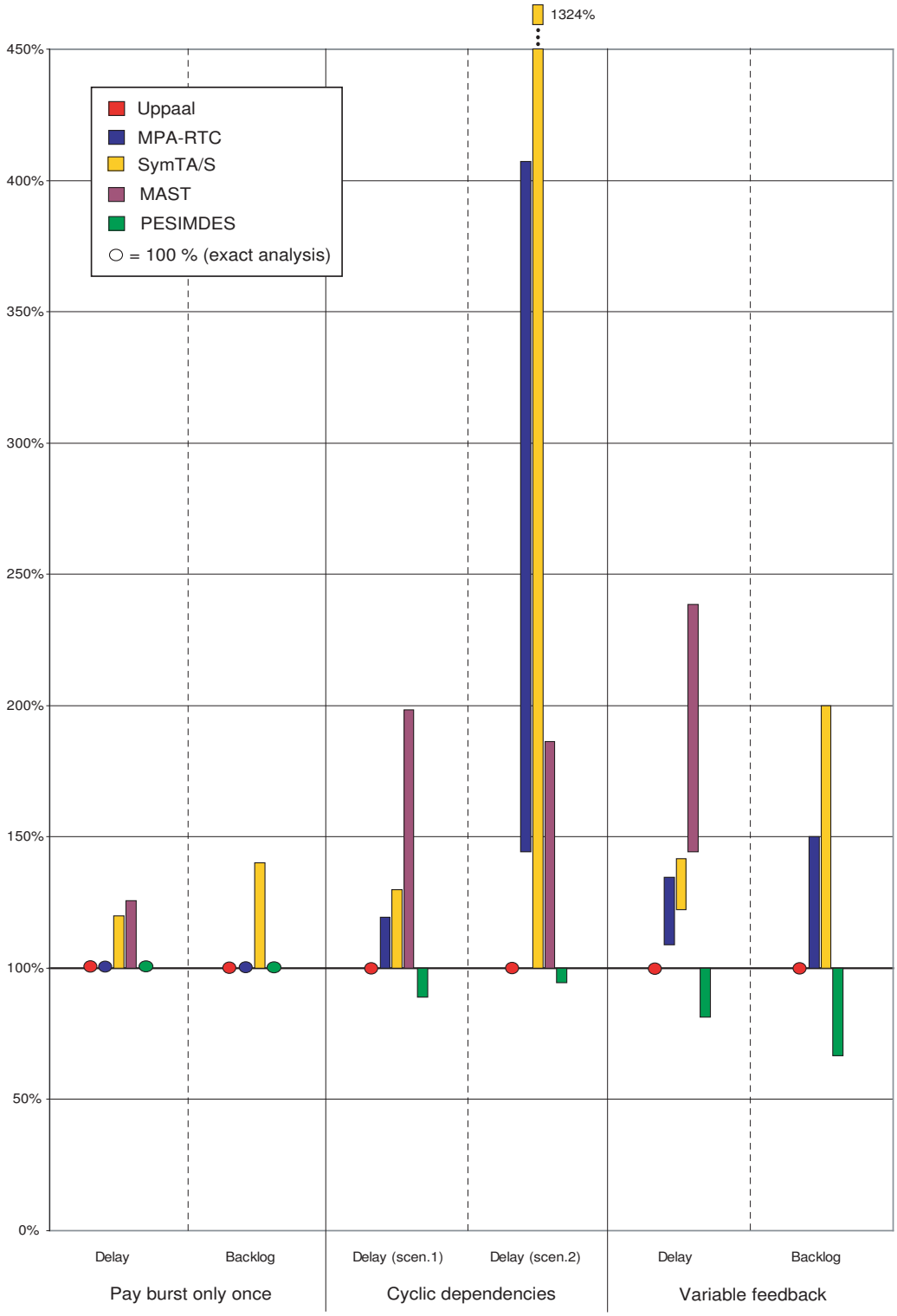


Figure 6.25: Overview of analysis accuracy (part 1)

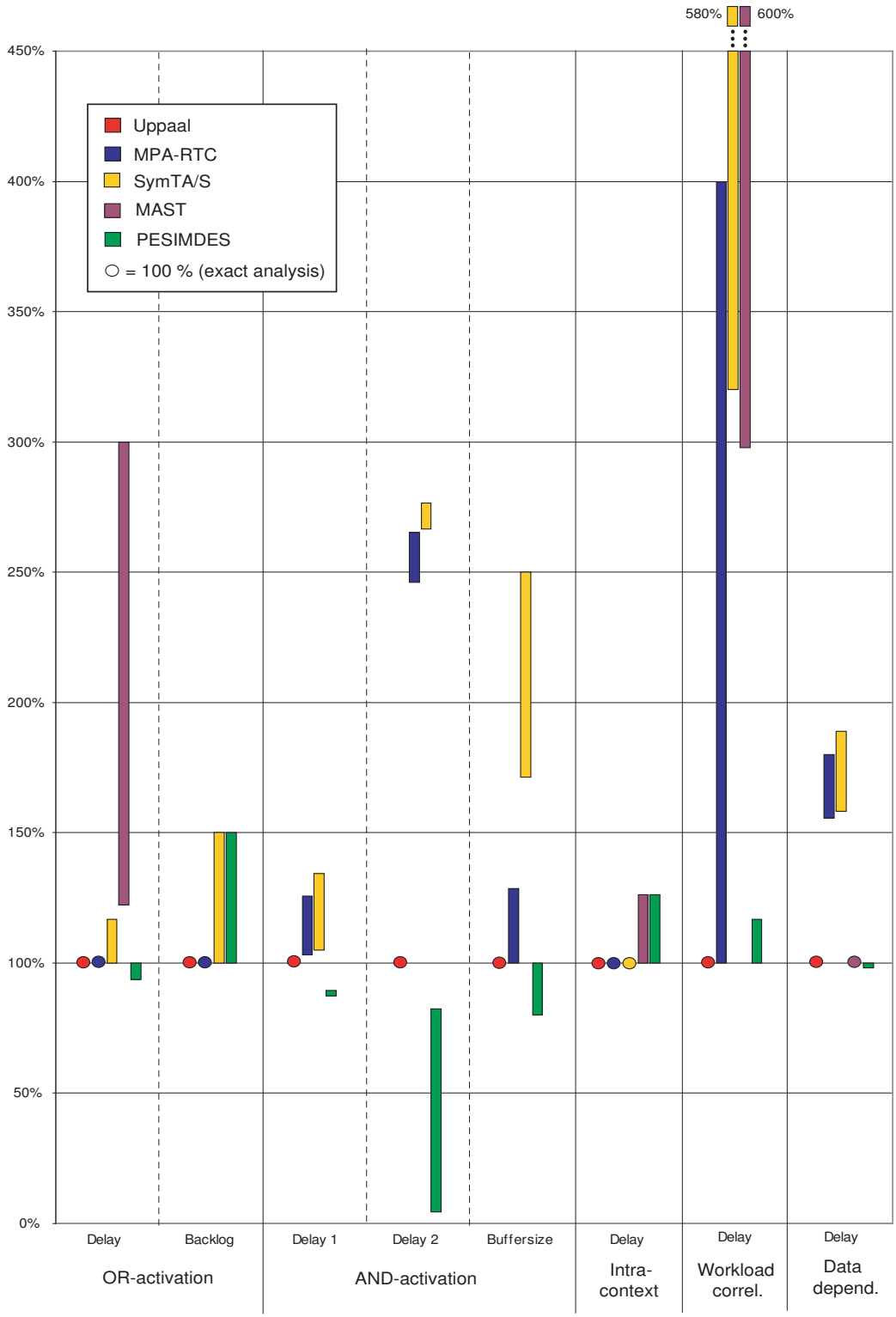


Figure 6.26: Overview of analysis accuracy (part 2)

The charts indicate that the performance analysis method based on the Uppaal model checker provides exact analysis results for all the case studies. Regarding the other performance analysis methods, one can see that for most of the case studies analyzed MPA-RTC provides better results than SymTA/S, which in turn provides better results than the MAST tool. However, there are also exceptions to this trend.

Overall, except for the Uppaal-approach based on model checking, no performance analysis method did better than another in all the case studies considered. This means that the accuracy of the different approaches significantly depends on the particular system characteristics and parameters.

Moreover, we would like to point out that the Uppaal approach is not the panacea for performance analysis: the exact results are often paid for by a huge analysis effort. For instance, for single parameter configurations of the case study *Variable Feedback* (see Section 6.3) the running time of the Uppaal model checker is up to a thousand times longer than the analysis time of the other methods.

Chapter 7

Conclusions

7.1 Conclusions

We have given an initial contribution to the evaluation and comparison of methods for performance analysis of distributed embedded systems. With respect to previous work this thesis has shifted the focus from the presentation and refinement of single performance analysis approaches to their benchmarking and comparison.

We presented an overview of several formal approaches to performance analysis and examined the various methods with respect to analyzable performance metrics, modeling scope, accuracy, modularity, analysis effort and scalability. We also evaluated modeling effort, tool support and end-user complexity for the approaches considered. By means of such evaluations and comparisons we support system designers in choosing the performance analysis method that is most suitable for their particular requirements.

We showed that the different approaches are very heterogeneous in terms of modeling power and usability. In addition, the considered case studies demonstrated that there are important differences with respect to the accuracy of the various analysis methods. These results highlight some modeling difficulties and analysis pitfalls of the different approaches. In particular, the results can drive some research on the improvement of the single performance analysis methods.

Moreover, the comparison showed that the accuracy ranking of the various performance analysis methods varies for different systems. This means that no analysis approach always provides better results than another. Hence, it makes sense for a system designer to combine different performance analysis methods in order to obtain accurate performance predictions.

Further, we showed that it can be very useful to combine formal performance analysis with performance simulation. While in general there is no way to determine the accuracy of performance bounds provided by a formal method,

in combination with performance simulation more significant conclusions about the accuracy of the analysis may be reached.

7.2 Outlook

There are several interesting topics for future research in the context of this thesis.

First of all, the set of benchmark problems can and should be extended. In this thesis the evaluation and comparison of performance analysis methods is based on a few case studies reproducing important properties of distributed embedded systems. However, there are several other common system attributes that have not yet been considered. For instance it could be interesting to compare the various performance analysis methods with respect to the analysis of:

- systems with dynamic scheduling policies (e.g. EDF)
- systems with hierarchical scheduling
- systems with timing correlations among different event streams
- systems with control dependencies
- systems with shared resources that are accessed in mutual exclusion
- systems with traffic shaping

In addition, it would be useful to examine in more detail the scalability of the different performance analysis approaches by analyzing larger systems. This thesis contains some consideration of scalability in terms of modeling effort. However, it would also be very interesting to analyze the scalability of the different methods with respect to accuracy and analysis effort.

Furthermore, the evaluation and comparison should be extended to other performance analysis methods not considered in this thesis. Moreover, a comparison with approaches apart from formal analysis or simulation could be useful. For instance it would be interesting to consider stochastic performance analysis approaches.

Finally, it would be very helpful to automate the combined application of different performance analysis methods. In this thesis the first steps towards the automated combination of methods were already taken by defining the draft of a tool-independent data format for the description of distributed embedded systems and their performance analysis. This format can be extended and on

top of it appropriate transformers can be implemented, which convert the common system description into the proprietary formats of the single performance analysis tools.

Appendix A

An extendible data format for the description of distributed embedded systems

This appendix presents a draft of an XML data format for the description of distributed embedded system architectures and their performance analysis.

A.1 Motivation

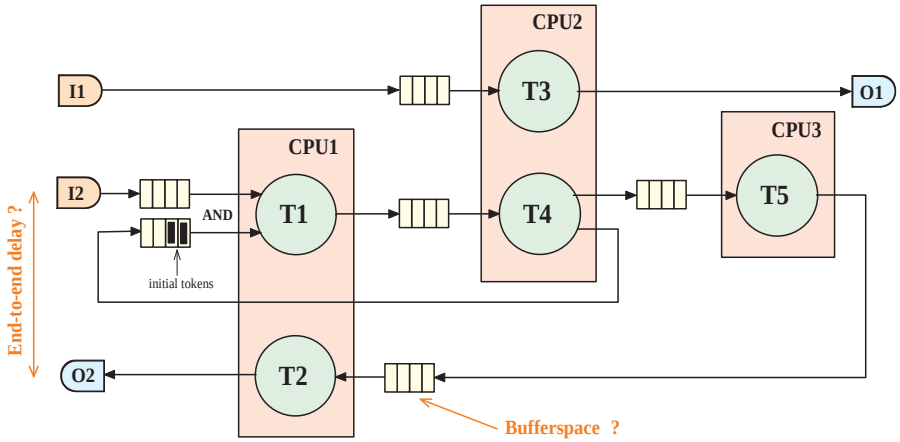
Every tool for performance analysis has its own way to specify the system architecture and the requested performance metrics. This makes the comparison of performance analysis tools difficult, as several different descriptions for the same system must be provided. To avoid this laborious and error-prone process we introduce a draft of a tool-independent and extendible data format for the description of distributed embedded systems and their performance analysis.

This XML format is a first step towards the automated combination of performance analysis methods: on top of the XML format appropriate transformers may be implemented, that convert the common system description into the proprietary formats of the single performance analysis tools.

A.2 Example

This section provides a simple example of system description in the proposed XML format.

Figure A.1 defines a distributed embedded system and the corresponding performance characteristics to analyze.



Input streams	I1: periodic with burst (P=20ms, J=55ms, d=2ms) I2: periodic (P=10ms)
Resource sharing	CPU1: FP preemptive CPU2: EDF preemptive
Task WCETs	T1: 2ms, T2: 1ms, T3: 7ms, T4: 3ms, T5: 8ms
Scheduling param.	priority T1: high, priority T2: low, rel. deadline T3: 18ms, rel. deadline T4: 35ms

Figure A.1: Example system

The corresponding XML description is given below.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<performance_analysis
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="perfest.xsd">

<system>

  <resources>
       <FP name="CPU1" preemptive="yes"/>
       <EDF name="CPU2" preemptive="yes"/>
       <FP name="CPU3" preemptive="yes"/>
  </resources>

  <event_sources>
       <PJD name="I1">
          <period value="20" unit="ms" />
          <jitter value="55" unit="ms" />
          <min_interarr value="2" unit="ms" />
       </PJD>
       <PJD name="I2">
          <period value="10" unit="ms" />
       </PJD>
  </event_sources>
```

```
  <event_sinks>
       <event_sink name="O1" />
       <event_sink name="O2" />
  </event_sinks>

  <tasks>
       <task name="T1" input_no="2" activation_type="AND" />
       <task name="T2" />
       <task name="T3" />
       <task name="T4" output_no="2" />
       <task name="T5" />
  </tasks>

  <task_graphs>
       <task_graph>
           <link src="I1" dest="T3" />
           <link src="T3" dest="O1" />
       </task_graph>
       <task_graph>
           <link src="I2" dest="T1" dest_index="0" />
           <link src="T1" dest="T4" />
           <link src="T4" src_index="0" dest="T5" />
           <link src="T4" src_index="1" dest="T1" dest_index="1" />
           <link src="T5" dest="T2" />
           <link src="T2" dest="O2" />
       </task_graph>
  </task_graphs>

  <binding>
       <map task="T1" resource="CPU1">
            <wcet value="2" unit="ms" />
            <priority> 1 </priority>
       </map>
       <map task="T2" resource="CPU1">
            <wcet value="1" unit="ms" />
            <priority> 2 </priority>
       </map>
       <map task="T3" resource="CPU2">
           <wcet value="7" unit="ms" />
           <relative_deadline value="18" unit="ms" />
       </map>
       <map task="T4" resource="CPU2">
           <wcet value="3" unit="ms" />
           <relative_deadline value="35" unit="ms" />
       </map>
       <map task="T5" resource="CPU3">
            <wcet value="8" unit="ms" />
            <priority> 1 </priority>
       </map>
  </binding>

  <state>
     <token task="T1" input_index="1" quantity="2"/>
  </state>

</system>


<observe>
   <latency src="I2" dest="O2" />
   <backlog task="T2" />
</observe>

</performance_analysis>
```

A.3 Description of the data format - XML Schema

The following XML Schema definition describes formally the structure of admissible system specifications in form of XML files.[1]

```
<?xml version="1.0" encoding="ISO-8859-1" ?> <xs:schema xmlns:xs=
"http://www.w3.org/2001/XMLSchema">


<!--*****************************__Root_element__************************************-->

<xs:element name="performance_analysis">
 <xs:complexType>
   <xs:sequence>
     <xs:element ref="system"/>
     <xs:element ref="observe"/>
   </xs:sequence>
 </xs:complexType>
</xs:element>

<!--*******************************************************************************-->

<xs:element name="system">
 <xs:complexType>
   <xs:sequence>
     <xs:element ref="resources"/>
     <xs:element ref="event_sources"/>
     <xs:element ref="event_sinks"/>
     <xs:element ref="tasks"/>
     <xs:element ref="task_graphs"/>
     <xs:element ref="binding"/>
     <xs:element ref="state" minOccurs="0"/>
   </xs:sequence>
 </xs:complexType>
</xs:element>


<xs:element name="observe">
 <xs:complexType>
  <xs:sequence>
    <xs:group ref="group_observe" />
    <xs:group ref="group_observe" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>

<!--*******************************************************************************-->

<xs:element name="resources">
 <xs:complexType>
  <xs:sequence>
    <xs:group ref="group_resources" />
    <xs:group ref="group_resources" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>


<xs:element name="event_sources">
 <xs:complexType>
  <xs:sequence>
    <xs:group ref="group_event_sources" />
    <xs:group ref="group_event_sources" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>


<xs:element name="event_sinks">
 <xs:complexType>
  <xs:sequence>
    <xs:element name="event_sink" maxOccurs="unbounded">
      <xs:complexType>
        <xs:attribute name="name" type="xs:ID" use="required"/>
      </xs:complexType>
    </xs:element>
```

[1] Attention should be paid to the limited expressiveness of XML Schema: not every inconsistency in the system description can be avoided through XML Schema restrictions! However, an appropriate pre-processor may be implemented to undertake the remaining consistency checks.

```
  </xs:sequence>
 </xs:complexType>
</xs:element>


<xs:element name="tasks">
 <xs:complexType>
  <xs:sequence>
    <xs:element name="task" maxOccurs="unbounded">
      <xs:complexType>
        <xs:attribute name="name" type="xs:ID" use="required"/>
        <xs:attribute name="input_no" type="xs:integer" use="optional"/>
        <xs:attribute name="activation_type" type="andor" use="optional"/>
        <xs:attribute name="output_no" type="xs:integer" use="optional"/>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>


<xs:element name="task_graphs">
 <xs:complexType>
  <xs:sequence>
    <xs:element name="task_graph" maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="link" maxOccurs="unbounded">
            <xs:complexType>
              <xs:attribute name="src" type="xs:IDREF" use="required"/>
              <xs:attribute name="dest" type="xs:IDREF" use="required"/>
              <xs:attribute name="src_index" type="xs:integer" use="optional"/>
              <xs:attribute name="dest_index" type="xs:integer" use="optional"/>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>


<xs:element name="binding">
 <xs:complexType>
  <xs:sequence>
    <xs:element name="bind" maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="wcet" type="time" />
          <xs:element name="priority" type="xs:integer" minOccurs="0" />
          <xs:element name="rel_deadline" type="time" minOccurs="0" />
          <xs:element name="slot_no" type="xs:integer" minOccurs="0" />
        </xs:sequence>
        <xs:attribute name="task" type="xs:IDREF" use="required"/>
        <xs:attribute name="resource" type="xs:IDREF" use="required"/>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>


<xs:element name="state">
 <xs:complexType>
  <xs:sequence>
    <xs:element name="token" maxOccurs="unbounded">
      <xs:complexType>
        <xs:attribute name="task" type="xs:IDREF" use="required"/>
        <xs:attribute name="input_index" type="xs:integer" use="optional"/>
        <xs:attribute name="quantity" type="xs:integer" use="optional"/>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>


<!--*****************************************************************************-->

<xs:group name="group_observe">
   <xs:choice>
     <xs:element name="latency">
       <xs:complexType>
         <xs:attribute name="src" type="xs:IDREF" use="required"/>
         <xs:attribute name="dest" type="xs:IDREF" use="required"/>
       </xs:complexType>
     </xs:element>
     <xs:element name="backlog">
       <xs:complexType>
```

```
         <xs:attribute name="task" type="xs:IDREF" use="required"/>
         <xs:attribute name="input_index" type="xs:integer" use="optional"/>
       </xs:complexType>
     </xs:element>
   </xs:choice>
</xs:group>

<!--*****************************************************************************-->

<xs:group name="group_resources">
   <xs:choice>
     <xs:element ref="FP"/>
     <xs:element ref="EDF"/>
     <xs:element ref="TDMA"/>
   </xs:choice>
</xs:group>


<xs:element name="FP" type="resource_base"/>

<xs:element name="EDF" type="resource_base"/>

<xs:element name="TDMA">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="resource_base">
         <xs:sequence>
            <xs:element name="slots">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="slot" maxOccurs="unbounded">
                        <xs:complexType>
                           <xs:attribute name="name" type="xs:string" use="required"/>
                           <xs:attribute name="length" type="xs:decimal" use="required"/>
                           <xs:attribute name="unit" type="timeunit" use="required"/>
                        </xs:complexType>
                     </xs:element>
                  </xs:sequence>
               </xs:complexType>
            </xs:element>
         </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:element>


<xs:complexType name="resource_base">
  <xs:attribute name="name" type="xs:ID" use="required"/>
  <xs:attribute name="preemptive" type="yesno" use="required"/>
</xs:complexType>


<!--*****************************************************************************-->

<xs:group name="group_event_sources">
   <xs:choice>
     <xs:element ref="PJD"/>
     <xs:element ref="arrival_curve"/>
     <xs:element ref="trace"/>
   </xs:choice>
</xs:group>


<xs:element name="PJD">
 <xs:complexType>
   <xs:all>
     <xs:element name="period" type="time"/>
     <xs:element name="jitter" type="time" minOccurs="0"/>
     <xs:element name="min_interarr" type="time" minOccurs="0"/>
   </xs:all>
   <xs:attribute name="name" type="xs:ID" use="required"/>
 </xs:complexType>
</xs:element>


<xs:element name="arrival_curve">
 <xs:complexType>
   <xs:all>
     <xs:element name="upper" type="xs:string"/>
     <xs:element name="lower" type="xs:string"/>
   </xs:all>
   <xs:attribute name="name" type="xs:ID" use="required"/>
 </xs:complexType>
</xs:element>


<xs:element name="trace">
 <xs:complexType>
   <xs:attribute name="name" type="xs:ID" use="required"/>
```

```
   <xs:attribute name="file" type="xs:string" use="required"/>
 </xs:complexType>
</xs:element>


<!--*****************************__Simple_types__***********************************-->

<xs:simpleType name="yesno">
  <xs:restriction base="xs:string">
     <xs:enumeration value="yes"/>
     <xs:enumeration value="no"/>
  </xs:restriction>
</xs:simpleType>


<xs:simpleType name="timeunit">
  <xs:restriction base="xs:string">
     <xs:enumeration value="s"/>
     <xs:enumeration value="ms"/>
     <xs:enumeration value="us"/>
     <xs:enumeration value="ns"/>
     <xs:enumeration value="ps"/>
     <xs:enumeration value="fs"/>
  </xs:restriction>
</xs:simpleType>


<xs:simpleType name="andor">
  <xs:restriction base="xs:string">
     <xs:enumeration value="and"/>
     <xs:enumeration value="or"/>
  </xs:restriction>
</xs:simpleType>

<!--*****************************************************************************-->

<xs:complexType name="time">
  <xs:attribute name="value" type="xs:decimal" use="required"/>
  <xs:attribute name="unit" type="timeunit" use="required"/>
</xs:complexType>

<!--*****************************************************************************-->


</xs:schema>
```

Appendix B

PESIMDES User Guide

This appendix provides a short user guide for the PESIMDES library.[1] Only basic SystemC skills are required to use the PESIMDES library for performance estimation of distributed embedded systems.

B.1 Setup

PESIMDES is built on top of the SystemC 2.1 C++ class library provided by OSCI[2]. It is necessary to download, compile and install the SystemC library in order to use PESIMDES. Please refer to *http://www.systemc.org/* for more information about the setup of SystemC.

In order to use PESIMDES components it is necessary to download the PESIMDES library and include it in your SystemC program:

```
#include <pesimdes.h>
```

Make sure that your compiler and linker can find the corresponding files of the PESIMDES library.

B.2 Modeling

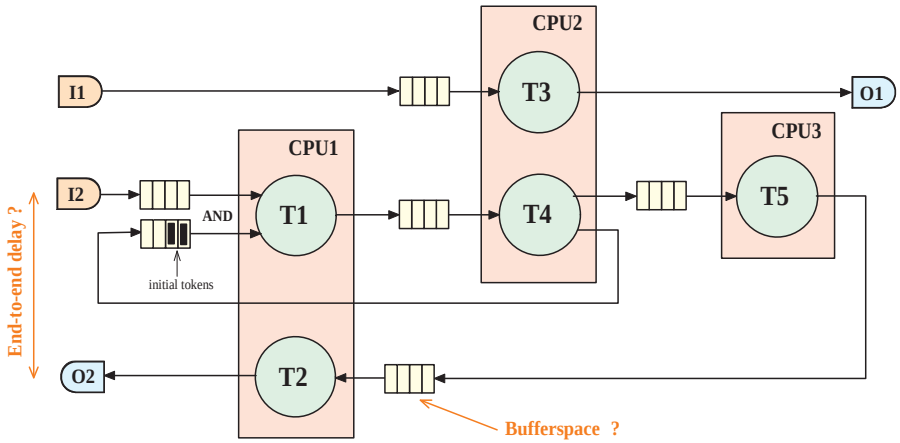
To model a distributed embedded system with PESIMDES for performance analysis, it is sufficient to write a simple SystemC program that instantiates and connects proper modules of the PESIMDES library. In particular it is necessary to:

- instantiate the event stream generators

[1] available as open source software at *http://www.mpa.ethz.ch*
[2] The Open SystemC Initiative *http://www.systemc.org/*

- instantiate the event sinks
- instantiate the tasks
- instantiate the activation buffers
- instantiate the processing / communication resources
- assign the tasks to resources
- link the event generators, tasks, activation buffers and event sinks

In the following the modeling procedure is demonstrated considering the system depicted in Figure B.1 as example.



Input streams	I1: periodic with burst (P=20ms, J=55ms, d=2ms) I2: periodic (P=10ms)
Resource sharing	CPU1: FP preemptive CPU2: EDF preemptive
Task WCETs	T1: 2ms, T2: 1ms, T3: 7ms, T4: 3ms, T5: 8ms
Scheduling param.	priority T1: high, priority T2: low, rel. deadline T3: 18ms, rel. deadline T4: 35ms

Figure B.1: Example system

The following listing reports the SystemC code that models the system of Figure B.1 using components of the PESIMDES library. The instantiation of the event stream generators, tasks, activation buffers, resources and event sinks is straightforward. More documentation about the single constructor calls can

be found at the PESIMDES webpage.[3] The same applies for the assignment of tasks to resources and the connection of the various modules.

```
#include "pesimdes.h"

#define MAXBUFFER 1000

bool verbose = false;


int sc_main (int argc , char *argv[]) {

     // initialize the random number generator
    srand(-24);


    // Module instantiations

    // Event stream generators
    input_periodic_with_burst_uwj input_generator_1 ("I1",sc_time(20,
        SC_MS),sc_time(55,SC_MS),sc_time(2,SC_MS),"input_trace_1.tra");
    input_periodic input_generator_2 ("I2",sc_time(10,SC_MS),"
        input_trace_2.tra");

    // Tasks
    task t1("T1",2,"AND",1);
    task t2("T2");
    task t3("T3");
    task t4("T4",2);
    task t5("T5");

    // Resources
    resource_n_tasks_fixed_priority_preemptive cpu_1("CPU1",2);
    resource_n_tasks_EDF_preemptive cpu_2("CPU2",2);
    resource_n_tasks_fixed_priority_preemptive cpu_3("CPU3",1);

    // Event sinks
    output_display display_1("O1");
    output_display display_2("O2");


    // Task mapping
    cpu_1.assign_task(t1,0,sc_time(2,SC_MS),1);
    cpu_1.assign_task(t2,1,sc_time(1,SC_MS),2);
    cpu_2.assign_task(t3,0,sc_time(7,SC_MS),sc_time(18,SC_MS));
    cpu_2.assign_task(t4,1,sc_time(3,SC_MS),sc_time(35,SC_MS));
    cpu_3.assign_task(t5,0,sc_time(8,SC_MS),1);


    // Channel instantiations

    // Task activation buffers
    my_sc_fifo<event_token> buffer_T1_1(MAXBUFFER);
    my_sc_fifo<event_token> buffer_T1_2(MAXBUFFER);
    my_sc_fifo<event_token> buffer_T2(MAXBUFFER);
    my_sc_fifo<event_token> buffer_T3(MAXBUFFER);
    my_sc_fifo<event_token> buffer_T4(MAXBUFFER);
    my_sc_fifo<event_token> buffer_T5(MAXBUFFER);
```

[3] *http://www.mpa.ethz.ch*

```
55      // Dummy buffers for event sinks
56      sc_fifo<event_token> dummy_buffer_display_1(1);
57      sc_fifo<event_token> dummy_buffer_display_2(1);


60      // Port binding

62      // Stream I1-O1
63      input_generator_1.out(buffer_T3);
64      t3.in[0](buffer_T3);
65      t3.out[0](dummy_buffer_display_1);
66      display_1.in(dummy_buffer_display_1);

68      // Stream I2-O2
69      input_generator_2.out(buffer_T1_1);
70      t1.in[0](buffer_T1_1);
71      t1.out[0](buffer_T4);
72      t4.in[0](buffer_T4);
73      t4.out[0](buffer_T5);
74      t4.out[1](buffer_T1_2);
75      t1.in[1](buffer_T1_2);
76      t5.in[0](buffer_T5);
77      t5.out[0](buffer_T2);
78      t2.in[0](buffer_T2);
79      t2.out[0](dummy_buffer_display_2);
80      display_2.in(dummy_buffer_display_2);


83      // Initial state
84      set_initial_token(buffer_T1_2,2);


87      // Start simulation
88      sc_start(100000,SC_MS);


91      // Print performance results
92      cout << "\n␣Max.␣observed␣End-to-End␣Delay␣from␣I2␣to␣O2:␣" <<
            sc_time_output(display_2.getmaxlatency("I2")) << "␣(event␣no.␣"
             << display_2.getseqnoofmaxlatency("I2") << ")\n";
93      cout << "Max.␣observed␣Backlog␣T2:␣" << buffer_T2.getmaxbacklog()
            << "\n";

95      return 0;
96  }
```

Listing B.1: SystemC model for the example system based on the PESIMDES library

B.3 Simulation

Once the topology of the system is modeled by instantiating and connecting proper PESIMDES modules, it is necessary to configure the simulation. In particular the following simulation parameters must be set:

Maximum Buffersize

A maximum size for the task activation buffers must be specified (see line 3 of Listing B.1). Note that the buffer size does not influence the behavior

of the system, i.e. a full buffer does not block a writing task. In case of a buffer overflow an appropriate exception is raised and the simulation is stopped.

Simulation length

The length of the simulation must be specified as parameter of the `sc_start` call (see line 88 of Listing B.1), which determines the duration of the SystemC simulation. Note that by this parameter you specify the amount of time to simulate and not the running time of the simulation.

Simulation outputs

In order to obtain the estimations for the requested performance characteristics of the system it is necessary to output the return values of the functions `getmaxlatency` and `getmaxbacklog` called on the corresponding components (see lines 92 and 93 of Listing B.1)

Random number generator

Various event generators make use of a random number generator. The corresponding seed value can be set explicitly (see line 11 of Listing B.1). By choosing twice the same seed number the results of a simulation can be reproduced.

Verbose option

It is possible to instruct the PESIMDES components to print to standard output a detailed simulation log. This is done by setting the verbose flag (see line 5 of Listing B.1). The log traces generation, processing and propagation of every single event in the system. Note that the use of the verbose feature may increase significantly the running time of the simulation.

Appendix C

Case studies - System models

This appendix contains the models used to analyze the performance characteristics of the systems presented in chapter 6. The SymTA/S models were generated with a graphical user interface and are reported in form of screenshots. The same applies for the Uppaal models, in this case we depict only the corresponding timed automata networks. The corresponding SymTA/S and Uppaal files are available online (as well as all other models reported in this appendix).[1]

The following list specifies the versions of the tools used to perform the various analyses:

- RTC Toolbox for Matlab v1.0 beta revision 319
- SymTA/S v0.8 beta EVAL
- MAST v1.3.6
- UPPAAL v3.5.9
- PESIMDES v0.1 beta

[1] *http://www.mpa.ethz.ch*

C.1 Models case study 1: Pay burst only once

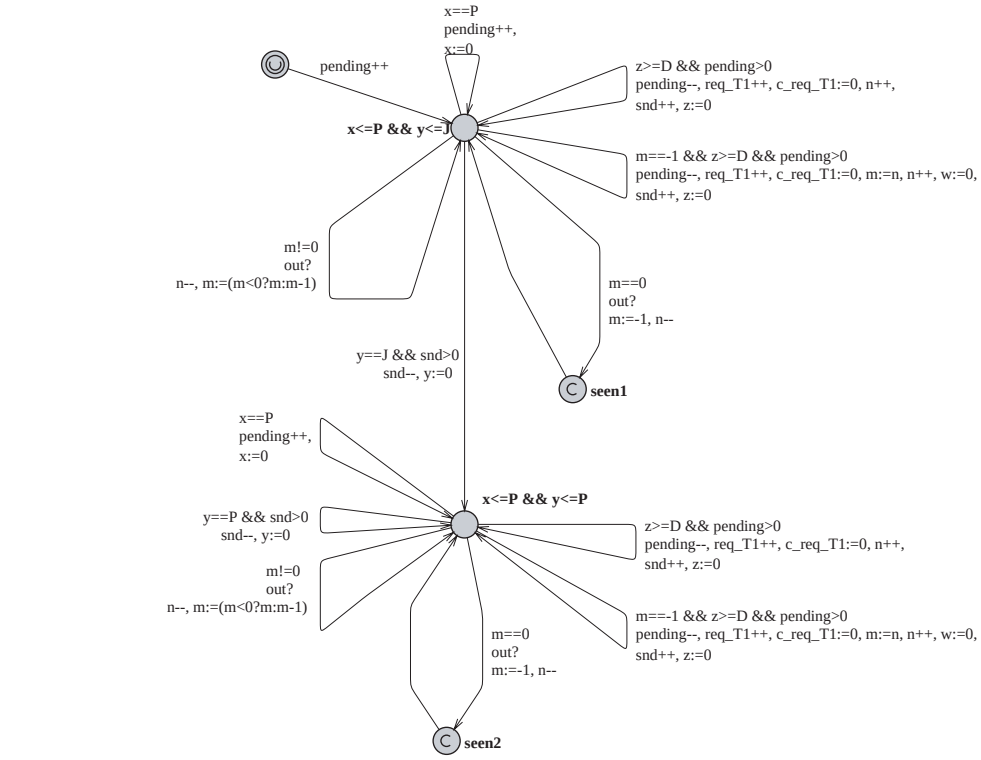
MPA-RTC

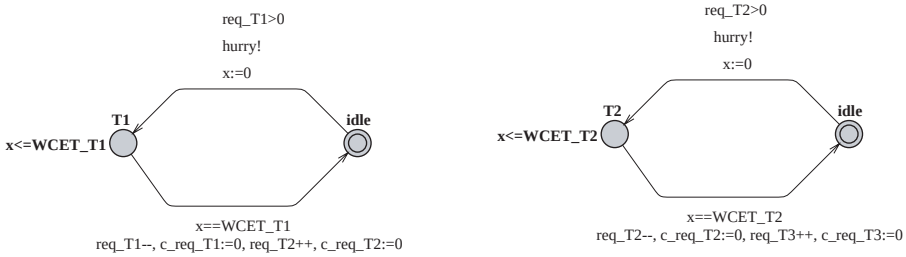
```
% Author(s): E. Wandeler
% Copyright 2004-2006 Computer Engineering and Networks Laboratory (TIK)
% ETH Zurich, Switzerland.
% All rights reserved.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Setup of system parameters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Event stream
% ========================================================================
p = 10;
j = 50;
d = 1;

% Worst-case execution demands
% ========================================================================
wced1 = 1;
wced2 = 4;
wced3 = 8;

% Processor speeds
% ========================================================================
cpu1 = 1;
cpu2 = 1;
cpu3 = 1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Construct input curves
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
tic;
% Construct the input arrival curves
% ========================================================================
aui1 = pjdu(p,j,d);
ali1 = pjdl(p,j,d);

% Construct the service curves
% ========================================================================
bui1 = fs(cpu1);
bli1 = fs(cpu1);
bui2 = fs(cpu2);
bli2 = fs(cpu2);
bui3 = fs(cpu3);
bli3 = fs(cpu3);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Analysis.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Compute the arrival and service curves through a chain of Greedy
% Processing Components.
% ========================================================================
[auo1 alo1 buo1 blo1] = gpc(aui1, ali1, bui1, bli1, wced1);
[auo2 alo2 buo2 blo2] = gpc(auo1, alo1, bui2, bli2, wced2);
[auo3 alo3 buo3 blo3] = gpc(auo2, alo2, bui3, bli3, wced3);

% Compute the total delay through service curve convolution.
% ========================================================================
delay_conv = del(aui1,bli1,wced1,bli2,wced2,bli3,wced3);

tTot = toc;

% Compute the total delay as the sum from the individual delays.
% ========================================================================
delay_add = del(aui1,bli1,wced1) + del(auo1,bli2,wced2) + del(auo2,bli3,wced3);


% Display the results.
% ========================================================================
disp(['Total Delay (computed with convolutions) : ' num2str(delay_conv)])
disp(['Sum of the individual delays             : ' num2str(delay_add)])
disp(['Analysis Time                            : ' num2str(tTot) 's'])
```

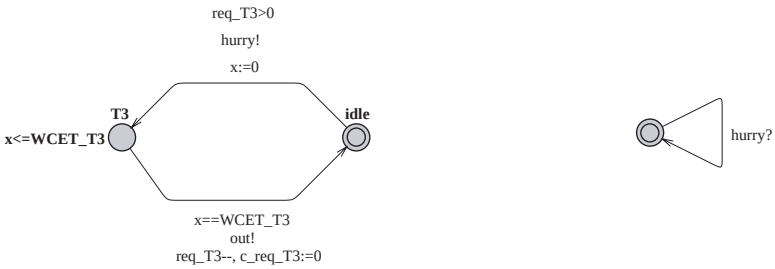
MAST

```
Model (
   Model_Name  => p9,
   Model_Date  => 2005-12-20);


Processing_Resource (
    Type            => Regular_Processor,
    Name            => CPU1);

Processing_Resource (
    Type            => Regular_Processor,
    Name            => CPU2);

Processing_Resource (
    Type            => Regular_Processor,
    Name            => CPU3);


Scheduler (
   Type            => Primary_Scheduler,
   Name            => CPU1,
   Host            => CPU1,
   Policy          =>
      ( Type      => Fixed_Priority));


Scheduler (
   Type            => Primary_Scheduler,
   Name            => CPU2,
   Host            => CPU2,
   Policy          =>
      ( Type      => Fixed_Priority));


Scheduler (
   Type            => Primary_Scheduler,
   Name            => CPU3,
   Host            => CPU3,
   Policy          =>
      ( Type      => Fixed_Priority));



Scheduling_Server (
    Type               => Regular,
    Name               => T1,
    Server_Sched_Parameters     => (
        Type => Fixed_Priority_policy,
        The_Priority => 1,
        Preassigned  => Yes),
    Scheduler          => CPU1);


Scheduling_Server (
    Type               => Regular,
    Name               => T2,
    Server_Sched_Parameters     => (
        Type => Fixed_Priority_policy,
        The_Priority => 1,
        Preassigned  => Yes),
    Scheduler          => CPU2);


Scheduling_Server (
    Type               => Regular,
    Name               => T3,
    Server_Sched_Parameters     => (
        Type => Fixed_Priority_policy,
        The_Priority => 1,
        Preassigned => Yes),
    Scheduler          => CPU3);


Operation (
    Type                      => Simple,
    Name                      => T1,
    Best_Case_Execution_Time => 1,
    Worst_Case_Execution_Time => 1);


Operation (
    Type                      => Simple,
    Name                      => T2,
    Best_Case_Execution_Time => 4,
    Worst_Case_Execution_Time => 4);


Operation (
    Type                      => Simple,
    Name                      => T3,
    Best_Case_Execution_Time => 8,
    Worst_Case_Execution_Time => 8);



Transaction (
    Type => Regular,
    Name => transact1,
    External_Events => (
        (Type   => Periodic,
         Name   => in,
         Period => 10,
                  Max_Jitter => 70)),
    Internal_Events => (
        (Type   => regular,
         name   => T1_out),
        (Type   => regular,
         name   => T2_out),
        (Type   => regular,
         name   => T3_out,
                  Timing_Requirements => (
            Type         => Hard_Global_Deadline,
            Deadline     => 1000,
            Referenced_Event => in))),
    Event_Handlers => (
        (Type           => Activity,
         Input_Event        => in,
         Output_Event       => T1_out,
         Activity_Operation => T1,
         Activity_Server    => T1),
        (Type           => Activity,
         Input_Event        => T1_out,
         Output_Event       => T2_out,
         Activity_Operation => T2,
         Activity_Server    => T2),
        (Type           => Activity,
         Input_Event        => T2_out,
         Output_Event       => T3_out,
         Activity_Operation => T3,
         Activity_Server    => T3)));
```

Uppaal


x==P
pending++,
x:=0
pending++
x<=P && y<=J
z>=D && pending>0
pending--, req_T1++, c_req_T1:=0, n++,
snd++, z:=0
m==-1 && z>=D && pending>0
pending--, req_T1++, c_req_T1:=0, m:=n, n++, w:=0,
snd++, z:=0
m!=0
out?
n--, m:=(m<0?m:m-1)
m==0
out?
m:=-1, n--
y==J && snd>0
snd--, y:=0
seen1
x==P
pending++,
x:=0
x<=P && y<=P
y==P && snd>0
snd--, y:=0
z>=D && pending>0
pending--, req_T1++, c_req_T1:=0, n++,
snd++, z:=0
m!=0
out?
n--, m:=(m<0?m:m-1)
m==-1 && z>=D && pending>0
pending--, req_T1++, c_req_T1:=0, m:=n, n++, w:=0,
snd++, z:=0
m==0
out?
m:=-1, n--
seen2

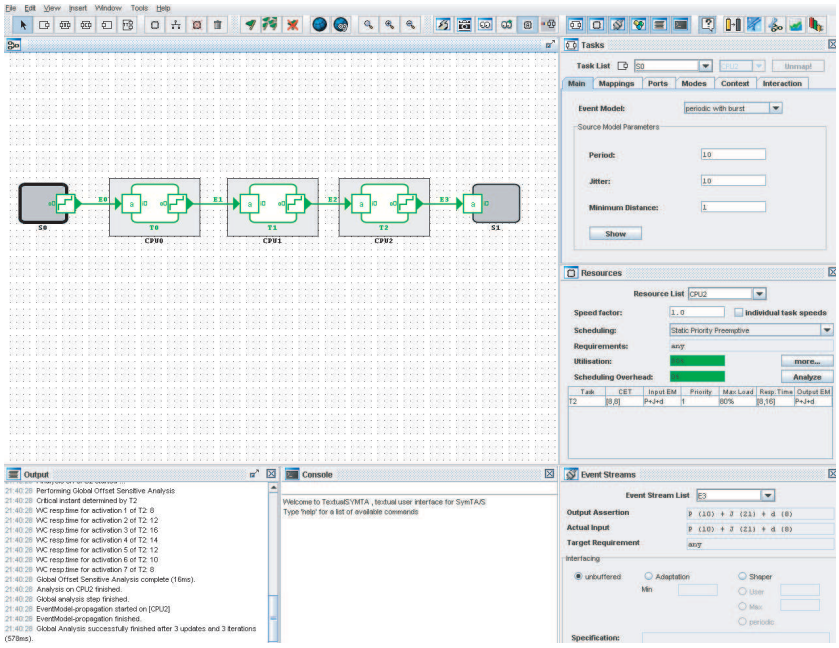

req_T1>0
hurry!
x:=0
T1
idle
x<=WCET_T1
x==WCET_T1
req_T1--, c_req_T1:=0, req_T2++, c_req_T2:=0
req_T2>0
hurry!
x:=0
T2
idle
x<=WCET_T2
x==WCET_T2
req_T2--, c_req_T2:=0, req_T3++, c_req_T3:=0


req_T3>0
hurry!
x:=0
T3
idle
x<=WCET_T3
x==WCET_T3
out!
req_T3--, c_req_T3:=0
hurry?

PESIMDES

```
#include "pesimdes.h"
#define MAXBUFFER 1000
bool verbose = false;

int sc_main (int argc , char *argv[]) {

    // initialize the random number generator
   srand(-11);

   // Module instantiations

   // Event stream generator
   input_periodic_with_burst_uwj input_generator
     ("I1",sc_time(10,SC_MS),sc_time(50,SC_MS),sc_time(1,SC_MS),"input_trace_1.tra");
   /* const double change_state_probabilities[3][3] = {{0.9, 0.05, 0.05}, {0.05, 0.9, 0.05}, {0.05, 0.05, 0.9}};
      input_periodic_with_burst_configurable_distribution input_generator
        ("I1",sc_time(10,SC_MS),sc_time(50,SC_MS),sc_time(1,SC_MS),change_state_probabilities,
          'u',"input_trace_1_b.tra");    */

   // Tasks
   task t1("T1");
   task t2("T2");
   task t3("T3");

   // Resources
   resource_n_tasks_fixed_priority_preemptive cpu_1("CPU1",1);
   resource_n_tasks_fixed_priority_preemptive cpu_2("CPU2",1);
   resource_n_tasks_fixed_priority_preemptive cpu_3("CPU3",1);

   // Event sinks
   output_display display("O1");

   // Task mapping
   cpu_1.assign_task(t1,0,sc_time(1,SC_MS),1);
   cpu_2.assign_task(t2,0,sc_time(4,SC_MS),1);
   cpu_3.assign_task(t3,0,sc_time(8,SC_MS),1);

   // Channel instantiations

   // Task activation buffers
   my_sc_fifo<event_token> buffer_T1(MAXBUFFER);
   my_sc_fifo<event_token> buffer_T2(MAXBUFFER);
   my_sc_fifo<event_token> buffer_T3(MAXBUFFER);

   // Dummy buffer for event sink
   sc_fifo<event_token> dummy_buffer_display(1);

   // Port binding

   input_generator.out(buffer_T1);
   t1.in[0](buffer_T1);
   t1.out[0](buffer_T2);
   t2.in[0](buffer_T2);
   t2.out[0](buffer_T3);
   t3.in[0](buffer_T3);
   t3.out[0](dummy_buffer_display);
   display.in(dummy_buffer_display);

   // Start simulation
   sc_start(100000,SC_MS);

   // Print performance results
   cout << "\n Max. observed End-to-End Delay: "
        << sc_time_output(display.getmaxlatency("I1")) <<
        " (event no. " << display.getseqnoofmaxlatency("I1") << ")\n";
   cout << "Max. observed Backlog T3: " << buffer_T3.getmaxbacklog() << "\n";

   return 0;
}
```

SymTA/S



XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<performance_analysis
xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="perfest.xsd">

<system>

  <resources>
       <FP name="CPU1" preemptive="yes"/>
       <EDF name="CPU2" preemptive="yes"/>
       <FP name="CPU3" preemptive="yes"/>
  </resources>

  <event_sources>
       <PJD name="I1">
          <period value="10" unit="ms" />
          <jitter value="50" unit="ms" />
          <min_interarr value="1" unit="ms" />
       </PJD>
  </event_sources>

  <event_sinks>
       <event_sink name="O1" />
  </event_sinks>

  <tasks>
       <task name="T1" />
       <task name="T2" />
       <task name="T3" />
  </tasks>

  <task_graphs>
       <task_graph>
           <link src="I1" dest="T1" />
           <link src="T1" dest="T2" />
           <link src="T2" dest="T3" />
           <link src="T3" dest="O1" />
       </task_graph>
  </task_graphs>

  <binding>
       <map task="T1" resource="CPU1">
            <wcet value="1" unit="ms" />
            <priority> 1 </priority>
       </map>
       <map task="T2" resource="CPU2">
            <wcet value="4" unit="ms" />
            <priority> 1 </priority>
       </map>
       <map task="T3" resource="CPU3">
           <wcet value="8" unit="ms" />
            <priority> 1 </priority>
       </map>
  </binding>

</system>

<observe>
   <latency src="I1" dest="O1" />
   <backlog task="T3" />
</observe>

</performance_analysis>
```

C.2 Models case study 2: Cyclic dependencies

MPA-RTC

```
% Author(s): E. Wandeler
% Copyright 2004-2006 Computer Engineering and Networks Laboratory (TIK)
% ETH Zurich, Switzerland.
% All rights reserved.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Setup of system parameters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Event stream
% ========================================================================
p = 10; j = 50; d = 0;

% Processor speeds
% ========================================================================
cpu1 = 1; cpu2 = 1;

% Task execution demands
% ========================================================================
wced1 = 1; wced2 = 4; wced3 = 4;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Construct input curves
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Construct the input arrival curves
% ========================================================================
aui1 = pjdu(p,j,d); ali1 = pjdl(p,j,d);

% Construct the service curves
% ========================================================================
bui1 = fs(cpu1); bli1 = fs(cpu1); bui2 = fs(cpu2); bli2 = fs(cpu2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Analysis of scenario 1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
tic;

% Compute the arrival and service curves for the stream
% ========================================================================
[auo1 alo1 buo1 blo1] = gpc(aui1, ali1, bui1, bli1, wced1);
[auo2 alo2 buo2 blo2] = gpc(auo1, alo1, bui2, bli2, wced2);
[auo3 alo3 buo3 blo3] = gpc(auo2, alo2, buo1, blo1, wced3);

% Compute the delay using both methods: addition and convolution
% ========================================================================
delay_conv = del(aui1,bli1,wced1,bli2,wced2,blo1,wced3);
delay_add = del(aui1,bli1,wced1) + del(auo1,bli2,wced2) + del(auo2,blo1,wced3);
delay = min(delay_conv, delay_add);

tTot = toc;

% Display the results
% ========================================================================
disp(['Scenarion 1: Priority(T3) < Priority(T1)'])
disp(['========================================================================'])
disp(['Delay         : ' num2str(delay)]) disp(['Analysis time : '
num2str(tTot) 's']) disp(' ')


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Analysis of scenario 2
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This analysis requires a fixed-point calculation
tic;

% Compute the arrival and service curves for the stream
% ========================================================================
% Set start values for fixed-point calculation
buo3 = bui1; blo3 = bli1;
buo3last = +buo3; % unary plus makes a clone!
blo3last = +blo3;

% Iterate up to 20 times
for i = [1:20]
    [auo1 alo1 buo1 blo1] = gpc(aui1, ali1, buo3, blo3, wced1);
    [auo2 alo2 buo2 blo2] = gpc(auo1, alo1, bui2, bli2, wced2);
    [auo3 alo3 buo3 blo3] = gpc(auo2, alo2, bui1, bli1, wced3);
    if ((buo3 == buo3last) & (blo3 == blo3last))
        break
    end
    buo3last = +buo3; % unary plus makes a clone!
    blo3last = +blo3;
end
```

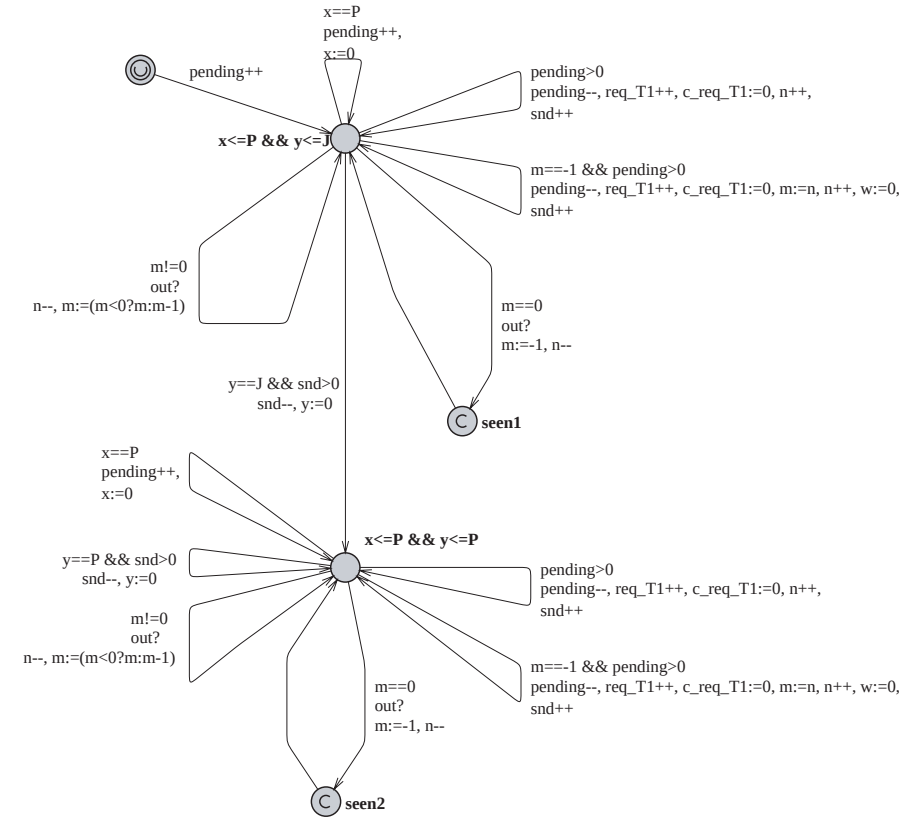
```
% Compute the delay using both methods: addition and convolution
% ======================================================================
delay_conv = del(aui1,blo3,wced1,bli2,wced2,bli1,wced3);
delay_add = del(aui1,blo3,wced1) + del(auo1,bli2,wced2) + del(auo2,bli1,wced3);
delay = min(delay_conv, delay_add);

tTot = toc;

% Display the results
% ======================================================================
disp(['Scenarion 2: Priority(T3) > Priority(T1) (fixed-point
calculation)'])
disp(['============================================================='])
disp(['Delay         : ' num2str(delay)]) disp(['Analysis time : '
num2str(tTot) 's']) if (((buo3 == buo3last) & (blo3 == blo3last)))
    disp(['Fixed point reached after ' num2str(i) ' iterations.'])
else
    disp(['Fixed point NOT reached after ' num2str(i) ' iterations.'])
end disp(' ')
```

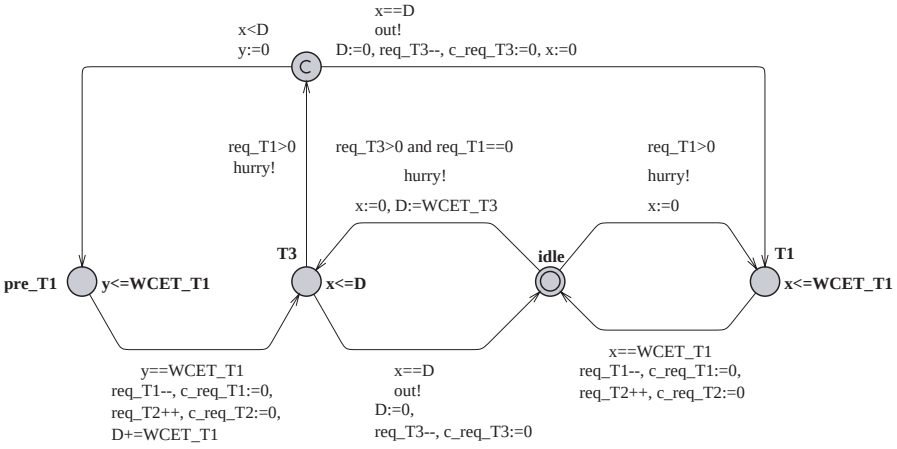
MAST

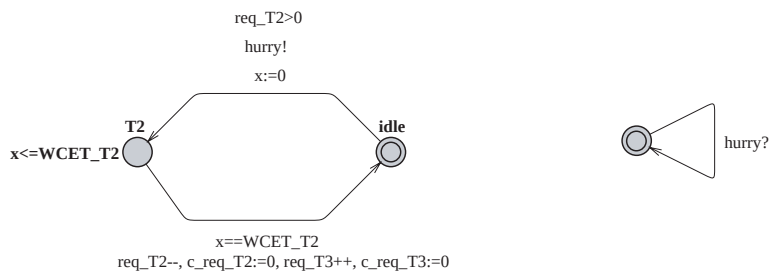
```
Model (
   Model_Name  => cyclic_dependencies,
   Model_Date  => 2005-12-13);

Processing_Resource (
    Type            => Regular_Processor,
    Name            => CPU1);

Processing_Resource (
    Type            => Regular_Processor,
    Name            => CPU2);

Scheduler (
   Type            => Primary_Scheduler,
   Name            => CPU1,
   Host            => CPU1,
   Policy          =>
      ( Type    => Fixed_Priority));

Scheduler (
   Type            => Primary_Scheduler,
   Name            => CPU2,
   Host            => CPU2,
   Policy          =>
      ( Type    => Fixed_Priority));

Scheduling_Server (
    Type               => Regular,
    Name               => T1,
    Server_Sched_Parameters     => (
        Type        => Fixed_Priority_policy,
        The_Priority        => 2,
--      The_Priority        => 1,
        Preassigned         => Yes),
    Scheduler          => CPU1);

Scheduling_Server (
    Type               => Regular,
    Name               => T2,
    Server_Sched_Parameters     => (
        Type        => Fixed_Priority_policy,
        The_Priority        => 1,
        Preassigned         => Yes),
    Scheduler          => CPU2);

Scheduling_Server (
    Type               => Regular,
    Name               => T3,
    Server_Sched_Parameters     => (
        Type        => Fixed_Priority_policy,
        The_Priority        => 1,
--      The_Priority        => 2,
        Preassigned         => Yes),
    Scheduler          => CPU1);
```

```
Operation (
    Type                       => Simple,
    Name                       => T1,
    Best_Case_Execution_Time => 1,
    Worst_Case_Execution_Time => 1);

Operation (
    Type                       => Simple,
    Name                       => T2,
    Best_Case_Execution_Time => 4,
    Worst_Case_Execution_Time => 4);

Operation (
    Type                       => Simple,
    Name                       => T3,
    Best_Case_Execution_Time => 4,
    Worst_Case_Execution_Time => 4);

Transaction (
    Type => Regular,
    Name => transact1,
    External_Events => (
        (Type   => Periodic,
         Name   => in,
         Period => 10,
                Max_Jitter => 47.5)),
    Internal_Events => (
        (Type   => regular,
         name   => T1_out),
        (Type   => regular,
         name   => T2_out),
        (Type   => regular,
         name   => T3_out,
                Timing_Requirements => (
            Type         => Hard_Global_Deadline,
            Deadline     => 1000,
            Referenced_Event => in))),
    Event_Handlers => (
        (Type           => Activity,
         Input_Event        => in,
         Output_Event       => T1_out,
         Activity_Operation => T1,
         Activity_Server    => T1),
        (Type           => Activity,
         Input_Event        => T1_out,
         Output_Event       => T2_out,
         Activity_Operation => T2,
         Activity_Server    => T2),
        (Type           => Activity,
         Input_Event        => T2_out,
         Output_Event       => T3_out,
         Activity_Operation => T3,
         Activity_Server    => T3)));
```

Uppaal


x==P
pending++,
x:=0
pending++
pending>0
pending--, req_T1++, c_req_T1:=0, n++,
snd++
x<=P && y<=J
m==-1 && pending>0
pending--, req_T1++, c_req_T1:=0, m:=n, n++, w:=0,
snd++
m!=0
out?
n--, m:=(m<0?m:m-1)
m==0
out?
m:=-1, n--
y==J && snd>0
snd--, y:=0
seen1
x==P
pending++,
x:=0
x<=P && y<=P
y==P && snd>0
snd--, y:=0
pending>0
pending--, req_T1++, c_req_T1:=0, n++,
snd++
m!=0
out?
n--, m:=(m<0?m:m-1)
m==-1 && pending>0
pending--, req_T1++, c_req_T1:=0, m:=n, n++, w:=0,
snd++
m==0
out?
m:=-1, n--
seen2

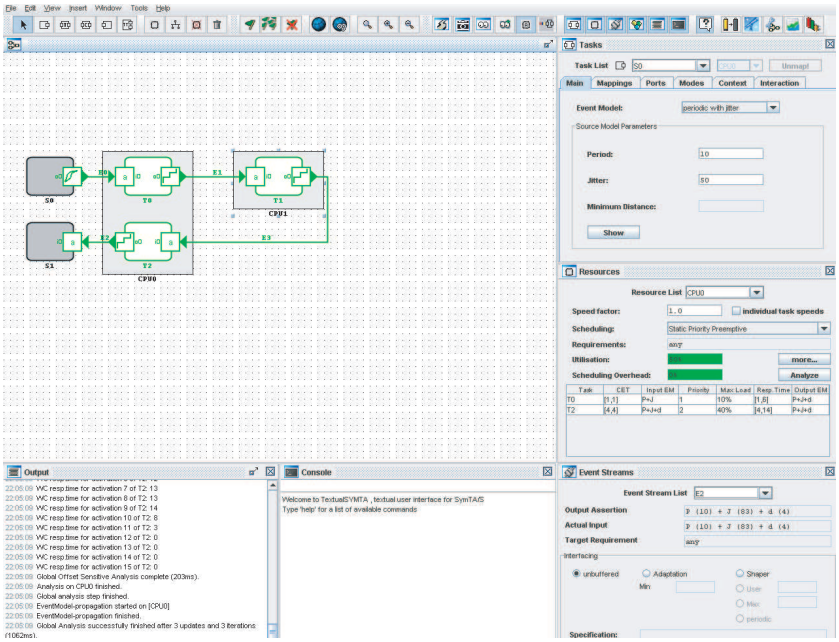

x<D
y:=0
x==D
out!
D:=0, req_T3--, c_req_T3:=0, x:=0
req_T1>0
hurry!
req_T3>0 and req_T1==0
hurry!
x:=0, D:=WCET_T3
req_T1>0
hurry!
x:=0
T3
idle
T1
pre_T1
y<=WCET_T1
x<=D
x<=WCET_T1
y==WCET_T1
req_T1--, c_req_T1:=0,
req_T2++, c_req_T2:=0,
D+=WCET_T1
x==D
out!
D:=0,
req_T3--, c_req_T3:=0
x==WCET_T1
req_T1--, c_req_T1:=0,
req_T2++, c_req_T2:=0


req_T2>0
hurry!
x:=0
T2
idle
x<=WCET_T2
x==WCET_T2
req_T2--, c_req_T2:=0, req_T3++, c_req_T3:=0
hurry?

PESIMDES

```
#include "pesimdes.h"
#define MAXBUFFER 1000
bool verbose = false;

int sc_main (int argc , char *argv[]) {

    // initialize the random number generator
   srand(-11);

   // Module instantiations

   // Event stream generator
   input_periodic_with_burst_uwj input_generator
     ("I1",sc_time(10,SC_MS),sc_time(50,SC_MS),sc_time(0,SC_MS),"input_trace_1.tra");
   /* const double change_state_probabilities[3][3] = {{0.9, 0.05, 0.05}, {0.05, 0.9, 0.05}, {0.05, 0.05, 0.9}};
      input_periodic_with_burst_configurable_distribution input_generator
        ("I1",sc_time(10,SC_MS),sc_time(50,SC_MS),sc_time(0,SC_MS),change_state_probabilities,
          'u',"input_trace_1_b.tra");    */

   // Tasks
   task t1("T1");
   task t2("T2");
   task t3("T3");

   // Resources
   resource_n_tasks_fixed_priority_preemptive cpu_1("CPU1",2);
   resource_n_tasks_fixed_priority_preemptive cpu_2("CPU2",1);

   // Event sinks
   output_display display("O1");

   // Task mapping
   cpu_1.assign_task(t1,0,sc_time(1,SC_MS),1);
   cpu_2.assign_task(t2,0,sc_time(4,SC_MS),1);
   cpu_1.assign_task(t3,0,sc_time(4,SC_MS),2);

   // Channel instantiations

   // Task activation buffers
   my_sc_fifo<event_token> buffer_T1(MAXBUFFER);
   my_sc_fifo<event_token> buffer_T2(MAXBUFFER);
   my_sc_fifo<event_token> buffer_T3(MAXBUFFER);

   // Dummy buffer for event sink
   sc_fifo<event_token> dummy_buffer_display(1);

   // Port binding

   input_generator.out(buffer_T1);
   t1.in[0](buffer_T1);
   t1.out[0](buffer_T2);
   t2.in[0](buffer_T2);
   t2.out[0](buffer_T3);
   t3.in[0](buffer_T3);
   t3.out[0](dummy_buffer_display);
   display.in(dummy_buffer_display);

   // Start simulation
   sc_start(100000,SC_MS);

   // Print performance results
   cout << "\n Max. observed End-to-End Delay: "
        << sc_time_output(display.getmaxlatency("I1")) <<
        " (event no. " << display.getseqnoofmaxlatency("I1") << ")\n";

   return 0;
}
```

SymTA/S



XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<performance_analysis xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="perfest.xsd">

<system>

  <resources>
       <FP name="CPU1" preemptive="yes"/>
       <FP name="CPU2" preemptive="yes"/>
  </resources>

  <event_sources>
       <PJD name="I1">
          <period value="10" unit="ms" />
          <jitter value="50" unit="ms" />
          <min_interarr value="0" unit="ms" />
       </PJD>
  </event_sources>

  <event_sinks>
       <event_sink name="O1" />
  </event_sinks>

  <tasks>
       <task name="T1" />
       <task name="T2" />
       <task name="T3" />
  </tasks>

  <task_graphs>
       <task_graph>
           <link src="I1" dest="T1" />
           <link src="T1" dest="T2" />
           <link src="T2" dest="T3" />
           <link src="T3" dest="O1" />
       </task_graph>
  </task_graphs>

  <binding>
       <map task="T1" resource="CPU1">
            <wcet value="1" unit="ms" />
            <priority> 1 </priority>
       </map>
       <map task="T2" resource="CPU2">
            <wcet value="4" unit="ms" />
            <priority> 1 </priority>
       </map>
       <map task="T3" resource="CPU1">
           <wcet value="4" unit="ms" />
            <priority> 2 </priority>
       </map>
  </binding>

</system>

<observe>
   <latency src="I1" dest="O1" />
</observe>

</performance_analysis>
```

C.3 Models case study 3: Variable Feedback

MPA-RTC

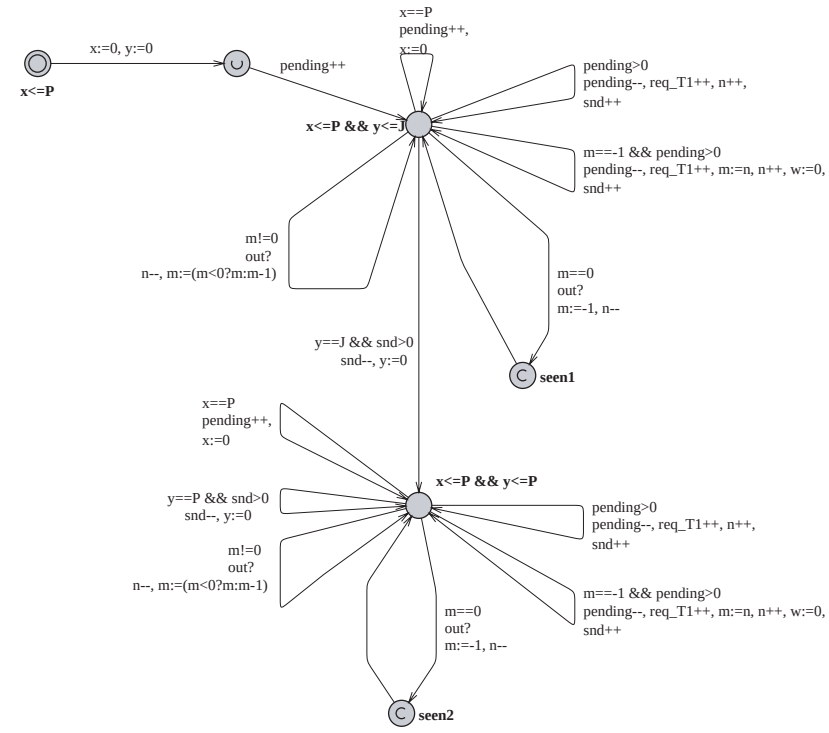
```
% Author(s): E. Wandeler
% Copyright 2004-2006 Computer Engineering and Networks Laboratory (TIK)
% ETH Zurich, Switzerland.
% All rights reserved.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Setup of system parameters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Event stream
% ========================================================================
% Stream 0:
p_S0 = 100; j_S0 = 200; d_S0 = 0;

% Stream 1:
p_S1 = 4;

% Processor speeds
% ========================================================================
% CPU0
b_CPU0 = 1;

% CPU1
b_CPU1 = 1;

% Task execution demands
% ========================================================================
ed_T0 = 20; ed_T1 = 20; ed_T2 = 15; ed_T3 = 3;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Construct input curves
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
tic;

% Construct the input arrival curves
% ========================================================================
auo0_S0 = pjdu(p_S0,j_S0,d_S0);
alo0_S0 = pjdl(p_S0,j_S0,d_S0);
auo0_S1 = pjdu(p_S1);
alo0_S1 = pjdl(p_S1);

% Construct the service curves
% ========================================================================
buo0_CPU0 = fs(b_CPU0);
blo0_CPU0 = fs(b_CPU0);
buo0_CPU1 = fs(b_CPU1);
blo0_CPU1 = fs(b_CPU1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Analysis.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Compute the arrival and service curves through a Greedy Processing
% Components for Stream 1.
% ========================================================================
[auo1_S1 alo1_S1 buo1_CPU1 blo1_CPU1] = gpc(auo0_S1, alo0_S1,
buo0_CPU1, blo0_CPU1, ed_T3);

% Compute the arrival and service curves through a chain of Greedy
% Processing Components for Stream 0.
% ========================================================================
[auo1_S0 alo1_S0 buo1_CPU0 blo1_CPU0] = gpc(auo0_S0, alo0_S0,
                                            buo0_CPU0, blo0_CPU0, ed_T0);
[auo2_S0 alo2_S0 buo2_CPU1 blo2_CPU1] = gpc(auo1_S0, alo1_S0, buo1_CPU1,
                                            blo1_CPU1, ed_T1);
[auo3_S0 alo3_S0 buo2_CPU0 blo2_CPU0] = gpc(auo2_S0, alo2_S0, buo1_CPU0,
                                            blo1_CPU0, ed_T2);

% Compute the total delay through service curve convolution and by addition.
% ========================================================================
d_conv = del(auo0_S0,blo0_CPU0,ed_T0,blo1_CPU1,ed_T1,blo1_CPU0,ed_T2);
d_add = del(auo0_S0,blo0_CPU0,ed_T0) ...
      + del(auo1_S0,blo1_CPU1,ed_T1) ...
      + del(auo2_S0,blo1_CPU0,ed_T2);
d = min(d_conv,d_add);

% Compute the backlog buffer size at T2.
% ========================================================================
b = buf(auo2_S0,blo1_CPU0,ed_T2);

tTot = toc;

% Display the results.
% ========================================================================
```

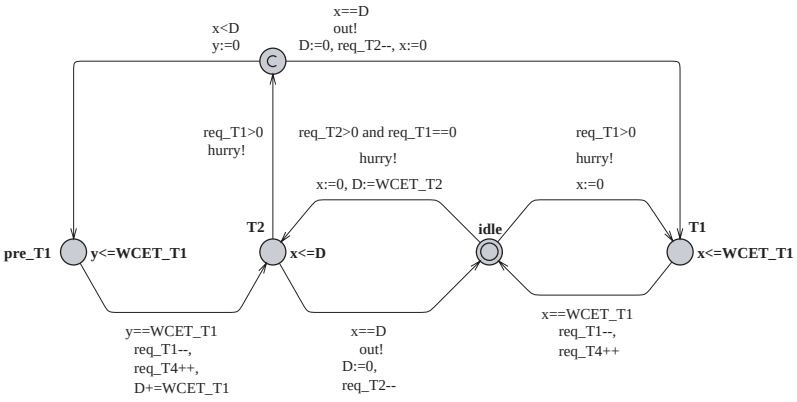
```
disp(['End-To-End Delay of Stream 0 : ' num2str(d)]) disp(['Backlog
Buffer at T2         : ' num2str(b)]) disp(['Analysis Time : '
num2str(tTot) 's'])
```

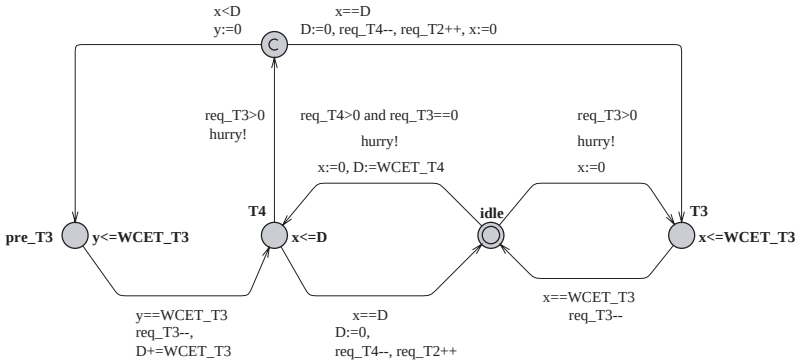
MAST

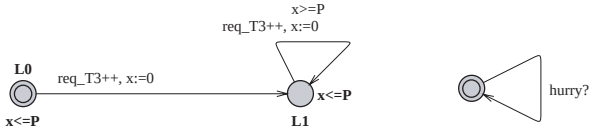
```
Model (
   Model_Name  => p5,
   Model_Date  => 2006-01-18);

Processing_Resource (
    Type            => Regular_Processor,
    Name            => CPU1);

Processing_Resource (
    Type            => Regular_Processor,
    Name            => CPU2);

Scheduler (
   Type            => Primary_Scheduler,
   Name            => CPU1,
   Host            => CPU1,
   Policy          =>
      ( Type    => Fixed_Priority));

Scheduler (
   Type            => Primary_Scheduler,
   Name            => CPU2,
   Host            => CPU2,
   Policy          =>
      ( Type    => Fixed_Priority));

Scheduling_Server (
    Type                => Regular,
    Name                => T1,
    Server_Sched_Parameters     => (
        Type        => Fixed_Priority_policy,
        The_Priority        => 2,
        Preassigned         => Yes),
    Scheduler            => CPU1);

Scheduling_Server (
    Type                => Regular,
    Name                => T2,
    Server_Sched_Parameters     => (
        Type        => Fixed_Priority_policy,
        The_Priority        => 1,
        Preassigned         => Yes),
    Scheduler            => CPU1);

Scheduling_Server (
    Type                => Regular,
    Name                => T3,
    Server_Sched_Parameters     => (
        Type        => Fixed_Priority_policy,
        The_Priority        => 2,
        Preassigned         => Yes),
    Scheduler           => CPU2);

Scheduling_Server (
    Type                => Regular,
    Name                => T4,
    Server_Sched_Parameters     => (
        Type        => Fixed_Priority_policy,
        The_Priority        => 1,
        Preassigned         => Yes),
    Scheduler           => CPU2);

Operation (
    Type                      => Simple,
    Name                      => T1,
    Best_Case_Execution_Time => 20,
    Worst_Case_Execution_Time => 20);

Operation (
    Type                      => Simple,
    Name                      => T2,
    Best_Case_Execution_Time => 15,
    Worst_Case_Execution_Time => 15);

Operation (
    Type                      => Simple,
    Name                      => T3,
    Best_Case_Execution_Time => 3,
    Worst_Case_Execution_Time => 3);

Operation (
    Type                      => Simple,
    Name                      => T4,
    Best_Case_Execution_Time => 20,
    Worst_Case_Execution_Time => 20);

Transaction (
    Type => Regular,
    Name => transact1,
    External_Events => (
        (Type   => Periodic,
         Name   => in1,
         Period => 30)),
    Internal_Events => (
        (Type   => regular,
         name   => T3_out,
                Timing_Requirements => (
            Type         => Hard_Global_Deadline,
            Deadline     => 1000,
            Referenced_Event => in1))),
    Event_Handlers => (
        (Type           => Activity,
         Input_Event        => in1,
         Output_Event       => T3_out,
         Activity_Operation => T3,
         Activity_Server    => T3)));

Transaction (
    Type => Regular,
    Name => transact2,
    External_Events => (
        (Type   => Periodic,
         Name   => in2,
         Period => 100,
                Max_Jitter => 200)),
    Internal_Events => (
        (Type   => regular,
         name   => T1_out),
        (Type   => regular,
         name   => T4_out),
        (Type   => regular,
         name   => T2_out,
                Timing_Requirements => (
            Type         => Hard_Global_Deadline,
            Deadline     => 10000,
            Referenced_Event => in2))),
    Event_Handlers => (
        (Type           => Activity,
         Input_Event        => in2,
         Output_Event       => T1_out,
         Activity_Operation => T1,
         Activity_Server    => T1),
        (Type           => Activity,
         Input_Event        => T1_out,
         Output_Event       => T4_out,
         Activity_Operation => T4,
         Activity_Server    => T4),
        (Type           => Activity,
         Input_Event        => T4_out,
         Output_Event       => T2_out,
         Activity_Operation => T2,
         Activity_Server    => T2)));
```

Uppaal


x:=0, y:=0
x<=P
pending++
x==P
pending++,
x:=0
x<=P && y<=J
pending>0
pending--, req_T1++, n++,
snd++
m==-1 && pending>0
pending--, req_T1++, m:=n, n++, w:=0,
snd++
m!=0
out?
n--, m:=(m<0?m:m-1)
m==0
out?
m:=-1, n--
y==J && snd>0
snd--, y:=0
seen1
x==P
pending++,
x:=0
x<=P && y<=P
y==P && snd>0
snd--, y:=0
pending>0
pending--, req_T1++, n++,
snd++
m!=0
out?
n--, m:=(m<0?m:m-1)
m==-1 && pending>0
pending--, req_T1++, m:=n, n++, w:=0,
snd++
m==0
out?
m:=-1, n--
seen2


x<D
y:=0
x==D
out!
D:=0, req_T2--, x:=0
req_T1>0
hurry!
req_T2>0 and req_T1==0
hurry!
x:=0, D:=WCET_T2
req_T1>0
hurry!
x:=0
pre_T1
y<=WCET_T1
T2
x<=D
idle
T1
x<=WCET_T1
y==WCET_T1
req_T1--,
req_T4++,
D+=WCET_T1
x==D
out!
D:=0,
req_T2--
x==WCET_T1
req_T1--,
req_T4++

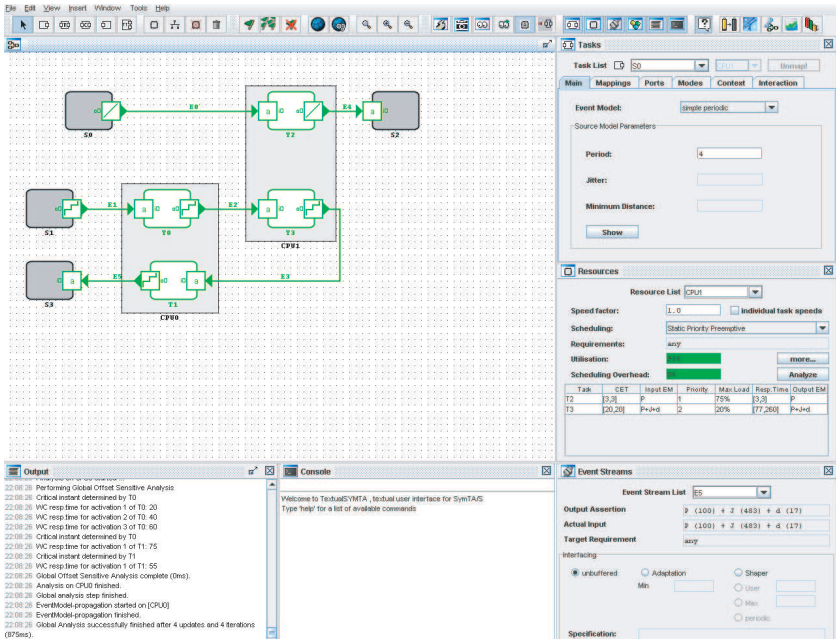

x<D
y:=0
x==D
D:=0, req_T4--, req_T2++, x:=0
req_T3>0
hurry!
req_T4>0 and req_T3==0
hurry!
x:=0, D:=WCET_T4
req_T3>0
hurry!
x:=0
pre_T3
y<=WCET_T3
T4
x<=D
idle
T3
x<=WCET_T3
y==WCET_T3
req_T3--,
D+=WCET_T3
x==D
D:=0,
req_T4--, req_T2++
x==WCET_T3
req_T3--


x>=P
req_T3++, x:=0
L0
x<=P
req_T3++, x:=0
x<=P
L1
hurry?

PESIMDES

```
#include "pesimdes.h"
#define MAXBUFFER 1000
bool verbose = false;

int sc_main (int argc , char *argv[]) {

    // initialize the random number generator
   srand(-11);

   // Module instantiations

   // Event stream generators
   input_periodic input_generator_1("I1",sc_time(4,SC_MS),"input_trace_1.tra");
   input_periodic_with_burst_uwj input_generator_2
     ("I2",sc_time(100,SC_MS),sc_time(200,SC_MS),sc_time(0,SC_MS),"input_trace_2.tra");
   /* const double change_state_probabilities[3][3] = {{0.9, 0.05, 0.05}, {0.05, 0.9, 0.05}, {0.05, 0.05, 0.9}};
      input_periodic_with_burst_configurable_distribution input_generator_2
        ("I2",sc_time(100,SC_MS),sc_time(200,SC_MS),sc_time(0,SC_MS),change_state_probabilities,
          'u',"input_trace_2_b.tra");    */

   // Tasks
   task t1("T1");
   task t2("T2");
   task t3("T3");
   task t4("T4");

   // Resources
   resource_n_tasks_fixed_priority_preemptive cpu_1("CPU1",2);
   resource_n_tasks_fixed_priority_preemptive cpu_2("CPU2",2);

   // Event sinks
   output_display display_1("O1");
   output_display display_2("O2");

   // Task mapping
   cpu_1.assign_task(t1,0,sc_time(20,SC_MS),1);
   cpu_1.assign_task(t2,1,sc_time(15,SC_MS),2);
   cpu_2.assign_task(t3,0,sc_time(3,SC_MS),1);
   cpu_2.assign_task(t4,1,sc_time(20,SC_MS),2);

   // Channel instantiations

   // Task activation buffers
   my_sc_fifo<event_token> buffer_T1(MAXBUFFER);
   my_sc_fifo<event_token> buffer_T2(MAXBUFFER);
   my_sc_fifo<event_token> buffer_T3(MAXBUFFER);
   my_sc_fifo<event_token> buffer_T4(MAXBUFFER);

   // Dummy buffers for event sinks
   sc_fifo<event_token> dummy_buffer_display_1(1);
   sc_fifo<event_token> dummy_buffer_display_2(1);

   // Port binding

   input_generator_1.out(buffer_T3);
   t3.in[0](buffer_T3);
   t3.out[0](dummy_buffer_display_1);
   display_1.in(dummy_buffer_display_1);

   input_generator_2.out(buffer_T1);
   t1.in[0](buffer_T1);
   t1.out[0](buffer_T4);
   t4.in[0](buffer_T4);
   t4.out[0](buffer_T2);
   t2.in[0](buffer_T2);
   t2.out[0](dummy_buffer_display_2);
   display_2.in(dummy_buffer_display_2);

   // Start simulation
   sc_start(100000,SC_MS);

   // Print performance results
   cout << "\n Max. observed End-to-End Delay I2-O2: "
        << sc_time_output(display_2.getmaxlatency("I2")) <<
        " (event no. " << display_2.getseqnoofmaxlatency("I2") << ")\n";
   cout << "Max. observed Backlog T2: " << buffer_T2.getmaxbacklog() << "\n";

   return 0;
}
```

SymTA/S



XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<performance_analysis xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="perfest.xsd">

<system>

  <resources>
       <FP name="CPU1" preemptive="yes"/>
       <FP name="CPU2" preemptive="yes"/>
  </resources>

  <event_sources>
       <PJD name="I1">
          <period value="4" unit="ms" />
       </PJD>
       <PJD name="I2">
          <period value="100" unit="ms" />
          <jitter value="200" unit="ms" />
          <min_interarr value="0" unit="ms" />
       </PJD>
  </event_sources>

  <event_sinks>
       <event_sink name="O1" />
       <event_sink name="O2" />
  </event_sinks>

  <tasks>
       <task name="T1" />
       <task name="T2" />
       <task name="T3" />
       <task name="T4" />
  </tasks>

  <task_graphs>
       <task_graph>
           <link src="I1" dest="T3" />
           <link src="T3" dest="O1" />
       </task_graph>
       <task_graph>
           <link src="I2" dest="T1" />
           <link src="T1" dest="T4" />
           <link src="T4" dest="T2" />
           <link src="T2" dest="O2" />
       </task_graph>
  </task_graphs>

  <binding>
       <map task="T1" resource="CPU1">
            <wcet value="20" unit="ms" />
            <priority> 1 </priority>
       </map>
       <map task="T2" resource="CPU1">
            <wcet value="15" unit="ms" />
            <priority> 2 </priority>
       </map>
       <map task="T3" resource="CPU2">
           <wcet value="3" unit="ms" />
            <priority> 1 </priority>
       </map>
       <map task="T4" resource="CPU2">
           <wcet value="20" unit="ms" />
            <priority> 2 </priority>
       </map>
  </binding>

</system>

<observe>
   <backlog task="T2" />
   <latency src="I2" dest="O2" />
</observe>

</performance_analysis>
```

C.4 Models case study 4: AND/OR task activation

MPA-RTC

```
% Author(s): E. Wandeler, S. Perathoner
% Copyright 2004-2006 Computer Engineering and Networks Laboratory (TIK)
% ETH Zurich, Switzerland.
% All rights reserved.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Setup of system parameters for OR-activation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Event streams
% ============================================================================
% Stream 0:
p_S0 = 100; j_S0 = 20;

% Stream 1:
p_S1 = 150; j_S1 = 60;

% Processor speeds
% ============================================================================
% CPU0
b_CPU0 = 1;

% Task execution demands
% ============================================================================
% T0
ed_T0 = 40;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Construct input curves
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

auo_S0 = pjdu(p_S0,j_S0);
alo_S0 = pjdl(p_S0,j_S0);
auo_S1 = pjdu(p_S1,j_S1);
alo_S1 = pjdl(p_S1,j_S1);
buo_CPU0 = fs(b_CPU0);
blo_CPU0 = fs(b_CPU0);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Analysis.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
tic;

% Compute the OR-activation
% ============================================================================
[auo_OR alo_OR] = or(auo_S0,alo_S0,auo_S1,alo_S1);

% Compute the output behavior of T0
% ============================================================================
[auoT0 aloT0 buoT0 bloT0] = gpc(auo_OR, alo_OR, buo_CPU0, blo_CPU0, ed_T0);

% Compute the delay and backlog on T0
% ============================================================================
d_OR = del(auo_OR,blo_CPU0,ed_T0);
b_OR = buf(auo_OR,blo_CPU0,ed_T0);


tTot = toc;

% Display the results
% ============================================================================
disp('=====================  OR-activation ==============================')
disp(['Delay     : ' num2str(d_OR)]) disp(['Backlog   :
' num2str(b_OR)]) disp(['Analysis time   : ' num2str(tTot) 's'])


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Setup of system parameters for AND-activation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Event streams
% ============================================================================
% Stream 2:
p_S2 = 100; j_S2 = 10;

% Stream 3:
p_S3 = 100; j_S3 = 190; d_S3 = 20;

% Processor speeds
% ============================================================================
% CPU1
b_CPU1 = 1;

% Task execution demands
```

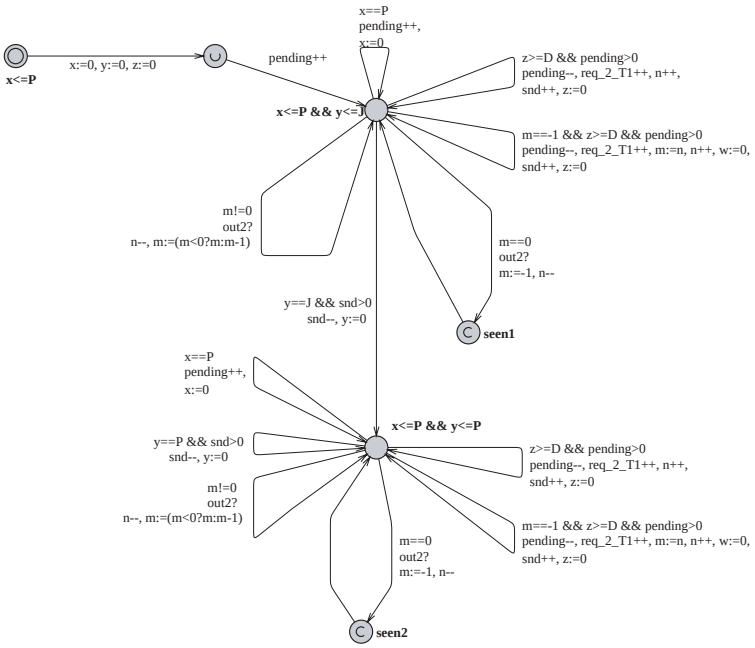
```
% ========================================================================
% T1
ed_T1 = 40;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Construct input curves
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

auo_S2 = pjdu(p_S2,j_S2);
alo_S2 = pjdl(p_S2,j_S2);
auo_S3 = pjdu(p_S3,j_S3,d_S3);
alo_S3 = pjdl(p_S3,j_S3,d_S3); buo_CPU1 = fs(b_CPU1); blo_CPU1 = fs(b_CPU1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Analysis.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
tic;

% Compute the AND-activation
% ========================================================================
[auo_AND alo_AND] = and(auo_S2,alo_S2,auo_S3,alo_S3);

% Compute the output behavior of T1
% ========================================================================
[auoT1 aloT1 buoT1 bloT1] = gpc(auo_AND, alo_AND, buo_CPU1, blo_CPU1, ed_T1);

% Compute the delay and backlog on T1
% ========================================================================
d_AND = del(auo_AND,blo_CPU1,ed_T1);
b_AND = buf(auo_AND,blo_CPU1,ed_T1);

% Consider also the AND-activation buffers
% ========================================================================
d_AND = d_AND + max(h(auo_S3,alo_S2),h(auo_S2,alo_S3));
b_AND = 2 * b_AND + max(v(auo_S3,alo_S2),v(auo_S2,alo_S3));

tTot = toc;

% Display the results
% ========================================================================
disp('====================  AND-activation ==============================')
disp(['Delay     : ' num2str(d_AND)]) disp(['Total buffersize   :
' num2str(b_AND)]) disp(['Analysis time   : ' num2str(tTot) 's'])
```

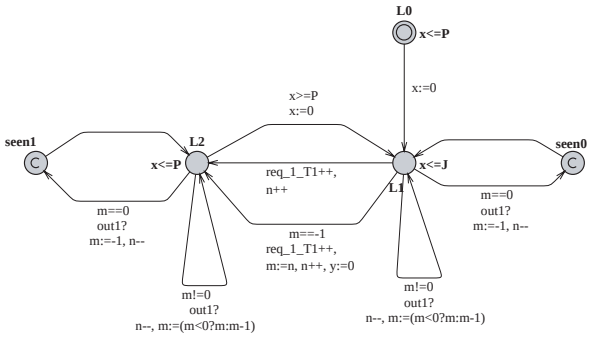
MAST (OR-activation)

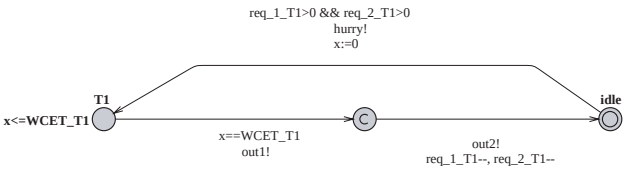
```
Model (
   Model_Name  => p3_OR,
   Model_Date  => 2006-01-16);

Processing_Resource (
    Type            => Regular_Processor,
    Name            => CPU1);

Scheduler (
   Type            => Primary_Scheduler,
   Name            => CPU1,
   Host            => CPU1,
   Policy          =>
      ( Type    => Fixed_Priority));

Scheduling_Server (
    Type               => Regular,
    Name               => T1,
    Server_Sched_Parameters     => (
        Type        => Fixed_Priority_policy,
        The_Priority        => 1,
        Preassigned         => Yes),
    Scheduler       => CPU1);

Operation (
    Type                      => Simple,
    Name                      => T1,
    Best_Case_Execution_Time => 60,
    Worst_Case_Execution_Time => 60);

Transaction (
    Type => Regular,
    Name => transact1,
    External_Events => (
        (Type   => Periodic,
         Name   => in_1,
         Period => 100,
         Max_Jitter => 20)
        ),
    Internal_Events => (
        (Type   => regular,
         name   => T1_out_1,
         Timing_Requirements => (
             Type         => Hard_Global_Deadline,
             Deadline     => 10000,
             Referenced_Event => in_1)
                )
        ),
    Event_Handlers => (
        (Type           => Activity,
         Input_Event        => in_1,
         Output_Event       => T1_out_1,
         Activity_Operation => T1,
         Activity_Server    => T1
                )
        )
);

Transaction (
    Type => Regular,
    Name => transact2,
    External_Events => (
        (Type   => Periodic,
         Name   => in_2,
         Period => 150,
                 Max_Jitter => 60)
        ),
    Internal_Events => (
        (Type   => regular,
         name   => T1_out_2,
         Timing_Requirements => (
             Type         => Hard_Global_Deadline,
             Deadline     => 10000,
```

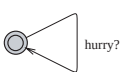
```
        Referenced_Event => in_2)
           )
    ),
Event_Handlers => (
    (Type            => Activity,
     Input_Event        => in_2,
     Output_Event       => T1_out_2,
                Activity_Operation => T1,
                Activity_Server    => T1
                    )
                )
        );
```

Uppaal (AND-activation)

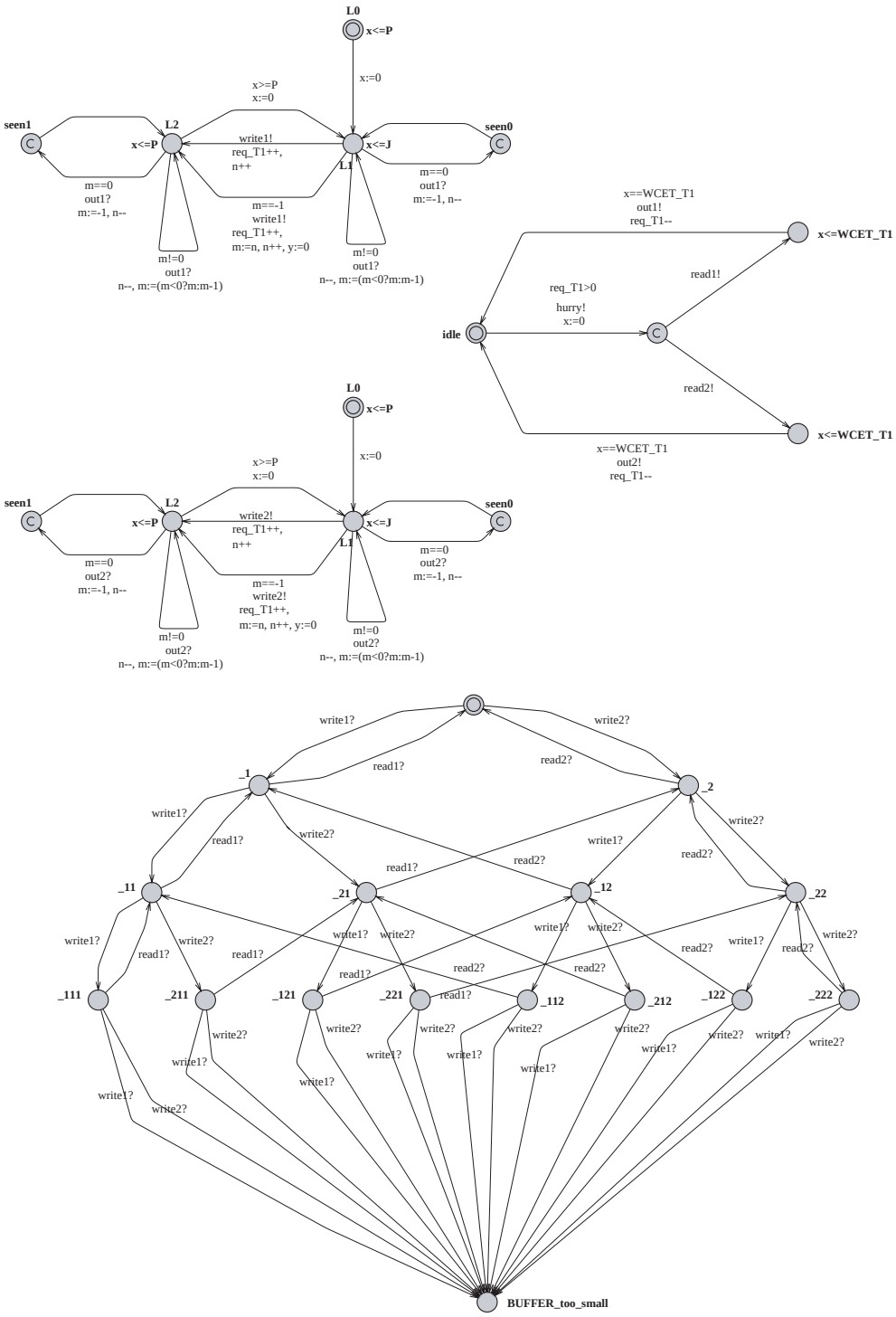








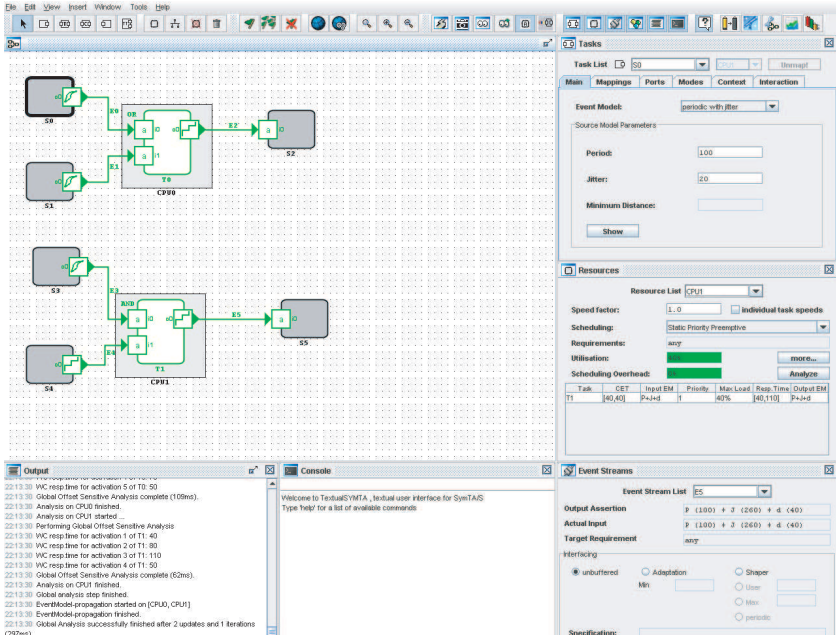
Uppaal (OR-activation)



PESIMDES

```
#include "pesimdes.h"
#define MAXBUFFER 1000
bool verbose = false;

int sc_main (int argc , char *argv[]) {

    // initialize the random number generator
   srand(-11);

   // Module instantiations

   // Event stream generators
   const double change_state_probabilities[3][3] = {{0.9, 0.05, 0.05}, {0.05, 0.9, 0.05}, {0.05, 0.05, 0.9}};
   input_periodic_with_jitter_configurable_distribution input_generator_1
        ("I1",sc_time(100,SC_MS),sc_time(20,SC_MS),change_state_probabilities,'u',"input_trace_1.tra");
   input_periodic_with_jitter_configurable_distribution input_generator_2
        ("I2",sc_time(150,SC_MS),sc_time(60,SC_MS),change_state_probabilities,'u',"input_trace_2.tra");
   input_periodic_with_jitter_configurable_distribution input_generator_3
        ("I3",sc_time(100,SC_MS),sc_time(10,SC_MS),change_state_probabilities,'u',"input_trace_3.tra");
   input_periodic_with_burst_configurable_distribution input_generator_4
        ("I4",sc_time(100,SC_MS),sc_time(190,SC_MS),sc_time(20,SC_MS),change_state_probabilities,
          'u',"input_trace_4.tra");

   // Tasks
   task t1("T1",2,"OR",1);
   task t2("T2",2,"AND",1);

   // Resources
   resource_n_tasks_fixed_priority_preemptive cpu_1("CPU1",1);
   resource_n_tasks_fixed_priority_preemptive cpu_2("CPU2",1);

   // Event sinks
   output_display display_1("O1");
   output_display display_2("O2");

   // Task mapping
   cpu_1.assign_task(t1,0,sc_time(40,SC_MS),1);
   cpu_2.assign_task(t2,0,sc_time(40,SC_MS),1);

   // Channel instantiations

   // Task activation buffers
   my_sc_fifo<event_token> buffer_T1_1(MAXBUFFER);
   my_sc_fifo<event_token> buffer_T1_2(MAXBUFFER);
   my_sc_fifo<event_token> buffer_T2_1(MAXBUFFER);
   my_sc_fifo<event_token> buffer_T2_2(MAXBUFFER);

   // Dummy buffers for event sinks
   sc_fifo<event_token> dummy_buffer_display_1(1);
   sc_fifo<event_token> dummy_buffer_display_2(1);

   // Port binding

   input_generator_1.out(buffer_T1_1);
   t1.in[0](buffer_T1_1);
   input_generator_2.out(buffer_T1_2);
   t1.in[1](buffer_T1_2);
   t1.out[0](dummy_buffer_display_1);
   display_1.in(dummy_buffer_display_1);

   input_generator_3.out(buffer_T2_1);
   t2.in[0](buffer_T2_1);
   input_generator_4.out(buffer_T2_2);
   t2.in[1](buffer_T2_2);
   t2.out[0](dummy_buffer_display_2);
   display_2.in(dummy_buffer_display_2);

   // Start simulation
   sc_start(100000,SC_MS);

   // Print performance results
   cout << "\n Max. observed End-to-End Delay I1-O1: "
        << sc_time_output(display_1.getmaxlatency("I1")) <<
        " (event no. " << display_1.getseqn1oofmaxlatency("I1") << ")\n";
   cout << "Max. observed Backlog T1_1: " << buffer_T1_1.getmaxbacklog() << "\n";
   cout << "Max. observed Backlog T1_2: " << buffer_T1_2.getmaxbacklog() << "\n";
   cout << "\n Max. observed End-to-End Delay I3-O2: "
        << sc_time_output(display_2.getmaxlatency("I3")) <<
        " (event no. " << display_2.getseqn1oofmaxlatency("I3") << ")\n";
   cout << "\n Max. observed End-to-End Delay I4-O2: "
        << sc_time_output(display_2.getmaxlatency("I4")) <<
        " (event no. " << display_2.getseqn1oofmaxlatency("I4") << ")\n";
   cout << "Max. observed Backlog T2_1: " << buffer_T2_1.getmaxbacklog() << "\n";
   cout << "Max. observed Backlog T2_2: " << buffer_T2_2.getmaxbacklog() << "\n";

   return 0;
}
```

SymTA/S



XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<performance_analysis xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="perfest.xsd">

<system>

  <resources>
       <FP name="CPU1" preemptive="no"/>
       <FP name="CPU2" preemptive="no"/>
  </resources>

  <event_sources>
       <PJD name="I1">
          <period value="100" unit="ms" />
          <jitter value="20" unit="ms" />
       </PJD>
       <PJD name="I2">
          <period value="150" unit="ms" />
          <jitter value="60" unit="ms" />
       </PJD>
       <PJD name="I3">
          <period value="100" unit="ms" />
          <jitter value="10" unit="ms" />
       </PJD>
       <PJD name="I4">
          <period value="100" unit="ms" />
          <jitter value="190" unit="ms" />
          <min_interarr value="20" unit="ms" />
       </PJD>
  </event_sources>

  <event_sinks>
       <event_sink name="O1" />
       <event_sink name="O2" />
  </event_sinks>

  <tasks>
       <task name="T1" input_no="2"
         activation_type="OR" />
       <task name="T2" input_no="2"
         activation_type="AND" />
  </tasks>

  <task_graphs>
       <task_graph>
          <link src="I1"
            dest="T1" dest_index="0" />
          <link src="I2" dest="T1"
            dest_index="1" />
          <link src="T1" dest="O1" />
       </task_graph>
       <task_graph>
          <link src="I3"
            dest="T2" dest_index="0" />
          <link src="I4"
            dest="T2" dest_index="1" />
          <link src="T2" dest="O2" />
       </task_graph>
  </task_graphs>

  <binding>
       <map task="T1" resource="CPU1">
            <wcet value="40" unit="ms" />
            <priority> 1 </priority>
       </map>
       <map task="T2" resource="CPU2">
            <wcet value="40" unit="ms" />
            <priority> 1 </priority>
       </map>
  </binding>

</system>

<observe>
   <latency src="I1" dest="O1" />
   <backlog task="T1" index="0" />
```

```
<backlog task="T1" index="1" />
<latency src="I3" dest="O2" />
<latency src="I4" dest="O2" />
<backlog task="T2" index="0" />
<backlog task="T2" index="1" />
```

```
</observe>

</performance_analysis>
```

C.5 Models case study 5: Intra-context information

MPA-RTC

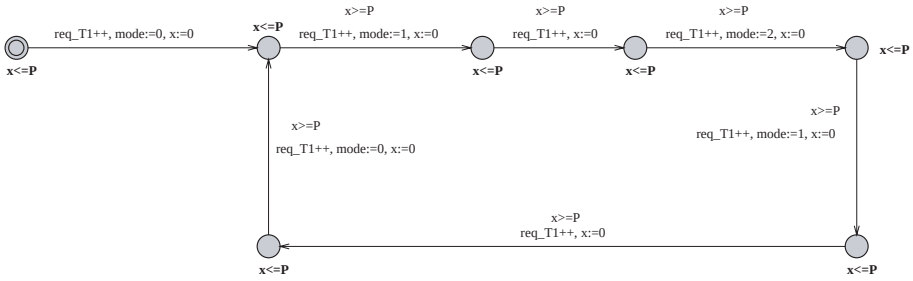
```
% Author(s): E. Wandeler
% Copyright 2004-2006 Computer Engineering and Networks Laboratory (TIK)
% ETH Zurich, Switzerland.
% All rights reserved.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Setup of system parameters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Event streams
% ============================================================================
% Stream S0:
% Pattern: BBPBBI => worst-case pattern is IBBPBB,
p_S0 = 200;

% Stream S1:
p_S1 = 1000;

% Processor speeds
% ============================================================================
cpu0 = 1;

% Task execution demands
% ============================================================================
ed_T0_I = 80;
ed_T0_P = 56;
ed_T0_B = 44;
ed_T1   = 200;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Construct input curves
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
tic;

% Construct the input arrival curves
% ============================================================================
% This curve is a resource-based arrival curve, that can be obtained by
% transformation of an event-based arrival curve with a workload curve.
aui_S0 = curve([[0 ed_T0_I 0];...
                [p_S0 ed_T0_I+ed_T0_B 0];...
                [2*p_S0 ed_T0_I+ed_T0_B+ed_T0_B 0];...
                [3*p_S0 ed_T0_I+ed_T0_B+ed_T0_B+ed_T0_P 0];...
                [4*p_S0 ed_T0_I+ed_T0_B+ed_T0_B+ed_T0_P+ed_T0_B 0];...
                [5*p_S0 ed_T0_I+ed_T0_B+ed_T0_B+ed_T0_P+ed_T0_B+ed_T0_B 0]], ...
                0, 6*p_S0, ed_T0_I+ed_T0_B+ed_T0_B+ed_T0_P+ed_T0_B+ed_T0_B);

aui_S1 = pjdu(p_S1);

% Construct the service curve
% ============================================================================
bli_CPU0 = fs(cpu0);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Analysis.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% To answer the design question, we do not need to analyze full GPCs.

% Compute the lower servie curve after processing T0
% ============================================================================
blo1_CPU0 = maxconv((bli_CPU0 - aui_S0), 0);

% Compute the delay of T1
% ============================================================================
d_T1 = del(aui_S1,blo1_CPU0,ed_T1);

tTot = toc;

% Display the results
% ============================================================================
disp(['Delay of T1   : ' num2str(d_T1)])
disp(['Analysis time : ' num2str(tTot) 's'])
disp(' ')
```

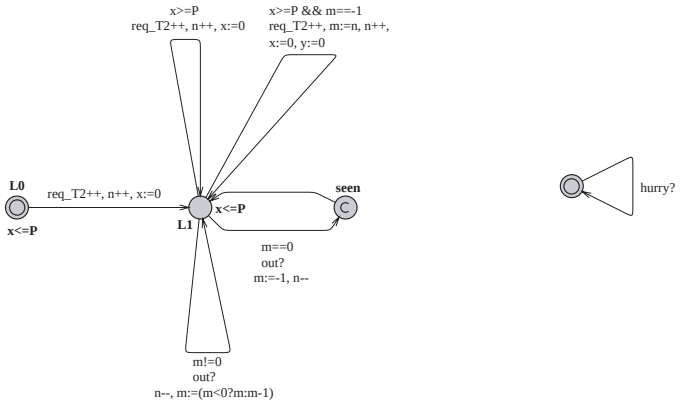
MAST (context-blind)

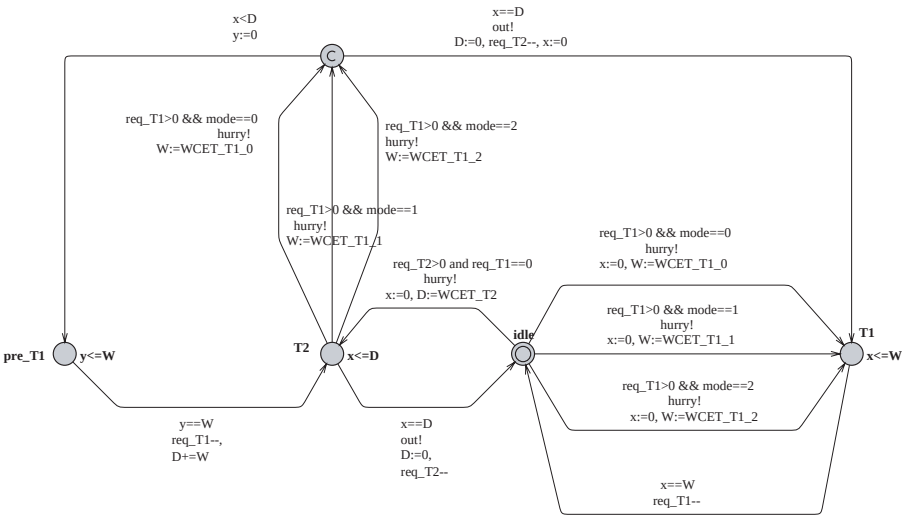
```
Model (
   Model_Name  => Intra-context,
   Model_Date  => 2006-01-19);

Processing_Resource (
    Type            => Regular_Processor,
    Name            => CPU1);

Scheduler (
   Type            => Primary_Scheduler,
   Name            => CPU1,
   Host            => CPU1,
   Policy          =>
      ( Type    => Fixed_Priority));

Scheduling_Server (
    Type               => Regular,
    Name               => T1,
    Server_Sched_Parameters     => (
        Type        => Fixed_Priority_policy,
        The_Priority        => 2,
        Preassigned         => Yes),
    Scheduler          => CPU1);

Scheduling_Server (
    Type               => Regular,
    Name               => T2,
    Server_Sched_Parameters     => (
        Type        => Fixed_Priority_policy,
        The_Priority        => 1,
        Preassigned         => Yes),
    Scheduler          => CPU1);

Operation (
    Type                      => Simple,
    Name                      => T1,
    Best_Case_Execution_Time => 5,
    Worst_Case_Execution_Time => 80);

Operation (
    Type                      => Simple,
    Name                      => T2,
    Best_Case_Execution_Time => 200,
    Worst_Case_Execution_Time => 200);
```

```
Transaction (
    Type => Regular,
    Name => transact1,
    External_Events => (
        (Type   => Periodic,
         Name   => in,
         Period => 200)),
    Internal_Events => (
        (Type   => regular,
         name   => T1_out,
                 Timing_Requirements => (
            Type         => Hard_Global_Deadline,
            Deadline     => 10000,
            Referenced_Event => in))),
    Event_Handlers => (
        (Type           => Activity,
         Input_Event        => in,
         Output_Event       => T1_out,
         Activity_Operation => T1,
         Activity_Server    => T1)));

Transaction (
    Type => Regular,
    Name => transact2,
    External_Events => (
        (Type   => Periodic,
         Name   => in2,
         Period => 1000)),
    Internal_Events => (
        (Type   => regular,
         name   => T2_out,
                 Timing_Requirements => (
            Type         => Hard_Global_Deadline,
            Deadline     => 50000,
            Referenced_Event => in2))),
    Event_Handlers => (
        (Type           => Activity,
         Input_Event        => in2,
         Output_Event       => T2_out,
         Activity_Operation => T2,
         Activity_Server    => T2)));
```

Uppaal


req_T1++, mode:=0, x:=0
x<=P
x>=P
req_T1++, mode:=1, x:=0
x>=P
req_T1++, x:=0
x>=P
req_T1++, mode:=2, x:=0
x<=P
x>=P
req_T1++, mode:=0, x:=0
x>=P
req_T1++, mode:=1, x:=0
x>=P
req_T1++, x:=0

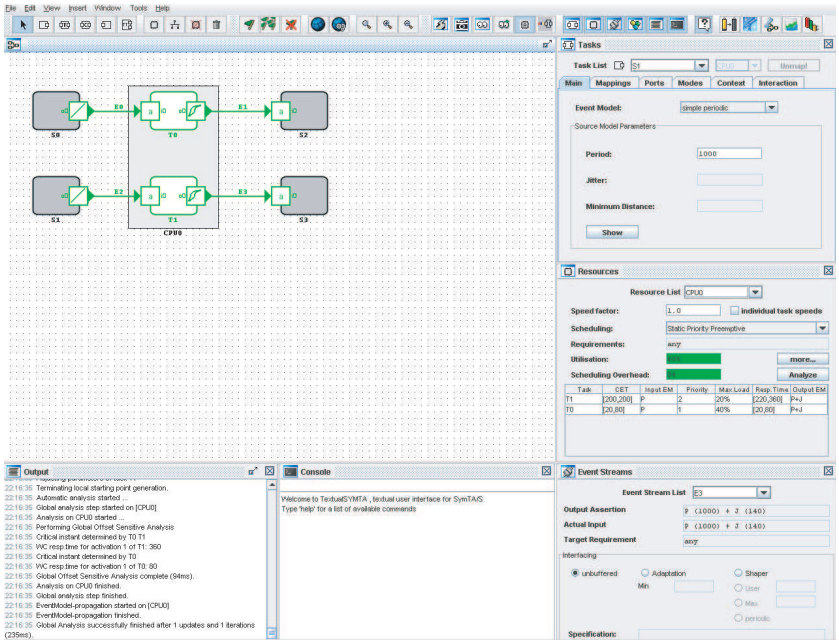

L0
x<=P
req_T2++, n++, x:=0
L1
x<=P
x>=P
req_T2++, n++, x:=0
x>=P && m==-1
req_T2++, m:=n, n++,
x:=0, y:=0
seen
m==0
out?
m:=-1, n--
m!=0
out?
n--, m:=(m<0?m:m-1)
hurry?


x<D
y:=0
x==D
out!
D:=0, req_T2--, x:=0
req_T1>0 && mode==0
hurry!
W:=WCET_T1_0
req_T1>0 && mode==2
hurry!
W:=WCET_T1_2
req_T1>0 && mode==1
hurry!
W:=WCET_T1_1
req_T1>0 && mode==0
hurry!
x:=0, W:=WCET_T1_0
req_T2>0 and req_T1==0
hurry!
x:=0, D:=WCET_T2
req_T1>0 && mode==1
hurry!
x:=0, W:=WCET_T1_1
req_T1>0 && mode==2
hurry!
x:=0, W:=WCET_T1_2
pre_T1
y<=W
T2
x<=D
idle
T1
x<=W
y==W
req_T1--,
D+=W
x==D
out!
D:=0,
req_T2--
x==W
req_T1--

PESIMDES (context-blind)

```
#include "pesimdes.h"
#define MAXBUFFER 1000
bool verbose = false;

int sc_main (int argc , char *argv[]) {

    // initialize the random number generator
   srand(-11);

   // Module instantiations

   // Event stream generators
   input_periodic input_generator_1("I1",sc_time(200,SC_MS),"input_trace_1.tra");
   input_periodic input_generator_2("I2",sc_time(1000,SC_MS),"input_trace_2.tra");

   // Tasks
   task t1("T1");
   task t2("T2");

   // Resources
   resource_n_tasks_fixed_priority_preemptive cpu("CPU",2);

   // Event sinks
   output_display display_1("O1");
   output_display display_2("O2");

   // Task mapping
   cpu.assign_task(t1,0,sc_time(80,SC_MS),1);
   cpu.assign_task(t2,1,sc_time(200,SC_MS),2);

   // Channel instantiations

   // Task activation buffers
   my_sc_fifo<event_token> buffer_T1(MAXBUFFER);
   my_sc_fifo<event_token> buffer_T2(MAXBUFFER);

   // Dummy buffers for event sinks
   sc_fifo<event_token> dummy_buffer_display_1(1);
   sc_fifo<event_token> dummy_buffer_display_2(1);

   // Port binding

   input_generator_1.out(buffer_T1);
   t1.in[0](buffer_T1);
   t1.out[0](dummy_buffer_display_1);
   display_1.in(dummy_buffer_display_1);

   input_generator_2.out(buffer_T2);
   t2.in[0](buffer_T2);
   t2.out[0](dummy_buffer_display_2);
   display_2.in(dummy_buffer_display_2);


   // Start simulation
   sc_start(100000,SC_MS);

   // Print performance results
   cout << "\n Max. observed End-to-End Delay I2-O2: "
        << sc_time_output(display_2.getmaxlatency("I2")) <<
        " (event no. " << display_2.getseqn1oofmaxlatency("I2") << ")\n";

   return 0;
}
```

SymTA/S



XML (context-blind)

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<performance_analysis xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="perfest.xsd">

<system>

  <resources>
       <FP name="CPU1" preemptive="no"/>
  </resources>

  <event_sources>
       <PJD name="I1">
          <period value="200" unit="ms" />
       </PJD>
       <PJD name="I2">
          <period value="1000" unit="ms" />
       </PJD>
  </event_sources>

  <event_sinks>
       <event_sink name="O1" />
       <event_sink name="O2" />
  </event_sinks>

  <tasks>
       <task name="T1" />
       <task name="T2" />
  </tasks>

  <task_graphs>
       <task_graph>
           <link src="I1" dest="T1" />
           <link src="T1" dest="O1" />
       </task_graph>
       <task_graph>
           <link src="I2" dest="T2" />
           <link src="T2" dest="O2" />
       </task_graph>
  </task_graphs>

  <binding>
       <map task="T1" resource="CPU">
            <wcet value="80" unit="ms" />
            <priority> 1 </priority>
       </map>
       <map task="T2" resource="CPU">
            <wcet value="200" unit="ms" />
            <priority> 2 </priority>
       </map>
  </binding>

</system>

<observe>
   <latency src="I2" dest="O2" />
</observe>

</performance_analysis>
```

C.6 Models case study 6: Workload correlations

MPA-RTC

```
% Author(s): E. Wandeler
% Copyright 2004-2006 Computer Engineering and Networks Laboratory (TIK)
% ETH Zurich, Switzerland.
% All rights reserved.

auS1 = pjdu(4,15,1);
alS1 = pjdl(4,15,1);
auS2 = pjdu(6,1);
alS2 = pjdl(6,1);
buCPU1 = fs(6);
blCPU1 = fs(6);
buCPU2 = fs(6);
blCPU2 = fs(6);

[auoT1e,aloT1e,bu,bl] = puregpc(auS1,alS1,buCPU1./5,blCPU1./20,1);
[auoT1r,aloT1r,bu,bl] = puregpc(auS1.*20,alS1.*5,buCPU1,blCPU1,1);

auiT2WCC = ceil(auoT1r./5).*5+10;
aliT2WCC = floor(aloT1r./20).*20;
auiT2WLT = ceil(auoT1e).*15;
aliT2WLT = floor(aloT1e).*5;

[au,al,buoCPU2WCC,bloCPU2WCC] = puregpc(min(auiT2WCC,auiT2WLT),max(aliT2WCC,aliT2WLT),buCPU2,blCPU2,1);
[au,al,buoCPU2WLT,bloCPU2WLT] = puregpc(auiT2WLT,aliT2WLT,buCPU2,blCPU2,1);

dWLT = h(auS2.*5,bloCPU2WLT);
dWCC = h(auS2.*5,bloCPU2WCC);

disp(['d_WLT=' num2str(dWLT) ', d_WCC=' num2str(dWCC)])
```

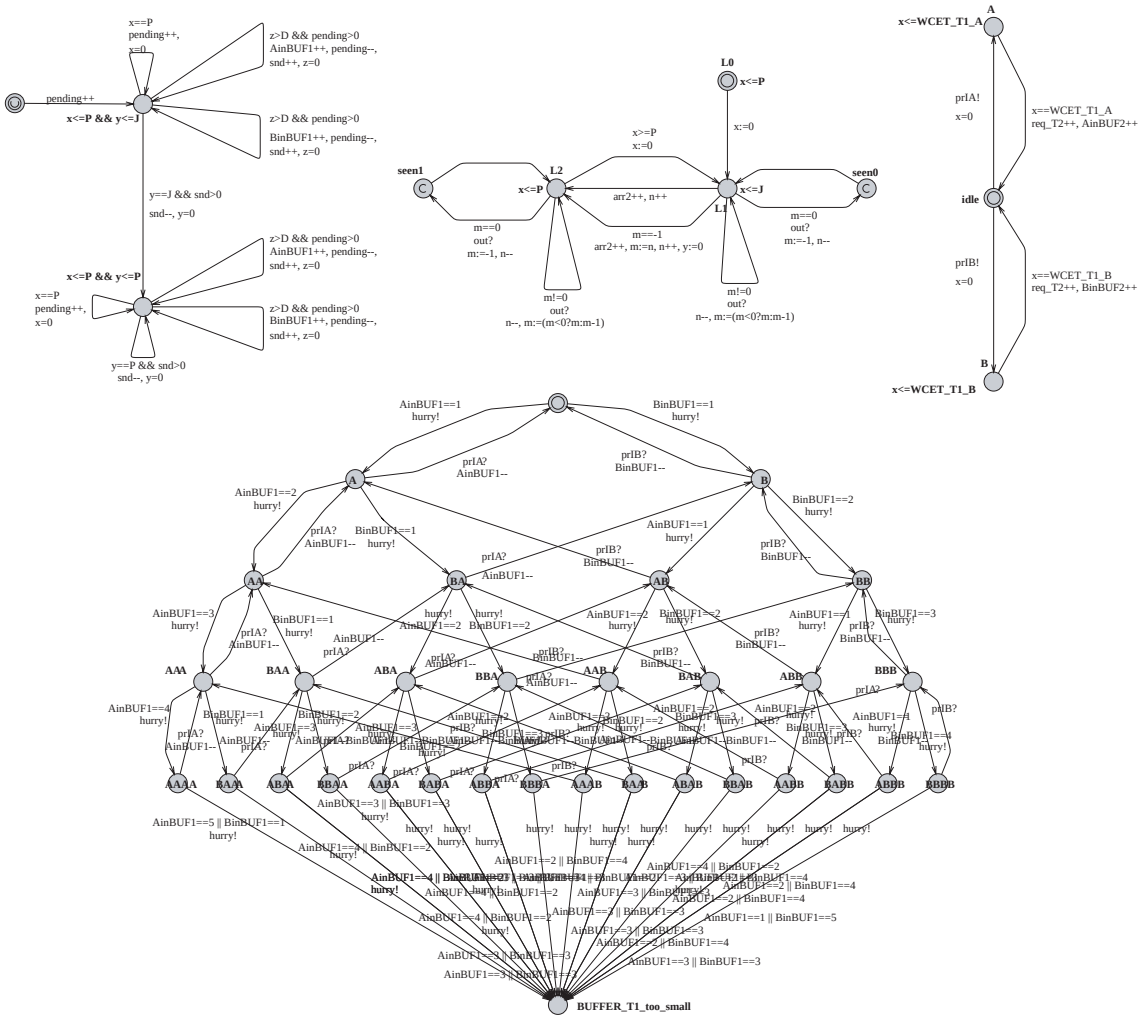
MAST (workload corr. blind)

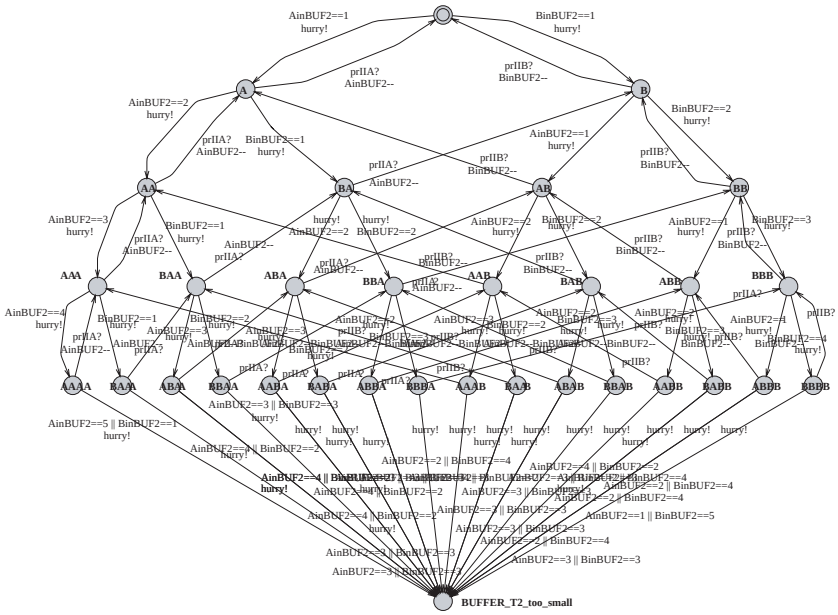
```
Model (
   Model_Name  => workload_correlation,
   Model_Date  => 2005-12-13);

Processing_Resource (
    Type            => Regular_Processor,
    Name            => CPU1,
    Speed_Factor    => 6);

Processing_Resource (
    Type            => Regular_Processor,
    Name            => CPU2,
    Speed_Factor    => 25);

Scheduler (
   Type            => Primary_Scheduler,
   Name            => CPU1,
   Host            => CPU1,
   Policy          =>
      ( Type    => Fixed_Priority));

Scheduler (
   Type            => Primary_Scheduler,
   Name            => CPU2,
   Host            => CPU2,
   Policy          =>
      ( Type    => Fixed_Priority));

Scheduling_Server (
    Type               => Regular,
    Name               => T1,
    Server_Sched_Parameters     => (
        Type        => Fixed_Priority_policy,
        The_Priority        => 1,
        Preassigned         => Yes),
    Scheduler          => CPU1);

Scheduling_Server (
    Type               => Regular,
    Name               => T2,
    Server_Sched_Parameters     => (
        Type        => Fixed_Priority_policy,
        The_Priority        => 2,
        Preassigned         => Yes),
    Scheduler          => CPU2);
```

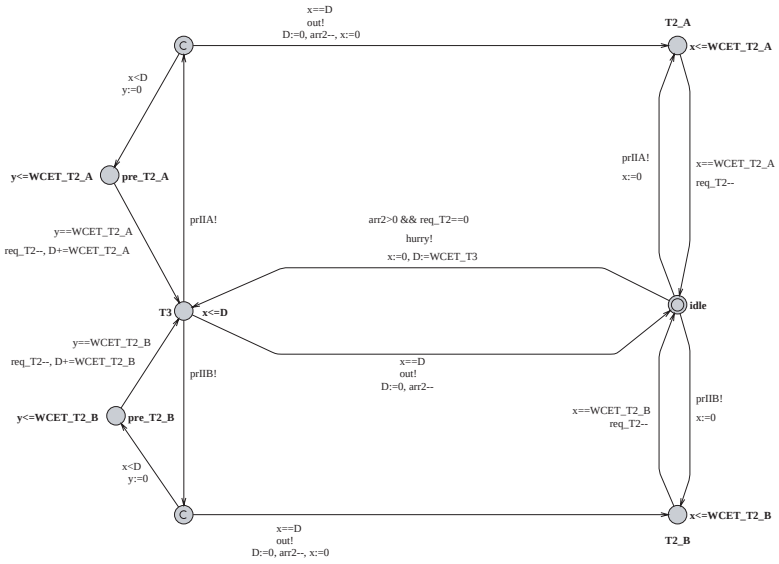
```
Scheduling_Server (
    Type               => Regular,
    Name               => T3,
    Server_Sched_Parameters     => (
        Type        => Fixed_Priority_policy,
        The_Priority        => 1,
        Preassigned         => Yes),
    Scheduler          => CPU2);

Operation (
    Type                      => Simple,
    Name                      => T1,
    Best_Case_Execution_Time => 5,
    Worst_Case_Execution_Time => 20);

Operation (
    Type                      => Simple,
    Name                      => T2,
    Best_Case_Execution_Time => 5,
    Worst_Case_Execution_Time => 15);

Operation (
    Type                      => Simple,
    Name                      => T3,
    Best_Case_Execution_Time => 5,
    Worst_Case_Execution_Time => 5);

Transaction (
    Type => Regular,
    Name => transact1,
    External_Events => (
        (Type   => Periodic,
         Name   => in,
         Period => 4,
         Max_Jitter => 15)),
    Internal_Events => (
        (Type   => regular,
         name   => T1_out),
        (Type   => regular,
         name   => T2_out,
         Timing_Requirements => (
             Type         => Hard_Global_Deadline,
             Deadline     => 1000,
             Referenced_Event => in))),
```

```
    Event_Handlers => (
        (Type           => Activity,
         Input_Event        => in,
         Output_Event       => T1_out,
         Activity_Operation => T1,
         Activity_Server    => T1),
        (Type           => Activity,
         Input_Event        => T1_out,
         Output_Event       => T2_out,
         Activity_Operation => T2,
         Activity_Server    => T2)));

Transaction (
    Type => Regular,
    Name => transact2,
    External_Events => (
        (Type   => Periodic,
         Name   => in,
         Period => 6,
         Max_Jitter => 1)),
    Internal_Events => (
        (Type   => regular,
         name   => T3_out,
         Timing_Requirements => (
             Type         => Hard_Global_Deadline,
             Deadline     => 1000,
             Referenced_Event => in))),
    Event_Handlers => (
        (Type           => Activity,
         Input_Event        => in,
         Output_Event       => T3_out,
         Activity_Operation => T3,
         Activity_Server    => T3)));
```

Uppaal





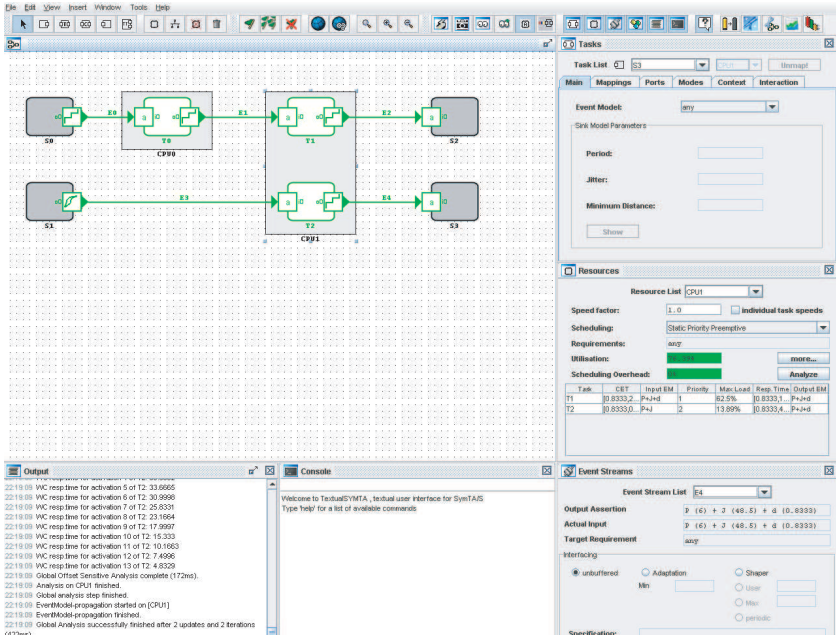


PESIMDES

For conciseness we do not report in this appendix the SystemC source code of the ad-hoc simulator implemented for this case study. The complete source code is available online.[2]

[2] *http://www.mpa.ethz.ch*

SymTA/S



XML (workload corr. blind)

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<performance_analysis xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="perfest.xsd">

<system>

  <resources>
       <FP name="CPU1" preemptive="yes"/>
       <FP name="CPU2" preemptive="yes"/>
  </resources>

  <event_sources>
       <PJD name="I1">
          <period value="4" unit="ms" />
          <jitter value="15" unit="ms" />
          <min_interarr value="1" unit="ms" />
       </PJD>
       <PJD name="I2">
          <period value="6" unit="ms" />
          <jitter value="1" unit="ms" />
       </PJD>
  </event_sources>

  <event_sinks>
       <event_sink name="O1" />
       <event_sink name="O2" />
  </event_sinks>

  <tasks>
       <task name="T1" />
       <task name="T2" />
       <task name="T3" />
  </tasks>

    <task_graphs>
         <task_graph>
             <link src="I1" dest="T1" />
             <link src="T1" dest="T2" />
             <link src="T2" dest="O1" />
         </task_graph>
         <task_graph>
             <link src="I2" dest="T3" />
             <link src="T3" dest="O2" />
         </task_graph>
    </task_graphs>

    <binding>
         <map task="T1" resource="CPU1">
              <wcet value="3.3333" unit="ms" />
              <priority> 1 </priority>
         </map>
         <map task="T2" resource="CPU2">
              <wcet value="2.5" unit="ms" />
              <priority> 1 </priority>
         </map>
         <map task="T3" resource="CPU2">
              <wcet value="0.8333" unit="ms" />
              <priority> 2 </priority>
         </map>
    </binding>

</system>

<observe>
   <latency src="I2" dest="O2" />
</observe>

</performance_analysis>
```

C.7 Models case study 7: Data dependencies

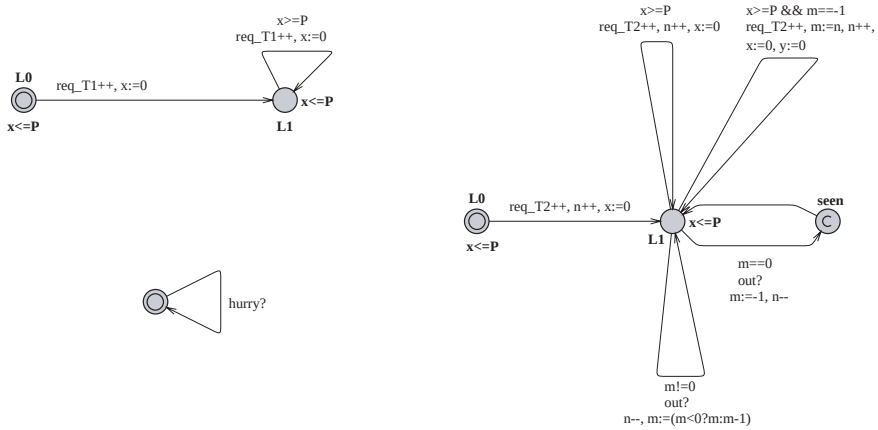
MPA-RTC

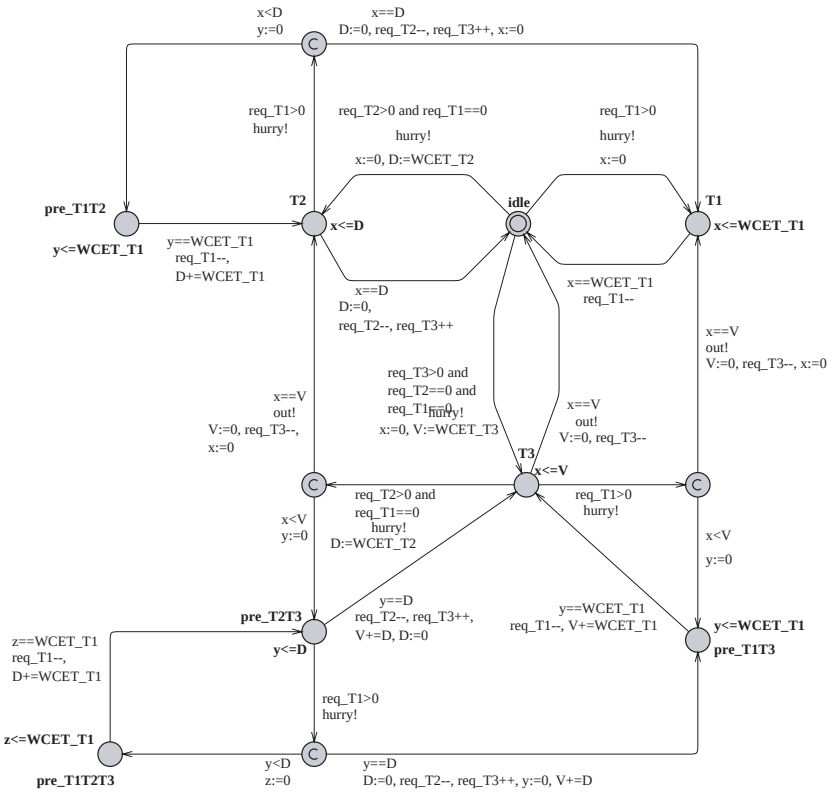
```
p_S0 = 80; p_S1 = 50;
b_CPU0 = 1;
ed_T0 = 15; ed_T1 = 20; ed_T2 = 10;

auo0_S0 = pjdu(p_S0);
alo0_S0 = pjdl(p_S0);
auo0_S1 = pjdu(p_S1);
alo0_S1 = pjdl(p_S1);

buo0_CPU0 = fs(b_CPU0);
blo0_CPU0 = fs(b_CPU0);

[auo1_S0 alo1_S0 buo1_CPU0 blo1_CPU0] = gpc(auo0_S0, alo0_S0, buo0_CPU0, blo0_CPU0, ed_T0);
[auo1_S1 alo1_S1 buo2_CPU0 blo2_CPU0] = gpc(auo0_S1, alo0_S1, buo1_CPU0, blo1_CPU0, ed_T1);
[auo2_S1 alo2_S1 buo3_CPU0 blo3_CPU0] = gpc(auo1_S1, alo1_S1, buo2_CPU0, blo2_CPU0, ed_T2);

d_add = del(auo0_S1,blo1_CPU0,ed_T1) + del(auo1_S1,blo2_CPU0,ed_T2);
d_conv = del(auo0_S1,blo1_CPU0,ed_T1,blo2_CPU0,ed_T2);
d = min(d_conv,d_add);
```

MAST

```
Model (
   Model_Name  => data_dependency,
   Model_Date  => 2006-01-19);

Processing_Resource (
    Type            => Fixed_Priority_Processor,
    Name            => CPU1);

Scheduling_Server (
    Type                => Fixed_Priority,
    Name                => T1,
    Server_Sched_Parameters     => (
        Type        => Fixed_Priority_policy,
        The_Priority        => 3,
        Preassigned         => Yes),
    Server_Processing_Resource  => CPU1);

Scheduling_Server (
    Type                => Fixed_Priority,
    Name                => T2,
    Server_Sched_Parameters     => (
        Type        => Fixed_Priority_policy,
        The_Priority        => 2,
        Preassigned         => Yes),
    Server_Processing_Resource  => CPU1);

Scheduling_Server (
    Type                => Fixed_Priority,
    Name                => T3,
    Server_Sched_Parameters     => (
        Type        => Fixed_Priority_policy,
        The_Priority        => 1,
        Preassigned         => Yes),
    Server_Processing_Resource  => CPU1);

Operation (
    Type                        => Simple,
    Name                        => T1,
    Best_Case_Execution_Time => 29,
    Worst_Case_Execution_Time => 29);

Operation (
    Type                        => Simple,
    Name                        => T2,
    Best_Case_Execution_Time => 20,
    Worst_Case_Execution_Time => 20);

Operation (
    Type                        => Simple,
    Name                        => T3,
    Best_Case_Execution_Time => 10,
    Worst_Case_Execution_Time => 10);

Transaction (
    Type => Regular,
    Name => transact1,
    External_Events => (
        (Type   => Periodic,
         Name   => E1,
         Period => 80)),
    Internal_Events => (
        (Type   => regular,
         name   => O1,
         Timing_Requirements => (
             Type         => Hard_Global_Deadline,
             Deadline     => 1000,
             Referenced_Event => E1))),
    Event_Handlers => (
        (Type           => Activity,
         Input_Event        => E1,
         Output_Event       => O1,
         Activity_Operation => T1,
         Activity_Server    => T1)));

Transaction (
    Type => Regular,
    Name => transact2,
    External_Events => (
        (Type   => Periodic,
         Name   => E2,
         Period => 50)),
    Internal_Events => (
        (Type   => regular,
         name   => O2),
        (Type   => regular,
         name   => O3,
         Timing_Requirements => (
             Type         => Hard_Global_Deadline,
             Deadline     => 1000,
             Referenced_Event => E2))),
    Event_Handlers => (
        (Type           => Activity,
         Input_Event        => E2,
         Output_Event       => O2,
         Activity_Operation => T2,
         Activity_Server    => T2),
        (Type           => Activity,
         Input_Event        => O2,
         Output_Event       => O3,
         Activity_Operation => T3,
         Activity_Server    => T3)));
```

Uppaal

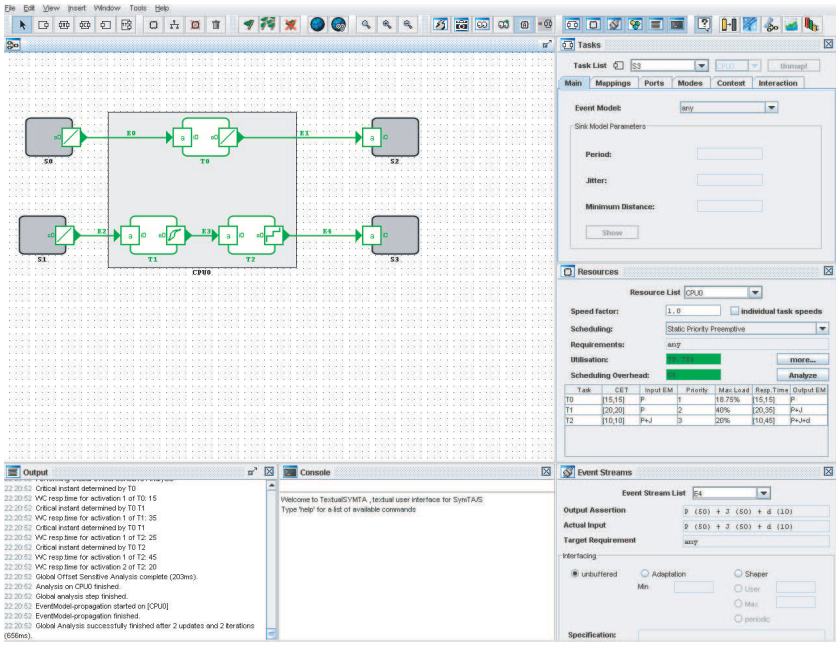

x>=P
req_T1++, x:=0
L0
req_T1++, x:=0
x<=P
x<=P
L1
hurry?
x>=P
req_T2++, n++, x:=0
x>=P && m==-1
req_T2++, m:=n, n++,
x:=0, y:=0
L0
req_T2++, n++, x:=0
x<=P
L1
x<=P
seen
m==0
out?
m:=-1, n--
m!=0
out?
n--, m:=(m<0?m:m-1)


x<D
y:=0
x==D
D:=0, req_T2--, req_T3++, x:=0
req_T1>0
hurry!
req_T2>0 and req_T1==0
hurry!
x:=0, D:=WCET_T2
req_T1>0
hurry!
x:=0
pre_T1T2
y<=WCET_T1
y==WCET_T1
req_T1--,
D+=WCET_T1
T2
x<=D
idle
T1
x<=WCET_T1
x==D
D:=0,
req_T2--, req_T3++
x==WCET_T1
req_T1--
x==V
out!
V:=0, req_T3--, x:=0
req_T3>0 and
req_T2==0 and
req_T1==0
hurry!
x:=0, V:=WCET_T3
x==V
out!
V:=0, req_T3--,
x:=0
x==V
out!
V:=0, req_T3--
T3
x<=V
req_T2>0 and
req_T1==0
hurry!
D:=WCET_T2
req_T1>0
hurry!
x<V
y:=0
x<V
y:=0
y==D
req_T2--, req_T3++,
V+=D, D:=0
y==WCET_T1
req_T1--, V+=WCET_T1
pre_T2T3
y<=D
y<=WCET_T1
pre_T1T3
z==WCET_T1
req_T1--,
D+=WCET_T1
req_T1>0
hurry!
z<=WCET_T1
pre_T1T2T3
y<D
z:=0
y==D
D:=0, req_T2--, req_T3++, y:=0, V+=D

PESIMDES

```
#include "pesimdes.h"
#define MAXBUFFER 1000
bool verbose = false;

int sc_main (int argc , char *argv[]) {

    // initialize the random number generator
   srand(-11);

   // Module instantiations

   // Event stream generators
   input_periodic input_generator_1("I1",sc_time(80,SC_MS),"input_trace_1.tra");
   input_periodic input_generator_2("I2",sc_time(50,SC_MS),"input_trace_2.tra");

   // Tasks
   task t1("T1");
   task t2("T2");
   task t2("T3");

   // Resources
   resource_n_tasks_fixed_priority_preemptive cpu("CPU",3);

   // Event sinks
   output_display display_1("O1");
   output_display display_2("O2");

   // Task mapping
   cpu.assign_task(t1,0,sc_time(15,SC_MS),1);
   cpu.assign_task(t2,1,sc_time(20,SC_MS),2);
   cpu.assign_task(t2,2,sc_time(10,SC_MS),3);

   // Channel instantiations

   // Task activation buffers
   my_sc_fifo<event_token> buffer_T1(MAXBUFFER);
   my_sc_fifo<event_token> buffer_T2(MAXBUFFER);
   my_sc_fifo<event_token> buffer_T3(MAXBUFFER);

   // Dummy buffers for event sinks
   sc_fifo<event_token> dummy_buffer_display_1(1);
   sc_fifo<event_token> dummy_buffer_display_2(1);

   // Port binding

   input_generator_1.out(buffer_T1);
   t1.in[0](buffer_T1);
   t1.out[0](dummy_buffer_display_1);
   display_1.in(dummy_buffer_display_1);

   input_generator_2.out(buffer_T2);
   t2.in[0](buffer_T2);
   t2.out[0](buffer_T3);
   t3.in[0](buffer_T3);
   t3.out[0](dummy_buffer_display_2);
   display_2.in(dummy_buffer_display_2);


   // Start simulation
   sc_start(100000,SC_MS);

   // Print performance results
   cout << "\n Max. observed End-to-End Delay I2-O2: "
        << sc_time_output(display_2.getmaxlatency("I2")) <<
        " (event no. " << display_2.getseqn1oofmaxlatency("I2") << ")\n";

   return 0;
}
```

SymTA/S



XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<performance_analysis xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="perfest.xsd">

<system>

  <resources>
       <FP name="CPU" preemptive="no"/>
  </resources>

  <event_sources>
       <PJD name="I1">
          <period value="80" unit="ms" />
       </PJD>
       <PJD name="I2">
          <period value="50" unit="ms" />
       </PJD>
  </event_sources>

  <event_sinks>
       <event_sink name="O1" />
       <event_sink name="O2" />
  </event_sinks>

  <tasks>
       <task name="T1" />
       <task name="T2" />
       <task name="T3" />
  </tasks>

  <task_graphs>
       <task_graph>
            <link src="I1" dest="T1" />
            <link src="T1" dest="O1" />
       </task_graph>
       <task_graph>
            <link src="I2" dest="T2" />
            <link src="T2" dest="T3" />
            <link src="T3" dest="O2" />
       </task_graph>
  </task_graphs>

  <binding>
       <map task="T1" resource="CPU">
            <wcet value="15" unit="ms" />
            <priority> 1 </priority>
       </map>
       <map task="T2" resource="CPU">
            <wcet value="20" unit="ms" />
            <priority> 2 </priority>
       </map>
       <map task="T3" resource="CPU">
            <wcet value="10" unit="ms" />
            <priority> 3 </priority>
       </map>
  </binding>

</system>

<observe>
   <latency src="I2" dest="O2" />
</observe>

</performance_analysis>
```

Appendix D

Task description (German)

Master thesis

for 1 or 2 students in Department D-ITET/D-INFK/D-BEPR

Evaluation und Vergleich von Methoden zur Performance Analyse von Real-Time Embedded Systems

Während des Designprozesses komplexer Eingebetteter Systeme mit Echtzeit-Bedingungen werden viele Fragen aufgeworfen. Ein Designer ist z.B. typischerweise daran interessiert, ob die Zeiteigenschaften einer bestimmten System-Architektur die Echtzeit-Bedingungen erfüllen werden, wie stark verschiedene On-Chip-Kommunikationskanäle ausgelastet sein werden, welcher Bus oder Prozessor der Flaschenhals sein wird, oder was die On-Chip-Speicheranforderungen sein werden. In den letzten Jahren wurde am TIK und auch weltweit viel Forschung betrieben, um Modelle und Methoden zu entwickeln, welche es erlauben die oben genannten Eigenschaften eines Systems schon in einer sehr frühen Designphase zu analysieren. Während auf diesem Gebiet der sogenannten System Level Performance Analyse in den letzten Jahren viele Fortschritte gemacht wurden, existieren bis heute noch keine Studien, welche die verschiedenen Methoden auf wissenschaftlicher Ebene evaluieren und miteinander vergleichen.

Bei dieser Masterarbeit sollen in einem ersten Schritt sinnvolle Kriterien zur Beurteilung von Performance Analyse Methoden entwickelt werden. Anhand dieser Kriterien sollen dann in einem weiteren Schritt eine Anzahl aktueller Methoden evaluiert werden. Basierend aud diesen Evaluationen sollen die verschiedenen Methoden schliesslich verglichen werden, und Stärken und Schwächen der verschiedenen Methoden identifiziert werden.

Kind of Work: 50% Theorie, 30% Konzept, 20% Implementierung
Requirements: Interesse an wissenschaftlicher Arbeit
Contact Person: Ernesto Wandeler, ETZ G75, +41 44 63 24528
Tutors: Ernesto Wandeler, Simon Künzli
Professor: Prof. Lothar Thiele

Bibliography

[1] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[2] Tobias Amnell, Elena Fersman, Leonid Mokrushin, Paul Pettersson, and Wang Yi. Times - a tool for modelling and implementation of embedded systems. In *TACAS '02: Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 460–464, London, UK, 2002. Springer-Verlag.

[3] Gerd Behrmann, Alexandre David, and Kim G. Larsen. A tutorial on UPPAAL. In Marco Bernardo and Flavio Corradini, editors, *Formal Methods for the Design of Real-Time Systems: 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004*, number 3185 in LNCS, pages 200–236. Springer–Verlag, September 2004.

[4] Jean-Yves Le Boudec and Patrick Thiran. *Network calculus: a theory of deterministic queuing systems for the internet*. Springer-Verlag New York, Inc., New York, NY, USA, 2001.

[5] Giorgio C. Buttazzo and Giorgio Buttanzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[6] S. Chakraborty, S. Künzli, and L. Thiele. A general framework for analysing system properties in platform-based embedded system designs. In *Proc. 6th Design, Automation and Test in Europe (DATE)*, pages 190–195, Munich, Germany, March 2003.

[7] Seamless (Mentor Graphics). `http://www.mentor.com/products/fv/hwsw_coverification/seamless/index.cfm`.

[8] Constance Heitmeyer and Dino Mandrioli. *Formal Methods for Real-Time Computing*. John Wiley & Sons, Inc., New York, NY, USA, 1996.

[9] Martijn Hendriks and Marcel Verhoef. Timed automata based analysis of embedded system architectures. Technical Report ICIS-R06003, ICIS, Radboud University, Nijmegen, The Netherlands, 2006.

[10] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst. System level performance analysis – the symta/s approach. *IEE Proceedings - Computers and Digital Techniques*, 152(2):148–166, 2005.

[11] Mathai Joseph and Paritosh K. Pandya. Finding response times in a real-time system. *BCS Computer Journal*, 29(5):390–395, 1986.

[12] Simon Künzli and Lothar Thiele. Generating event traces based on arrival curves. In *Proc. of 13th GI/ITG Conference on Measurement, Modeling, and Evaluation of Computer and Communication Systems (MMB)*, March 2006.

[13] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[14] J.C. Palencia Gutirrez M. Gonzlez Harbour, J.J. Gutirrez Garca and J.M. Drake Moyano. Mast: Modeling and analysis suite for real time applications. In *Proceedings of 13th Euromicro Conference on Real-Time Systems*, pages 125–134. IEEE Computer Society Press, 2001.

[15] Peter Marwedel. *Embedded System Design*. Kluwer Academic Publishers, 2003.

[16] A. Maxiaguine, S. Künzli, and L. Thiele. Workload characterization model for tasks with variable execution demand. In *Proc. 7th Design, Automation and Test in Europe (DATE)*, pages 1040–1045, Paris, France, February 2004.

[17] Christer Norström, Anders Wall, and Wang Yi. Timed automata as task models for event-driven systems. In *RTCSA '99: Proceedings of the Sixth International Conference on Real-Time Computing Systems and Applications*, page 182, Washington, DC, USA, 1999. IEEE Computer Society.

[18] ARTIST2 Workshop on Distributed Embedded Systems 2005. `http://www.tik.ee.ethz.ch/~leiden05/`.

[19] The Open SystemC Initiative (OSCI). `http://www.systemc.org`.

[20] Simon Perathoner, Ernesto Wandeler, and Lothar Thiele. Timed automata templates for distributed embedded system architectures. Technical Report 233, Computer Engineering and Networks Laboratory, Swiss Federal Institute of Technology, Zurich, Switzerland, November 2005.

[21] Paul Pop, Petru Els, and Zebo Peng. Performance estimation for embedded systems with data and control dependencies. In *CODES '00: Proceedings of the eighth international workshop on Hardware/software codesign*, pages 62–66, New York, NY, USA, 2000. ACM Press.

[22] Traian Pop, Petru Eles, and Zebo Peng. Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems. In *CODES '02: Proceedings of the tenth international symposium on Hardware/software codesign*, pages 187–192, New York, NY, USA, 2002. ACM Press.

[23] Kai Richter, Marek Jersak, and Rolf Ernst. A formal approach to mpsoc performance verification. *Computer*, 36(4):60–67, 2003.

[24] System Studio (Synopsis). `http://www.synopsys.com/products/cocentric\_studio/cocentric\_studio.html`.

[25] Lothar Thiele and Ernesto Wandeler. Performance analysis of distributed embedded systems. In Richard Zurawski, editor, *Embedded Systems Handbook*. CRC Press, 2005.

[26] Ken Tindell. Adding time-offsets to schedulability analysis. Technical Report YCS-94-221, University of York, Computer Science Dept.

[27] Ken Tindell and John Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocess. Microprogram.*, 40(2-3):117–134, 1994.

[28] Ernesto Wandeler, Alexander Maxiaguine, and Lothar Thiele. Quantitative characterization of event streams in analysis of hard real-time applications. *Real-Time Systems*, 29(2-3):205–225, 2005.

[29] Ernesto Wandeler and Lothar Thiele. Abstracting functionality for modular performance analysis of hard real-time systems. In *Asia and South Pacific Desing Automation Conference (ASP-DAC)*, pages pages 697–702, 2005.

[30] Ernesto Wandeler and Lothar Thiele. Characterizing workload correlations in multi processor hard real-time systems. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 46–55, 2005.

[31] SoC Designer with MaxSim Technology (ARM). `http://www.arm.com/products/DevTools/MaxSim.html`.

[32] Ti-Yen Yen and Wayne Wolf. Performance estimation for real-time distributed embedded systems. In *ICCD '95: Proceedings of the 1995 International Conference on Computer Design*, pages 64–71, Washington, DC, USA, 1995. IEEE Computer Society.