Real-Time Systems, 29, 205–225, 2005
© 2005 Springer Science + Business Media, Inc. Manufactured in The Netherlands.

Quantitative Characterization of Event Streams in Analysis of Hard Real-Time Applications

ERNESTO WANDELER wandeler@tik.ee.ethz.ch
ALEXANDER MAXIAGUINE maxiagui@tik.ee.ethz.ch
LOTHAR THIELE thiele@tik.ee.ethz.ch
Computer Engineering and Networks Laboratory, Swiss Federal Institute of Technology (ETH) Zurich, Switzerland

Abstract. Many real-time embedded systems process event streams that are composed of a finite number of different event types. Each different event type on the stream would typically impose a different workload to the system, and thus the knowledge of possible correlations and dependencies between the different event types could be exploited to get tighter analytic performance bounds of the complete system. We propose an abstract stream model to characterize such an event stream. The model captures the needed information of all possible traces of a class of event streams. Hence, it can be used to obtain hard bounded worst-case and best-case analysis results of a system. We show how the proposed abstract stream model can be obtained from a concrete stream specification, and how it can be used for performance analysis. The applicability of our approach and its advantages over traditional worst-case performance analysis are shown in a case study of a multimedia application.

Keywords: performance analysis, real-time calculus, network calculus, end-to-end delay analysis, real-time multimedia analysis

1. Introduction

A large class of real-time embedded systems contain processing units, in which tasks are triggered by the events of an incoming event stream. These events can often be classified into a finite number of different event types, where each event type typically triggers different tasks and therefore imposes a different workload to the processing unit. In particular, each different event type will usually have different worst-case and best-case execution requirements (WCET and BCET) on the processing unit. Typical examples of such systems are multimedia applications (Hughes et al., 2001).

When confronted with the problem of performance analysis of such a system, we would usually be interested in finding upper and lower bounds for the workload imposed by the whole event stream rather than by individual events within the stream. This would enable us to exploit the knowledge of possible correlations and dependencies between different event types present on the stream. And since usually not all event types have the same WCET (BCET), this analysis would often lead to tighter worst-case (best-case) performance bounds, without sacrificing guaranteed bounds.

Common existing analysis methods however have no means to exploit the correlations and dependencies of different event types in an event stream, they usually not even have a notion of different event types. To obtain hard bounds for the above mentioned system, these methods would assume each event to have the largest possible WCET (smallest BCET) any event could have on the processing unit (Liu and Layland, 1973). They would therefore approximate the worst-case (best-case) stream as consisting of a sequence of only the single event type with this largest WCET (smallest BCET). While this approach can be used for hard real-time analysis of a system, it will in general lead to overly pessimistic bounds and therefore to unnecessary expensive system designs (Bernat et al., 2002).

A common technique used to get tighter bounds and therefore less expensive system designs, is to use a set of benchmark applications to obtain the bounds for their specific workload (Lee et al., 1997). Such a straightforward approach however suffers from the major disadvantage of all simulation- and trace-based methods, that is the limitation to a particular realization of a workload. Since any concrete realization of a workload can in general not guarantee to cover all corner cases, this approach does not lead to guaranteed worst-case and best-case bounds and can therefore not be used in the analysis of hard real-time systems.

Other existing techniques use probabilistic models to describe the timing behavior of real-time systems (Tia et al., 1995). Such models can capture whole families of systems with similar statistical characteristics. The resulting bounds obtained from an analysis with these probabilistic models will however again be of probabilistic nature and hence are also not applicable for the analysis of hard real-time systems.

It becomes apparent that in order to obtain tighter, and at the same time guaranteed bounds for the workload imposed by an event stream, we need to develop proper abstractions that characterize such streams. The abstractions should thereby capture all possible traces of a class of streams in order to contain the information needed for obtaining hard bounded worst-case and best-case results.

Recently, several methods were presented that try to address this problem. A *logic of constraints* (LOC) developed in Balarin et al. (2001) can be used to capture in abstract form desirable properties of event streams such as event arrival patterns and the workload variability. The main purpose of LOC is to provide a formal way to specify system *requirements* instead of providing stream abstractions suitable for use in analytical methods for performance analysis.

Baruah (2003) uses *demand bound functions* to model variable workload produced by a task execution sequence under real-time constraints. Although the model captures conditional execution, it does not try to distinguish between different event types within the triggering event sequence of an individual task and, therefore, it cannot exploit knowledge about correlations among the event types within the stream.

In Jersak et al. (2004), propose a performance analysis technique that uses *system contexts* to improve analytic bounds. The system contexts represent various kinds of correlations in task execution sequences within a system. They show that exploiting knowledge about the structure of correlated event streams may result in improved bounds than without using it. However, the methods and models described in Jersak et al. (2004) are confined to particular problem instances and cannot be applied in a general context.

In the domain of communication networks, powerful abstractions have been developed to model flow of data (packets) through a network. The theoretical framework, called *Network Calculus* (Le Boudec and Thiran, 2001), provides means to deterministically reason about timing properties of the data flows. In our research, we extend the basic concepts of the Network Calculus to the domain of real-time embedded systems. More specifically, we provide means to model structure and variable execution demand of streams.

Contributions

In this research, we propose a new abstract model for event streams containing a finite number of different event types. The model has simple and clear semantics and implicitly captures all possible traces of a whole class of streams. Therewith, it contains the information needed to quantitatively reason on hard bounds for the stream.

We then present an analytic method, that can be used to obtain our proposed abstract stream models for any event stream that is specified by a finite state machine. Further we present a method, how the so obtained abstract stream models can be used to compute the workload characteristics, which the stream imposes on a particular processing unit.

We show the applicability of the presented methods in a detailed case study of a multimedia application, where we obtain tighter bounds for the systems worst-case performance than with conventional methods. Using the obtained results, the system could be implemented more power efficient and cheaper without violating its real-time constraints.

2. Abstract Stream Models

Suppose we have a system, where an event stream with different event types, generated by a stream generator, triggers the input of a processing unit. Each different event type will typically put a different workload on the processing unit, i.e. will have a different WCET and BCET.

The upper part of Figure 1 depicts a physical representation of such a system. On the left side, we have the stream generator that generates an instance of an event stream containing different event types. Further, we have a table with WCET's and BCET's for every event type, which we obtain from the processing unit. When the event stream arrives at the processing unit, every event on the stream imposes a workload, which is bounded by the WCET and the BCET of its event type. From the performance analysis point of view, we can therefore imagine the event stream as a stream of workloads. This workload stream represents the workload, that the event stream is imposing on the processing unit.

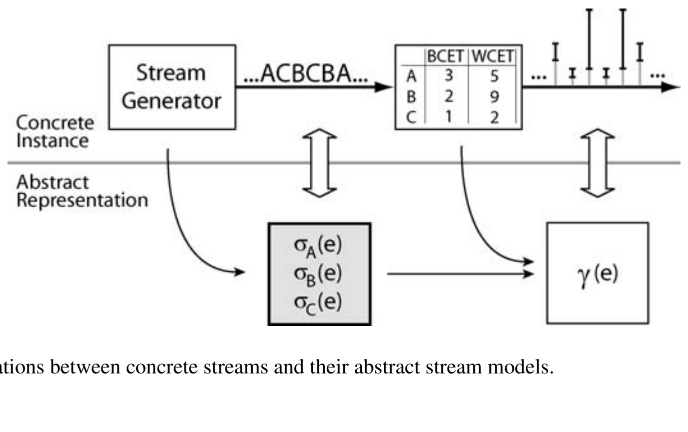

Figure 1. Relations between concrete streams and their abstract stream models.

For the performance analysis of such a system, we need an abstract stream representation, that represents all possible traces of a class of streams, and hence contains the information needed to obtain hard bounded worst-case and best-case results.

The lower part of Figure 1 depicts such an abstract representation of the system, where on the right side, $\gamma(e)$ represents the abstract workload stream. $\gamma(e)$ is called *workload curve* and was first introduced in Maxiaguine et al. (2004).

The concept of the abstract workload curves is very generic and allows their usage in several different contexts. In particular, workload curves can be used

- to model the usage of communication or computation resources by streams, where the workload they represent is expressed either in number of data bits or processing cycles,
- for performance analysis of multiprocessor stream architectures in frameworks as for example (Chakraborty et al., 2003) and
- to improve bounds in classical schedulability analysis of real-time systems, see e.g. (Liu and Layland, 1973).

So far, there are no methods to analytically deduce workload curves from available information about the system. To this end, we introduce the concept of *type rate curves*, that represent the abstract event stream with its different event types. In the lower part of Figure 1, the type rate curves $\sigma_A(e)$, $\sigma_B(e)$ and $\sigma_C(e)$ are shown on the left side.

3. A Quantitative Model for Event Streams

3.1. Type Rate Curves

Consider an event stream s, in which events of n different event types t_i may be present and let $E_i(u)$ be a counter, that counts the number of events of type t_i, occurring during u consecutive events on the event stream.

Definition 1 (Type Rate Curve). For any event stream s and for any event type t_i occurring on the stream, the lower type rate curve σ_i^l and the upper type rate curve σ_i^u satisfy the relation:

$$\sigma_i^l(v-u) \leq E_i(v) - E_i(u) \leq \sigma_i^u(v-u) \quad \forall 0 \leq u \leq v$$

Therefore, in any sequence of e consecutive events of the event stream s, at least $\sigma_i^l(e)$ and at most $\sigma_i^u(e)$ events are of type t_i.

Using the above definition, an event stream s can quantitatively be characterized by the complete set of type rate curves $\{\sigma_i^u, \sigma_i^l\}$ of all event types occurring in it.

Example 1. As a simple example let us consider an event stream containing three different event types A, B and C respectively, that always occur strictly following one of two patterns,

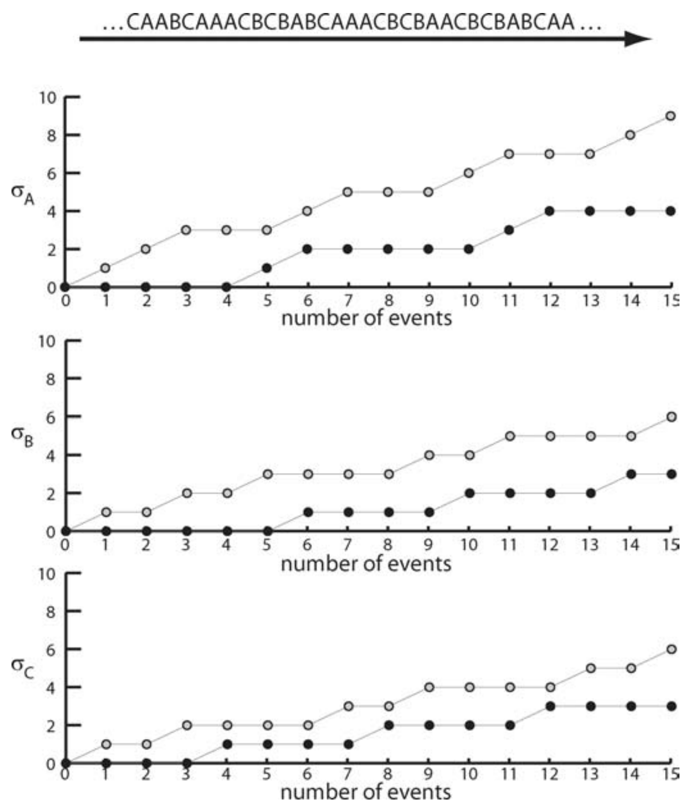

Figure 2. A short sequence of the event stream of Example 1 and its type rate curves.

either "*ABCBCA*" or "*AACB*". The order in which the patterns themselves occur on the stream is assumed to be completely random.

In Figure 2, a short sequence of the above specified event stream is shown, together with the complete set of type rate curves for the stream. On the x-axis of the type rate curves, the sequence length is given in number of events, whereas the values on the y-axis give the maximum and minimum number of times, an event of the specific type can occur in any sequence of this length.

Using these type rate curves, we can directly obtain information on possible type compositions in event sequences of the stream. For example we can see that in any event sequence of length 12, at least 4 and at most 7 events are of type A and at least 2 and at most 5 events are of type B, whereas the number of events of type C must lie between 3 and 4.

Type rate curves are an abstract representation of an event stream, describing in a compact way the event type correlations in it. It is thereby important to note that a set of type rate curves does not only describe a single instance of an event stream sequence, but rather a whole class of event streams. Only by that it is possible that type rate curves contain the information needed in worst case analysis.

A set of type rate curves is a very generic model of an event stream, that can capture arbitrarily complex dependencies and correlations on the occurrence of event types in the stream. In case of completely uncorrelated event types, that are statistically independent from each other, the event rate curves abstracting such a stream would have the form $\sigma^u(e) = e$ and $\sigma^l(e) = 0$ respectively. The existence of any dependencies and correlations on the other hand will immediately become apparent by event rate curves having the form $\sigma^u(e) < e$ and $\sigma^l(e) > 0$ for some values of e.

3.2. Obtaining Type Rate Curves

In order to be applicable in the analysis of hard real-time systems, the set of type rate curves of a stream must represent guaranteed bounds on the possible occurrence of different event types. In consequence of this requirement, they must be obtained from a formal stream specification or description, by using analytical methods.

There exist a variety of alternatives to formally specify event streams with different event types, some of which are finite state machines, arrival curves (Le Boudec and Thiran, 2001), where every event type as well as the complete stream itself would each be specified by a separate arrival curve, or some sort of formal notation like Logic of Constraints (Balarin et al., 2001), where stream properties such as rate, jitter or burstiness can be formally specified.

From virtually any of these specification methods we can analytically derive the set of type rate curves that abstract the event stream they describe. Exemplary, we introduce in the remaining part of this section a method to compute type rate curves for an event stream which is specified by a finite state machine.

Note, in practice it would sometimes also be useful to obtain type rate curves from simulation traces, especially if not enough knowledge is available of the correlations of different event types on the particular event stream. Even though the so obtained curves must not be used for the analysis of hard real-time systems, they can be useful for the analysis of soft real-time systems.

Computing Type Rate Curves From Finite State Machines

Event streams that are made up from a finite number of event patterns that may occur in a non-deterministic order, can be specified by a non-deterministic finite state machine, which we will call FSM_S, with a set of vertices V, a set of directed edges E and an alphabet Σ consisting of the different event types occurring on the stream. Every edge is labelled with a symbol of Σ and any run in FSM_S is a valid instance of a stream specified by the machine.

To compute the type rate curves of a given event type t_i for a stream specified by such a machine, we need to follow a procedure that can be broken down into three steps and that is described below.

Step 1: First we need to transform FSM_S into a type-specific finite state machine FSM_{t_i}. FSM_{t_i} has the same set of vertices V and edges E, but its edges are not labelled by the

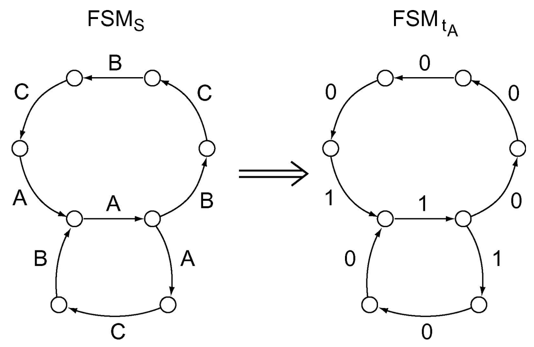

Figure 3. FSM_S and its FSM_{t_A}.

alphabet Σ, but are instead annotated with a weight w, such that every edge initially labelled with a symbol t_i has weight 1, while all other edges have weight 0. Figure 3 shows FSM_S and FSM_{t_A} for the stream specified in Example 1.

Step 2: In the type-specific finite state machine FSM_{t_i}, the value of the upper type rate curve $\sigma_i^u(e)$ equals the weight $w^u(e)$ of the maximum-weight path with length e, while the value of the lower type rate curve $\sigma_i^l(e)$ equals the weight $w^l(e)$ of the minimum-weight path with length e.

We can use Algorithm 1 to compute the weights $w^u(e)$ and $w^l(e)$ of the maximum-weight and minimum-weights paths of a finite state machine respectively up to a path length of n. The finite state machine thereby corresponds to a directed weighted graph $G(V, E, W)$ with vertices V, edges E and weights W.

The computational complexity of Algorithm 3.2 is $O(n|V||E|)$, while the memory requirement has a complexity of $O(|V|)$.

Algorithm 1. *Computing the weights of the maximum-weight and minimum-weight paths in a weighted graph G(V,E,W)*

Given a function $pred(v)$, that returns the set of all predecessor vertices of vertex v.
Given a function $weight^u(u, v)$ and a function $weight^l(u, v)$, that return the maximum weight and the minimum weight respectively, of all edges starting at vertex u and ending at vertex v.

```
$w_v^u(0) = 0, \forall v \in V$
$w_v^l(0) = 0, \forall v \in V$
for $i = 1$ to $n$ do
  for $\forall v \in V$ do
    if $|pred(v)| > 0$ then
      $w_v^u(i) = \max_{p \in pred(v)}\{w_p^u(i-1) + weight^u(p, v)\}$
      $w_v^l(i) = \min_{p \in pred(v)}\{w_p^l(i-1) + weight^l(p, v)\}$
    else
```

```
        $w^u_v(i) = -\infty$
        $w^l_v(i) = +\infty$
      end if
    end for
    $w^u(i) = \max_{v \in V}\{w^u_v(i)\}$
    $w^l(i) = \min_{v \in V}\{w^l_v(i)\}$
end for
```

Step 3: The values $w^u(e)$ represent $\sigma_i^u(e)$ for $e \leq n$, while for larger e, we can use their periodic continuation:

$$\sigma_i^u(e) = \left\lfloor \frac{e}{n} \right\rfloor \cdot \sigma_i^u(n) + \sigma_i^u\left(e - \left\lfloor \frac{e}{n} \right\rfloor n\right)$$

Since for the weights of maximum-weight paths in a weighted graph the relation $w^u(e_1 + e_2) \leq w^u(e_1) + w^u(e_2)$ holds true for $\forall e_1, e_2 \in \mathbb{Z}_{\geq 0}$ *(subadditivity)*, it can be shown that this periodic continuation provides a hard upper bound on the weight of the maximum-weight path of any length $e \in \mathbb{Z}_{\geq 0}$ and thus is a valid upper type rate curve for event type t_i.

Similarly, it can be shown that the periodic continuation of $w^l(e)$ is a valid lower type rate curve $\sigma_i^l(e)$ for event type t_i:

$$\sigma_i^l(e) = \left\lfloor \frac{e}{n} \right\rfloor \cdot \sigma_i^l(n) + \sigma_i^l\left(e - \left\lfloor \frac{e}{n} \right\rfloor n\right)$$

Using the presented method, tight (i.e. exact) type rate curves of a stream specified by FSM_S can be computed up to an arbitrary sequence length n, while hard upper and lower bounds can be obtained for any larger lengths.

For strongly connected graphs and with some more effort, it is also possible to obtain the tight type rate curves of a stream specified by FSM_S for all $e \in \mathbb{Z}_{\geq 0}$. The according techniques are described in Cohen et al. (1985) and (Karp, 1978).

3.3. A Formal Method to Compute Workload Curves

Since type rate curves are an abstract representation of only an event stream, they are completely independent from any concrete system. However, in order to use type rate curves for the analysis of a specific system, we need to combine the information contained in the type rate curves with the information available of the system. The result of this combination will be workload curves, that are an abstract representation of the workload generated in the system by the incoming event stream.

We will now present a formal method to compute workload curves for a given event stream by using its set of type rate curves. For this, we will first recall the definition of workload curves.

Given an event stream s, that is translated into a workload stream w. Let $W(u)$ be a counter that accumulates the total workload imposed to a processing unit by u consecutive events of the event stream.

Definition 2 (Workload Curve). For any workload stream w, the lower workload curve γ^l and the upper workload curve γ^u satisfy the relation:

$$\gamma^l(v-u) \leq W(v) - W(u) \leq \gamma^u(v-u) \quad \forall 0 \leq u \leq v$$

Therefore, any sequence of e consecutive events on the workload stream w, will impose a total workload of at least $\gamma^l(e)$ and at most $\gamma^u(e)$ to a processing unit.

Since workload curves always describe the workload imposed by a stream to a specific system, we need some information about this system, which in our case is expressed by the set of WCET and BCET for every event type occurring on the event stream.

Before we can compute the workload curves, we first need to collect all available data for each of the n event types into a data set $data_i = (\sigma_i^u, \sigma_i^l, wcet_i, bcet_i)$. The indices of these data sets must then be reordered, such that $data_1$ contains the data of the event type with the largest *wcet*, while $data_2$ represents the event type with the second largest *wcet*, and so on. Using the data in these reordered data sets, the following formula can be used to compute the upper workload curve $\gamma^u(e)$ for the specified system:

$$\gamma^u(e) = \sum_{i=1}^{n} \min\left\{ \max\left\{ e - \left(\sum_{k=1}^{i-1} \sigma_k^u(e) + \sum_{k=i+1}^{n} \sigma_k^l(e) \right), \sigma_i^l(e) \right\}, \sigma_i^u(e) \right\} \cdot wcet_i$$

The above formula computes for every $e \in \mathbb{Z}_{\geq 0}$ the worst-case workload, that an event sequence of length e could possibly impose to the system. In this worst-case event sequence, at least $\sigma_i^l(e)$ events of every event type t_i must be present. After fulfilling all these minimum requirements, the remaining sequence is filled up using events of the type t_i with largest possible *wcet*, while still considering the upper bound $\sigma_i^u(e)$ on the possible occurrence for every event type.

The lower workload curve $\gamma^l(e)$ can be computed in a very similar way. We only need to replace $wcet_i$ by $bcet_i$ in the above formula and reorder the indices of the data sets, such that $data_1$ contains the data of the event type with the smallest *bcet*.

Figure 4 shows the workload curve of the event stream of Example 1, that was computed using the type rate curves from Figure 2 and the WCET and BCET from the table in Figure 1. The x-axis denotes the sequence length in number of events, whereas the values on the y-axis give the workload imposed to a system by the stream. The upper and lower straight lines show the worst- and best-case workload bounds of the stream, that we would get without distinguishing between different event types.

4. Case Study

By giving an example of a simple design problem, we demonstrate the advantage of using type rate curves in performance analysis of real-time systems. For the solution of the problem, we employ in one case type rate curves and in the other we follow a common way to characterize event streams, i.e. we do not make a distinction between various event

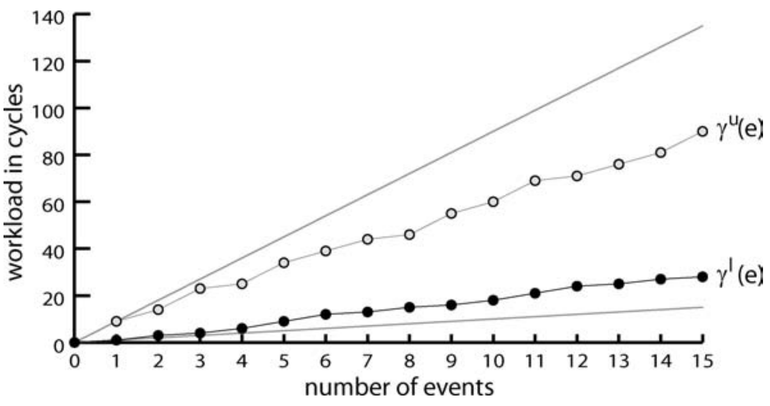

Figure 4. The workload curve for the event stream of Example 1.

types in a stream. We compare results obtained from both of the approaches to show the gain resulting from the application of type rate curves.

4.1. Application Scenario

For our example design problem, let us consider an application scenario as depicted in Figure 5.

Figure 5 shows a basic system model of a networked multimedia device. The device receives on its inputs real-time audio and video streams, compresses them and sends them to a network. The compression is performed by tasks running on a media processor.

Audio and video frames periodically arrive into corresponding input FIFO buffers. For each incoming audio or video frame the media processor executes an associated compression task. The audio stream is processed by a MP3 task, whereas the video stream is compressed by a MPEG-2 encoding task. After the processing, the streams are sent to output FIFO buffers.

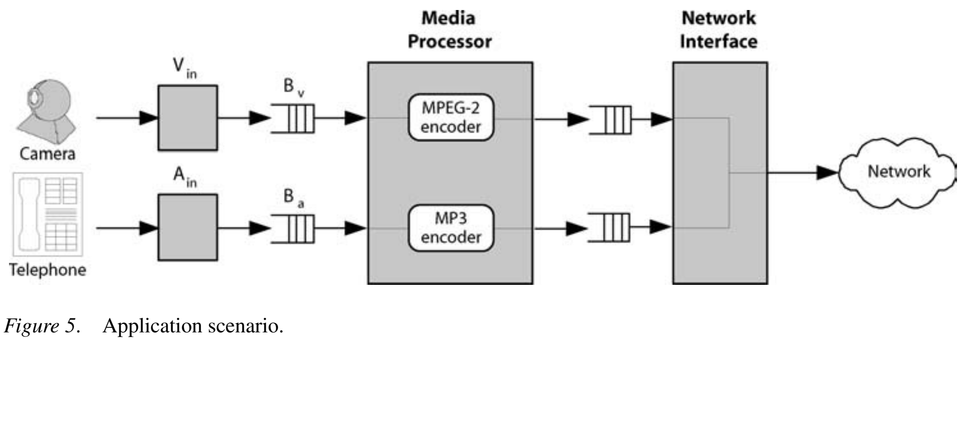

Figure 5. Application scenario.

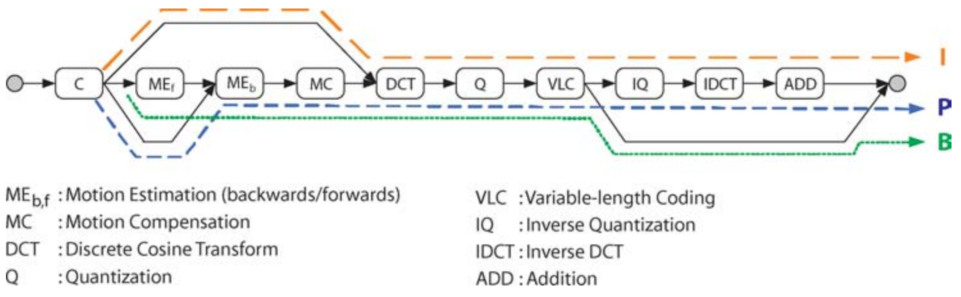
C
ME$_f$
ME$_b$
MC
DCT
Q
VLC
IQ
IDCT
ADD
I
P
B
ME$_{b,f}$: Motion Estimation (backwards/forwards)
MC : Motion Compensation
DCT : Discrete Cosine Transform
Q : Quantization
VLC : Variable-length Coding
IQ : Inverse Quantization
IDCT : Inverse DCT
ADD : Addition

Figure 6. Task graph of the MPEG-2 encoder showing processing of different frame types.

In our example we will use workload curves to characterize processing load imposed by the MPEG-2 encoding task on the media processor. We will obtain the workload curves from the the type rate curves of the video stream. For this, we need to know some details about the behavior of the MPEG-2 encoder, which we explain here.

The MPEG-2 compression scheme exploits three types of frames to encode video information. These are I-, P- and B-frames. Before the compression, each input video frame is assigned one of these types. Depending on the frame type, the encoder will execute different subtasks to compress the frame, as shown in Figure 6. As a result, different frame types will impose different execution demands on the processor.

The MPEG-2 standard[1] does not specify any particular implementation of the encoder algorithm. A designer, therefore, can choose between many different implementation options. In particular, the standard does not specify specific frame patterns that can be used by the encoder to compress a video sequence. Advanced encoders use several predefined patterns and can arbitrary switch between them based on the characteristics of the input video sequence. Likewise our encoder can generate three commonly used patterns: IPB, IPBB and IPBBPBB. Furthermore, if a scene change has been detected in the input video stream, the encoder can interrupt the currently generated pattern at any place and start generation of a new one. This behavior is, however, subject to a constraint: whenever a scene change has been detected and a new pattern has been started, at least three consecutive frames must be encoded *without* pattern interruption. An FSM describing the frame pattern generation process is shown in Figure 7.

The processing capacity of the media processor is shared between the video and audio streams. We assume that the video stream has a higher priority than the audio stream and the processor scheduler is preemptive. Further, we assume that no other tasks except for the video and audio encoding tasks are running on the processor.

The video and audio streams must be processed in real time. Since the audio stream has lower priority than the video stream, audio frames may experience a processing delay, that depends on interference from the video encoding task. To ensure quality of the audio stream, we impose a constraint on the delay. We require the delay to be not greater than some value.

Our design problem is to determine minimum clock rate of the media processor such that the *delay constraint* for the audio stream is satisfied.

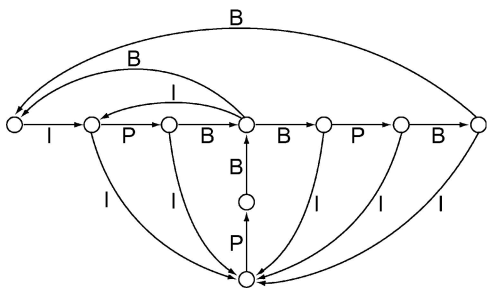

Figure 7. FSM specifying frame patterns generated by the MPEG-2 encoder.

We will provide a general solution to the problem based on the theory of Real-Time Calculus (Chakraborty et al., 2003), that has its roots in Network Calculus (Cruz, 1991; Le Boudec and Thiran, 2001). In the next subsection we give the necessary theoretical background.

4.2. Theoretical Background: Real-Time Calculus

Real-Time Calculus uses abstract models to capture timing properties of event streams and capabilities of processing resources. Timing properties of event streams are modelled by *arrival curves*, whereas the capabilities of processing resources are represented by *service curves*.

Definitions of arrival and service curves are similar to that of workload curves. An arrival curve $\bar{\alpha}(\Delta)$ of an event stream is defined as an *upper bound* on the number of *events* seen in the stream within any time interval Δ.

The processing capabilities of a processor (or communication bus) are usually expressed in number of processor (bus) cycles per time unit. Thus, a service curve $\beta(\Delta)$ is defined as a *lower bound* on the number of *cycles* available to an event stream within any time interval Δ.

To obtain useful performance metrics for a system we combine the arrival and service curves in a computation. However, before we can combine the curves we must be able to transform event-based quantities into cycle-based quantities and vice versa. For this purpose, we introduce the following notation. $\alpha(\Delta)$ denotes the upper bound on processor *cycles* that may be requested by an event stream in any time interval Δ. $\bar{\beta}(\Delta)$ denotes the lower bound on number of *events* that can be processed by a resource within any time interval Δ. Then, we can employ the upper workload curve to do the transformations:

$$\alpha(\Delta) = \gamma^u(\bar{\alpha}(\Delta)) \tag{1}$$

$$\beta(\Delta) = \gamma^u(\bar{\beta}(\Delta)) \tag{2}$$

One performance metric that can be calculated using Real-Time Calculus is an upper bound on the delay, that may be experienced by an event stream due to processing on a

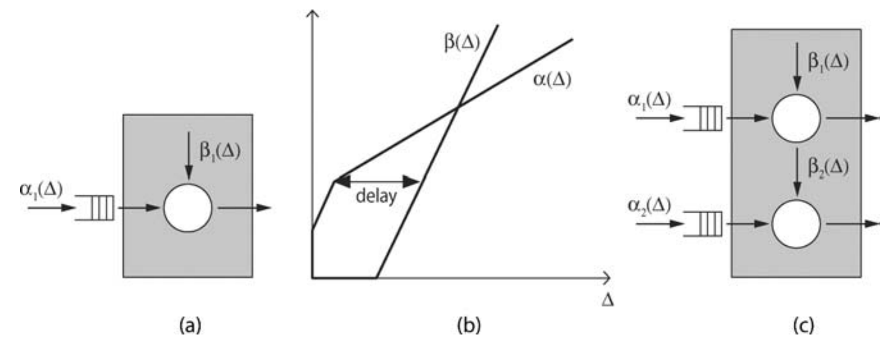

Figure 8. Abstraction of stream processing (a), calculation of the upper bound on the processing delay (b) and abstraction of fixed-priority scheduling (c).

communication or computational resource. To compute the upper bound on the processing delay we combine the arrival curve of an event stream and the service curve provided to this stream as follows:

$$delay \leq \sup_{\Delta \in \mathbb{R}_{\geq 0}} \{\inf\{\tau \geq 0 : \bar{\alpha}(\Delta) \leq \bar{\beta}(\Delta + \tau)\}\} \tag{3}$$

Figure 8(b) gives a graphical representation of (3).

When several event streams are processed by a single resource, each of them gets only a portion of the full processing capacity of the resource. A scheduler decides which of the streams gets the service at every instant of time. If a stream has not completely used the portion of the service assigned to it, the remaining part of the service can be transferred for processing of other streams. The *remaining service* can be computed using the formula below.

$$\beta_r(\tau) = \sup_{\forall \Delta \in [0,\tau]} \{\beta(\Delta) - \alpha(\Delta)\} \tag{4}$$

In (4) β and α denote service and arrival curves of the stream respectively, and β_r designates the remaining service curve. Figure 8)c) shows how the *curve processing* described by (4) is done in the case of a fixed priority scheduler.

Finally, we have to introduce a notion of pseudo-inverse functions (Le Boudec and Thiran, 2001). All arrival and service curves satisfy the following properties: $\phi(\Delta_1) \leq \phi(\Delta_2)$ iff $\Delta_1 < \Delta_2$, and $\phi(\Delta) = 0\ \forall \Delta \leq 0$, where ϕ denotes an arrival or a service curve. Hence, we define the pseudo-inverse of ϕ as follows:

$$\phi^{-1}(e) = \inf_{e \in \mathbb{Z}_{\geq 0}} \{\Delta : \phi(\Delta) \geq e,\ \Delta \in \mathbb{R}_{\geq 0}\} \tag{5}$$

Now, with this theoretical background we are ready to compute the minimum clock rate of the media processor satisfying a given delay constraint for the audio stream in our application scenario. First, we derive an analytical bound on the minimum clock rate, and after that, we explain how it can be computed in a practical setting.

4.3. Analytical Bound on the Minimum Clock Rate

Let D denote the delay constraint that we wish to satisfy for the audio stream. Then, using (3) we have to require that

$$D \geq \sup_{\Delta \in \mathbb{R}_{\geq 0}} \{\inf\{\tau \geq 0 : \bar{\alpha}_a(\Delta) \leq \bar{\beta}_a(\Delta + \tau)\}\} \tag{6}$$

where $\bar{\alpha}_a$ and $\bar{\beta}_a$ designate event-based arrival and service curves of the audio stream, respectively.

Let e denote a number of events. Then the delay constraint (6) can be expressed via pseudo-inverse functions of arrival and service curves

$$\bar{\beta}_a^{-1}(e) \leq D + \bar{\alpha}_a^{-1}(e), \quad \forall e \in \mathbb{Z}_{\geq 0} \tag{7}$$

Using definition (5) we can restate (7) as follows

$$\bar{\beta}_a\big(D + \bar{\alpha}_a^{-1}(e)\big) \geq e, \quad \forall e \in \mathbb{Z}_{\geq 0} \tag{8}$$

The constraint (8) is expressed in terms of event-based quantities. It says that in order to satisfy the delay constraint D for the audio stream, we have to ensure that at least e audio frames are completely processed within the time interval of length $D + \bar{\alpha}^{-1}(e)$. Since, we are interested in finding the minimum *clock* rate of the media processor, we need to translate (8) into a requirement expressed in number of clock cycles. We do it by using the upper workload curve γ_a^u of the audio stream:

$$\beta_a\big(D + \bar{\alpha}_a^{-1}(e)\big) \geq \gamma_a^u(e), \quad \forall e \in \mathbb{Z}_{\geq 0} \tag{9}$$

The right-hand side of (9) denotes the upper bound on the number of processor cycles that may be needed to completely process any number e of consecutive audio frames.

From the problem definition we know that the video encoding task has a higher priority than the audio task. Hence, the video stream can acquire the full processor capacity, while the audio stream can get only the service that has been left after the processing of the video stream. To find the remaining service for the audio stream we can use (4).

$$\beta_a(\tau) = \sup_{\forall \Delta \in [0,\tau]} \{\beta_v(\Delta) - \alpha_v(\Delta)\} \tag{10}$$

where β_v and α_v denote the service and arrival curves of the video stream, respectively.

Using (10) and by noting that the service curve corresponding to the full processor capacity is determined as $f\Delta$, where f denotes the processor clock rate, we can restate constraint (9) as follows.

$$\sup_{\forall\Delta\in[0,D+\bar{\alpha}_a^{-1}(e)]} \{f\Delta - \alpha_v(\Delta)\} \geq \gamma_a^u(e), \quad \forall e \in \mathbb{Z}_{\geq 0} \tag{11}$$

To satisfy (11) for a given value of e, it is sufficient to select f large enough such that there exists $\Delta' \in [0, D + \bar{\alpha}_a^{-1}(e)]$ for which

$$f(e)\Delta' \geq \alpha_v(\Delta') + \gamma_a^u(e)$$

holds true. Since we are looking for the minimum clock rate we can require that

$$f(e) \geq \inf_{\forall\Delta\in[0,D+\bar{\alpha}_a^{-1}(e)]} \left\{\frac{\alpha_v(\Delta) + \gamma_a^u(e)}{\Delta}\right\}, \quad \forall e \in \mathbb{Z}_{\geq 0} \tag{12}$$

To ensure that for any $e \in \mathbb{Z}_{\geq 0}$ the above constraint is satisfied we take the maximum value of all possible values of $f(e)$.

$$f_{\min} = \sup_{\forall e\in\mathbb{Z}_{\geq 0}} \left\{ \inf_{\forall\Delta\in[0,D+\bar{\alpha}_a^{-1}(e)]} \left\{\frac{\alpha_v(\Delta) + \gamma_a^u(e)}{\Delta}\right\}\right\} \tag{13}$$

or, equivalently,

$$f_{\min} = \sup_{\forall e\in\mathbb{Z}_{\geq 0}} \left\{ \inf_{\forall\Delta\in[0,D+\bar{\alpha}_a^{-1}(e)]} \left\{\frac{\gamma_v^u(\bar{\alpha}_v(\Delta)) + \gamma_a^u(e)}{\Delta}\right\}\right\} \tag{14}$$

Equation (14) gives the analytical bound on the minimum clock rate of the media processor required to satisfy the delay constraint D for the audio stream.

4.4. Computation of Minimum Clock Rate

To see the gain from the application of the type rate curves, we compute the minimum processor clock rate for two different cases. In one case we obtain the upper workload curve for the video encoding task γ_v^u by employing the type rate curves of the video stream. Like this we can take into account the per-frame execution demand variability of the video stream. In the other case, we use a conventional worst-case analysis approach: we get the workload curves under the pessimistic assumption that all frames within the video stream take the same (largest possible) WCET for their processing.

To compute γ_v^u from the type rate curves of the video stream, first we have to obtain the type rate curves themselves. To do this we apply the method described in Section 3.2 to the finite state machine of the MPEG-2 encoder shown in Figure 7. After that, we follow the method described in Section 3.3 to derive γ_v^u from the obtained type rate curves.

Figures 9 and 10 show results of these computation steps. We can see the obtained type rate curves in Figure 9 and the derived from them upper workload curve γ_v^u in Figure 10. For the computations we used system parameters as specified in Table 1.

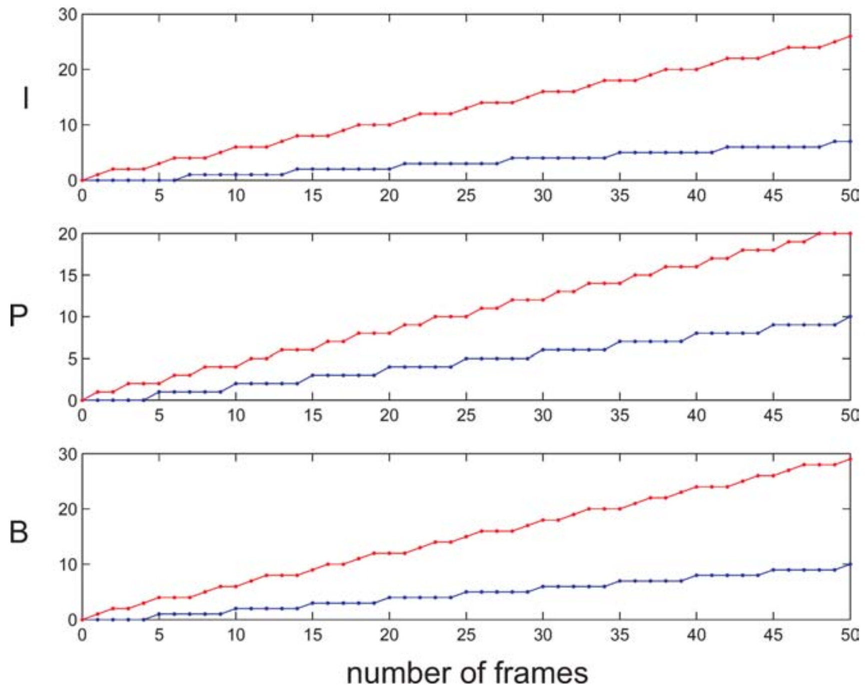

Figure 9. Type rate curves of the MPEG-2 video stream.

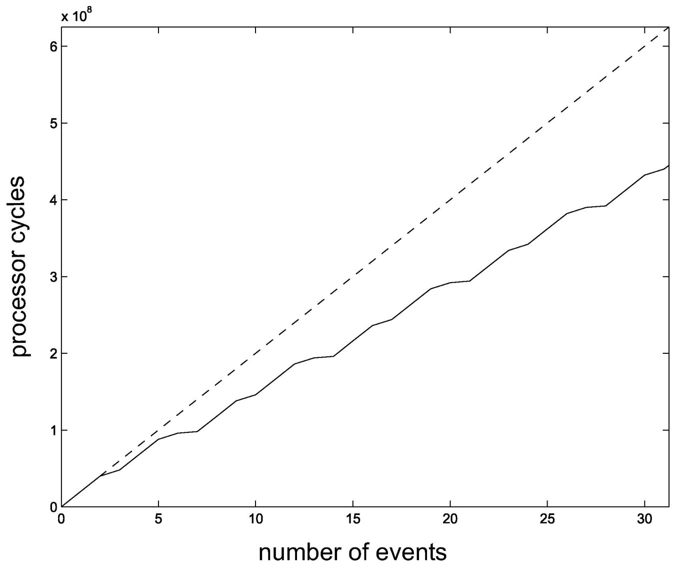

Figure 10. The upper workload curves of the video encoding task. The workload curve obtained using type rate curves is shown with the solid line.

Table 1. System parameters.

Parameter	Value
	Audio stream
r_a	44100/1152 fps
$wcet_a$	$5 \cdot 10^6$ cycles
	Video stream
r_v	25 fps
$wcet_I$	$2 \cdot 10^6$ cycles
$wcet_P$	$8 \cdot 10^6$ cycles
$wcet_B$	$20 \cdot 10^6$ cycles

For comparison in Figure 10 we also plot γ_v^u computed *without* using the type rate curves. In this case, the upper workload curve of the encoding task is determined as

$$\gamma_v^u(e) = e \cdot \max\{wcet_I, wcet_P, wcet_B\}$$

where $wcet_I$, $wcet_P$ and $wcet_B$ denote worst-case execution demands for I-, P- and B-frame types, respectively.

To be able to compute the minimum clock rate, besides the upper workload curve of the video encoding task γ_v^u we also need to know the arrival curves of the video ($\bar{\alpha}_v$) and audio ($\bar{\alpha}_a$) streams as well as the upper workload curve of the audio MP3 task γ_a^u.

To obtain the upper workload curve of the MP3 task γ_a^u, we assume that all audio frames are of the same type and, therefore, in the worst case, require the same amount of cycles $wcet_a$ to be processed. Thus, we can put $\gamma_a^u(e) = e wcet_a$.

According to our problem statement, the video and audio frames arrive with constant rates into the input FIFO buffers. Let r_v and r_a denote the video and audio input frame rates, correspondingly. Then, the respective arrival curves can be determined as $\bar{\alpha}_v(\Delta) = \lceil r_v \Delta \rceil$ and $\bar{\alpha}_a(\Delta) = \lceil r_a \Delta \rceil$. Note, that in general the shapes of the arrival curves may be more complex.

Finally, using Eq. (14) we can compute the minimum clock rate of the media processor that satisfies the given delay constraint D for the audio stream. Note, that in the theory it is necessary to evaluate equation (14) up to $e \to \infty$. This is to ensure that the value of the clock rate is high enough to sustain the overall average load imposed by the streams on the processor. However, computing up to $e \to \infty$ is not possible in practice.

We can overcome this problem by trading off the accuracy of computation to the computational time. We can evaluate (14) up to any given number of events $e_{\max}$, but we have to put an additional constraint on the clock rate:

$$f_{\min} \geq \frac{\left(\gamma_v^u\left(\bar{\alpha}_v\left(\bar{\alpha}_a^{-1}(e_{\max})\right)\right) + \gamma_a^u(e_{\max})\right)}{\bar{\alpha}_a^{-1}(e_{\max})} \tag{15}$$

With (15) we simply require that on some time interval of length $\bar{\alpha}_a^{-1}(e_{\max})$ the processor must provide the amount of service needed to fully process all video and audio frames that may arrive within that interval. If $e_{\max}$ is too small, then the constraint (15) may be very pessimistic but the evaluation time of (14) can be short. If $e_{\max}$ is large enough, then the right-hand side of (15) approaches the average case from above but the evaluation time may correspondingly increase.

4.5. Experimental Results

Using Eq. (14) in conjunction with the constraint (15) we computed the minimum clock rate of the media processor for a range of values of the delay constraint D and for the system parameters specified in Table 1. Figure 11 shows the results of these computations. The solid line was computed using type rate curves, while the dashed line shows the results obtained without type rate curves.

In Figure 12 we plot the gain resulted from the usage of the type rate curves. For different values of D, the plot shows the percentage by which the minimum clock rate computed using type rate curves is smaller than the minimum clock rate computed by the conventional approach (i.e. under the assumption that all video frames have the same execution requirement in the worst case). By inspecting the plot we can see that for large values of the audio delay constraint we can gain up to 20% of savings in the clock rate by applying the type rate curves for the analysis of the system.

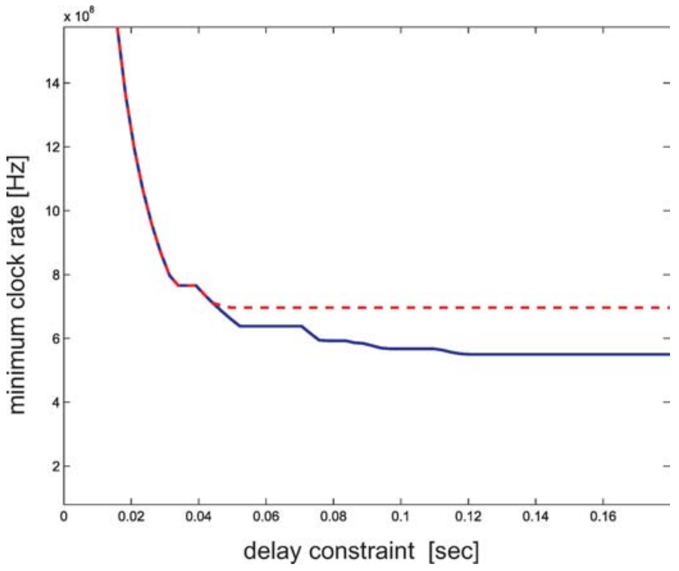

Figure 11. The minimum clock rate computed with type rate curves (solid line) and without using them (dashed line).

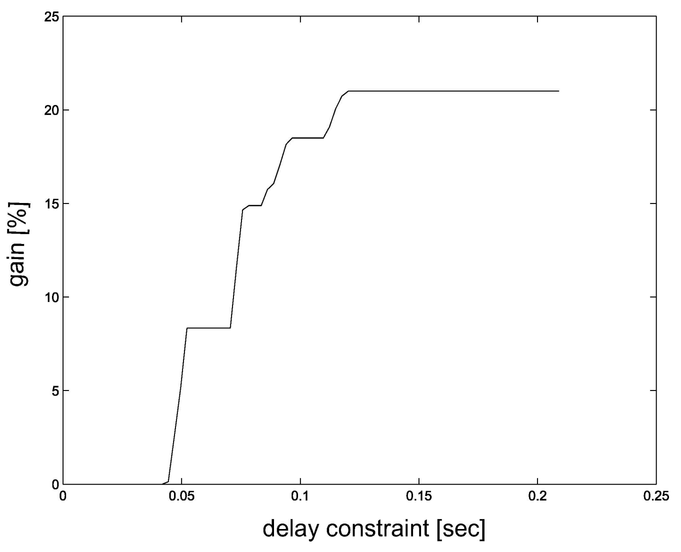

Figure 12. The gain obtained from using type rate curves.

5. Conclusions

In this paper, we presented a new abstract stream model to characterize event streams that are composed of a finite number of different event types. We further presented an analytic method to obtain these abstract stream models from a stream specification as well as a method to deduce an abstract workload characteristic that such a stream is imposing on a processing unit. We have shown the applicability of the presented methods in context with performance analysis of a multimedia application, where a considerable improvement of the worst-case performance bounds was achieved compared to conventional methods.

Note

1. International Standard Organization. Information technology—Generic Coding of Moving Pictures and Associated Audio Information—Part 2: Video, ISO/IEC 13818–2.

References

Balarin, F., Burch, J., Lavagno, L., Passerone, R., Sangiovanni-Vincentelli, A., and Watanabe, Y. 2001. Constraints specification at higher levels of abstraction. In *Proceedings of HLDVT 2001.*

Baruah, S. K. 2003. Dynamic- and static-priority scheduling of recurring real-time tasks. *Real-Time Systems* 24(1): 93–128.

Bernat, G., Colin, A., and Petters, S. M. 2002. WCET analysis of probabilistic hard real-time systems. In *Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS'02). IEEE Computer Society*, pp. 279–288.

Chakraborty, S., Künzli, S., and Thiele, L. 2003. A general framework for analysing system properties in platformbased embedded system designs. In *Proc. 6th Design, Automation and Test in Europe (DATE)*, Munich, Germany, pp. 190–195.

Cohen, G., Dubois, D., Quadrat, J. P., and Voit, M. 1985. A linear-system-theoretic view of discrete-event processes and its use for performance avaluation in manufacturing. *IEEE Transactions on Automatic Control*, AC-30(3): 210–220.

Cruz. R. 1991. A calculus for network delay. *IEEE Trans. Information Theory* 37(1): 114–141.

Hughes, C. J., Kaul, P., Adve, S. V., Jain, R., Park, C., and Srinivasan, J. 2001. Variability in the execution of multimedia applications and implications for architecture. In *Proceedings of the 28th International Symposium on Computer Architecture*, pp. 254–265.

Jersak, M., Henia, R., and Ernst, R. 2004. Context-aware performance analysis for efficient embedded systems design. In *Proc. 7th Design, Automation and Test in Europe (DATE).*

Karp, R. M. 1978. A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics* (23): 309–311.

Le Boudec J., and Thiran, P. 2001. *Network Calculus—A Theory of Deterministic Queuing Systems for the Internet*. LNCS 2050, Springer Verlag.

Lee, C., Potkonjak, M., and Mangione-Smith, W. H. 1997. Mediabench: A tool for evaluating and synthesizing multimedia and communicatons systems. In *International Symposium on Microarchitecture*, pp. 330–335.

Liu, C., and Layland, J. 1973. Scheduling algorithms for multiprogramming in hard real-time environment. *Journal of the ACM* 20(1): 46–61.

Maxiaguine, A., Künzli, S., and Thiele, L. 2004. Workload characterization model for tasks with variable execution demand. In *Proc. 7th Design, Automation and Test in Europe (DATE).*

Tia, T.-S., Deng, Z., Shankar, M., Storch, M., Sun, J., Wu, L.-C., and Liu, J. W.-S. 1995. Probabilistic performance guarantee for real-time tasks with varying computation times. In *Proceedings of the IEEE Real Time Technology and Applications Symposium*, IEEE Computer Society, pp. 164–173.

Ernesto Wandeler is a Ph.D. student at the Computer Engineering and Networks Laboratory of the Swiss Federal Institute of Technology, Zurich. His research interests include models, methods and tools for system-level performance analysis of real-time embedded systems. He holds a Dipl. El.-Ing. degree from ETH Zurich. In 2003, he received the "Willi Studer Price" and the "ETH Medal", both from the Swiss Federal Institute of Technology, Zurich. He is a student member of the IEEE and the ACM.

Alexander Maxiaguine is a Ph.D. student at the Computer Engineering and Networks Laboratory of the Swiss Federal Institute of Technology, Zurich. His research interests include models and methods for system-level performance analysis and scheduling of embedded multiprocessor architectures, especially for real-time multimedia applications. Maxiaguine has an M.S. in electrical engineering from the Moscow Technical University of Communications and Informatics. He is a member of the IEEE and the ACM.

Lothar Thiele is a full professor of computer engineering at the Swiss Federal Institute of Technology, Zurich. His research interests include models, methods and software tools for the design of embedded systems, embedded software and bioinspired optimization techniques. In 1986 he received the "Dissertation Award" of the Technical University of Munich, in 1987, the "Outstanding Young Author Award" of the IEEE Circuits and Systems Society, in 1988, the Browder J. Thompson Memorial Award of the IEEE, and in 2000–2001, the "IBM Faculty Partnership Award". In 2004, he joined the German Academy of Natural Scientists Leopoldina.