FIFO Scheduling and Event Count Curves for Modeling Structured Event Streams in Modular Performance Analysis

Simon Perathoner, Tobias Rein, Lothar Thiele, Kai Lampka
Computer Engineering and Networks Laboratory, ETH Zurich, Switzerland

Abstract

The growing complexity of distributed embedded real-time systems requires elaborate models and methods in their design process. While simulation does not sufficiently cover corner cases and may have excessive run times for complex systems, analytic methods for system level performance analysis have been established in the past. Their efficiency in computing hard bounds on buffer sizes, end-to-end delays or throughput has proven their usefulness. One of the major drawbacks of these methods is their limited scope in terms of system classes that can be analyzed with high accuracy. In this paper we extend existing methods for analyzing heterogeneous distributed embedded systems such that different types of data streams can be composed to a higher level event stream with multiple hierarchies. The method is based on a novel characterization of structured event streams, i.e., event streams for which the individual events belong to a finite number of classes. The new class of methods we provide can be embedded into well known compositional frameworks for performance analysis such as Symta/S and MPA (Modular Performance Analysis). We propose and compare two different approaches which take advantage of event stream structures. Realistic examples are given that are used to apply these new models and methods to performance analysis.

1. Introduction

1.1 Motivation

The design of stream-based distributed embedded systems poses tremendous difficulties in terms of efficiency and predictability. In particular, the various event streams that provide communication between the system components interact and interfere on joint resources such as buses, bridges, routers and communication networks. In a similar way, components of the application that are responsible for processing the data packets interfere in terms of their timing behavior on shared computing resources such as microprocessors, digital signal processors or dedicated computing components.

Accurate performance analysis is a central step in the design of distributed and parallel embedded systems. This fact is based on its role during design space exploration for early performance analysis as well as on its instrumental role for final system validation after the design is finished. It is widely acknowledged that models and methods for system-wide performance analysis need to satisfy a few requirements such as (a) coverage of possible system behaviors, (b) accuracy of the obtained results and (c) computational efficiency of the deployed evaluation tools.

Previous research developed a rich methodology for carrying out analytic performance analysis. Furthermore, the employed methods such as the Modular Performance Analysis (MPA) [3, 17], SymTA/S [6], MAST [9], and holistic methods [10, 11] resulted in a range of mature software tools for carrying out the analysis of system designs. As main feature, the developed workbenches allow to compute key characteristics like buffer sizes, event delays and resource utilization rates. In contrast to an empirical approach as provided by system simulation the analytic methodologies are exhaustive, i. e. they cover all possible system behaviors s. t. one obtains hard bounds on the above characteristics. Furthermore, some of the above mentioned approaches are compositional, hence they are computationally efficient, scale with system sizes and allow to analyze distributed heterogeneous embedded systems with a wide range of resource arbitration and scheduling principles.

However, one of the major drawbacks of the above mentioned analytic methods for performance analysis is their restricted modeling scope. Each of the methods has an associated model of computation which covers computation, resource sharing, communication among others. A loss in analysis accuracy can be expected if the system under analysis does not closely fit the class of models that is directly supported. As a result, there is a great interest in extending the underlying model of computation for each of the compositional methods.

In distributed embedded systems that involve complex communication systems such as networks or buses, one can very often find a highly relevant design pattern which (a) merges different event streams into a single one and (b) extracts different types of event streams from a single stream based on event type information. For example, data from different streams could be combined in packets and transmitted jointly over a communication channel. In [13], the combined data stream is denoted as hierarchical event stream (HES). Once a packet has been successfully delivered and arrives at the end-point of the communication channel, the data have to be de-packaged, i. e. the hierarchical event stream has to be decomposed into the individual data streams. Another related scenario is the simple merging of event or data streams from different sources without packaging, the joint processing or communication of the corresponding individual events or data packets and finally, the splitting of the joined data stream in its individual streams, see Fig. 1.1.

In this paper, we introduce a major extension to the aforementioned methods that allows to join and fork event streams with high accuracy. Furthermore, we keep the compositionality of the analysis methodology in contrast to all other known methods: The proposed extension based on Event Count Curves is completely transparent to all other analysis components, i.e. they can be used without any change and can ignore the fact that the modeled subsystem computes or communicates joined event streams. As a result, one of the main prerequisites for a compositional performance analysis is met.

1.2 Contributions and outline of the work

The paper contains the following new results:

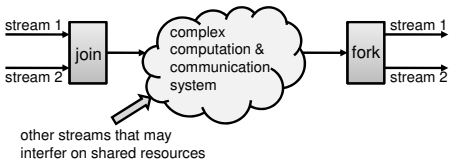


Figure 1. Joining and forking of streams.

- We propose two new approaches to analyze the processing and communication of joined event streams in modular system performance analysis. The first approach uses the principle of FIFO scheduling (first-in-first-out) to manage event hierarchies whereas the second approach uses Event Count Curves.
- The concept of Event Count Curves is shown to be orthogonal to previously used stream models such as PJD (periodic with jitter) and arrival curves. Therefore, compositionality of previous analysis methods is not affected as the new representation is transparent to all other analysis components.
- We extend the analysis method based on Event Count Curves in order to support *arbitrary* decompositions of streams.
- Experiments compare the different available methods (hierarchical event streams [13], FIFO scheduling, ECC) and a realistic application scenario is analyzed (heterogeneous communication system) in detail.

Organization of the paper: Section 2 discusses related work. In Section 3 we give details about the Modular Performance Analysis (MPA) framework [3, 15] adopted in this paper. We introduce a small application scenario in Section 4 which will be used to illustrate the proposed methods. In Sections 5 and 6 we present two different approaches to deal with structured event streams. The first approach is considering FIFO scheduling while the second uses a new concept denoted as Event Count Curves. Finally, in Section 7, we evaluate the proposed methods by means of a realistic case study and conclude our work in Section 8.

2. Related Work

Recently, Rox et al. [13] introduced the concept of Hierarchical Event Streams (HES) that allow to embed different types of streams in a higher level structure and proposed a Hierarchical Event Model (HEM) to represent such hierarchical streams. The key of the approach are the so-called inner event streams that are embedded in the HES bounding the event occurrences of events related to an individual input stream. When an operation is applied to the HES, i. e. the stream is processed in a task, an inner update function keeps track of the status of the inner event streams. Therefore, the HES can be decomposed into the individual streams whenever needed. The authors demonstrated significant improvements in terms of tightness of the worst case response time by using HEM instead of flat event stream models. However, the method is not transparent to the component models: The components need to explicitly handle the HES and their analysis requires a deep processing of HEM.

Albers [1] uses a hierarchical data structure to describe repetitively occurring patterns within event streams. Since such scenarios refer to the combination of streams, the authors also refer to their approach as hierarchical event stream. Nevertheless, the topic faced in the present work is not related to the work of [1], since it does not assume any pattern of event occurrence.

In this paper, we use the Modular Performance Analysis (MPA) based on Real-Time Calculus, as described in [3, 15, 17], as the basis for our analysis. With this compositional approach, a system is modeled as a set of components interconnected by streams representing the flow of events or resource units. However, contrary to other techniques, the streams are not defined on a straight time-line, but on the time-interval domain.

In the context of modular performance analysis methods, tasks activated by multiple inputs have been considered. In particular OR- and AND-activated tasks have been modeled, see [5, 8]. The OR-activation of tasks is closely related to the joining of event streams discussed in this work. But the results in [5, 8] do not allow to separate a joined stream again into its individual sub-streams. Timing correlations in the presence of simple split-join scenarios of event streams have been studied in [7, 14]. However, the present work is different, as we consider join and fork operations based on event types and focus on the structure of the joined streams.

FIFO scheduling has been considered in Network Calculus, see [2]. The results are related to the service that is given to individual streams that are processed in a FIFO order by some resource. These results closely relate to one of our approaches to model hierarchical event streams.

3. Analytic Real-time Performance Analysis

The authors of [3, 15] extend the classical Network Calculus [4] towards the analysis of embedded real-time systems. The obtained compositional methodology is denoted as Modular Performance Analysis (MPA) and allows the computation of upper and lower bounds on key performance metrics, such as delays, buffer sizes and resource utilizations.

In the following, we will briefly summarize some basic concepts of MPA that are necessary for the paper. The basic building blocks are analysis components which can be understood as abstract stream modifiers. They take streams of events and resource units and convert them into streams of remaining resource units and outgoing events, respectively. However, contrary to classical scheduling theory, the cumulative counting functions as employed in the MPA framework are defined on the time-interval domain, rather than on the straight time-line. Hence, the abstraction of event and resource streams κ can be described as a pair of cumulative counting functions denoted as the upper curve κ^u and the lower curve κ^l. Within the analysis, such pairs of curves bound the maximum and minimum number of events or resource units seen on a stream within ANY time interval of length $\Delta \in [0, \infty)$. Thus, a pair of upper and lower curves $\kappa(\Delta) := [\kappa^l(\Delta), \kappa^u(\Delta)]$ bounds a potentially infinite set of streams of events or available resource units according to

$$\kappa^l(t-s) \leq R(s,t) \leq \kappa^u(t-s) \tag{1}$$

with $s < t$ and $\kappa^l(0) = \kappa^u(0) = 0$. Here, the cumulative function $R(s,t)$ denotes the total number of events or available resource units that are present on a stream within the time interval $[s,t)$.

As pointed out above, analysis components can be interpreted as abstract stream converters which do not convert individual streams of events or resource units, but their abstract representations κ. For actually computing the output curves bounding the number of outgoing events or resource units with respect to intervals of length $\Delta = t - s$, it is now necessary to distinguish among arrival and service curves: an arrival curve $\alpha := (\alpha^l, \alpha^u)$ refers to the abstract description of event streams, whereas a service curve $\beta := (\beta^l, \beta^u)$ refers to the upper and lower bounding functions of available resource units. On the basis of min- and max-plus algebra it is now possible to compute the outputs α' and β' of an analysis component with event and resource input abstractions α and β, see e.g. [16]:

$$\begin{aligned}
\alpha^{u\prime} &= \min\{(\alpha^u \otimes \beta^u) \oslash \beta^l, \beta^u\} \\
\alpha^{l\prime} &= \min\{(\alpha^l \oslash \beta^u) \otimes \beta^l, \beta^l\} \\
\beta^{u\prime} &= (\beta^u - \alpha^l) \overline{\oslash} 0 \\
\beta^{l\prime} &= (\beta^l - \alpha^u) \overline{\otimes} 0
\end{aligned} \tag{2}$$

Here we use the abbreviations

$$\begin{aligned}(a \otimes b)(\Delta) &= \inf_{0\le\lambda\le\Delta}\{a(\Delta-\lambda)+b(\lambda)\}\\ (a \oslash b)(\Delta) &= \sup_{\lambda\ge 0}\{a(\Delta+\lambda)-b(\lambda)\}\\ (a \overline{\otimes} b)(\Delta) &= \sup_{0\le\lambda\le\Delta}\{a(\Delta-\lambda)+b(\lambda)\}\\ (a \overline{\oslash} b)(\Delta) &= \inf_{\lambda\ge 0}\{a(\Delta+\lambda)-b(\lambda)\}\end{aligned} \tag{3}$$

which are denoted as min-plus and max-plus convolution and deconvolution operators, respectively.

For the above equations we assume that a component operates greedily on incoming events, i. e. the progress in producing output events depends only on the availability of resource units. Waiting events are stored in an input queue. In other words, a greedy processing component (GPC) as described by (2) consumes as much of the available resource β as necessary for processing the waiting and newly appearing events of stream α. If the provided resources β are insufficient for processing the arriving events of stream α, additional backlog occurs and jobs are further delayed. If there are less events arriving than could be processed based on β, there are unused resource units β' available.

In this sense one may compute an upper bound δ on the delay experienced by an event that is processed by a GPC:

$$\delta = \sup_{\lambda\ge 0}\left\{\inf\{\tau \ge 0 \;:\; \alpha^u(\lambda) \le \beta^l(\lambda+\tau)\}\right\} \tag{4}$$

Graphically, δ is given as the largest horizontal distance between the upper arrival curve (maximally requested service) and the lower service curve (minimally offered service).

Based on the above concepts, one can describe a complex embedded real-time system as a network of GPCs interacting via abstract bounding functions α and β. The sharing of resources is modeled by connecting the respective GPCs via outgoing and ingoing service curves, where the order of the GPCs reflects the fixed priorities of the modeled processing tasks. Other scheduling schemes like TDMA can be modeled by adapting the ingoing service curves accordingly.

4. Motivational Example

In this section we introduce a simple application scenario of a multi-processor system in which different data flows are merged, processed, and separated again. The system will serve as example throughout the paper and will be used to illustrate the presented methods. In Section 6.5 we provide the analysis of the system and discuss the obtained results.

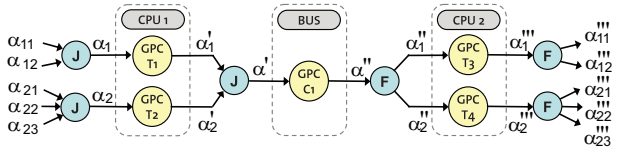


Figure 2. MPA model of example system

The MPA model of the considered system is shown in Figure 2. It contains various event streams that represent the flow of data through the system. For simplicity, in the figure we omit the resource streams of the model. The system contains several computation and communication tasks, represented as GPC components in the MPA model. It processes five input event streams α_{ij} that are joined (J) and forked (F) several times between the various processing components. For the joining of streams we assume 'OR' semantics. This means that the join operator produces one output event for each input event arriving on any of the inputs. We assume that the events of the joined stream are still distinguishable with respect to their provenance. In particular, we will talk about events 'of type e_x' to say that they origin from stream α_x and we will adopt this notation also in the reminder of the paper. Note that in the case of successive joins, the single events of the joined stream belong to several types. For instance, in the model shown in Figure 2, an event in the stream α' originating from stream α_{12} is of type e_1 but at the same time of type e_{12}. For the forking of streams we assume that every input event is forwarded to only one output stream, depending on its type. In the figure, the indices of the event streams denote the paths of the different event types. For instance, the streams represented by α_{21} and α'''_{21} contain events of the same type.

We assume that both CPU1 and CPU2 implement a preemptive fixed-priority scheduling policy, where task T1 has higher priority than task T2, and task T3 has higher priority than task T4. For simplicity, we assume that the inputs are periodic streams with jitter and that the tasks are characterized by best-case and worst-case execution times (BCET, WCET). The corresponding parameters can be found in Table 1. The goal of the analysis is to characterize the output event streams as precisely as possible.

Stream	Period	Jitter
α_{11}	100	30
α_{12}	90	15
α_{21}	30	0
α_{22}	80	20
α_{23}	75	5

Task	BCET	WCET
T1	2	3
T2	3	4
C1	9	12
T3	2	3
T4	4	5

Table 1. Parameters for the example system

Terminology: Note that in this paper we distinguish *simple* and *structured* event streams. The former term refers to an event stream in which all events have the same type. The latter is used for a stream in which the events have different types, i.e., belong to different sub-streams. In particular, a structured event stream results if we merge several simple (or structured) event streams. Note further, that in this paper we use the terms 'structured stream' and 'joined stream' interchangeably.

5. FIFO Scheduling

A first way to model the processing of joined event streams in system level performance analysis is to keep the individual substreams separated in the representation of the system and to adapt the existing models of the processing and communication components such that they can explicitly handle multiple input streams. To this end, we introduce a new component for the MPA framework that is based on FIFO scheduling. The use of such a component to represent the processing of a joined event stream is justified by the observation that the combination of two or more event streams preserves the arrival order of the individual events. A simple example of this modeling principle is shown in Figure 3. The left side of the figure shows the processing of a joined event stream by a task T. After the processing the stream is forked into its composing substreams. The right side of the figure shows the equivalent model making use of a FIFO component. The component represents the processing of two equivalent tasks T that are scheduled in FIFO order of their activations. The suggested modeling approach repre-

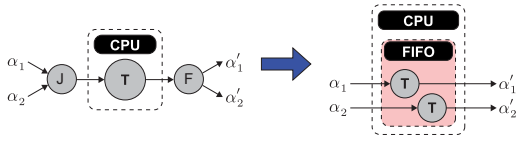


Figure 3. Processing of structured stream modeled by FIFO component

sents structured streams as bundles of distinct streams. Hence, each

time a system processes a structured stream, in the MPA model this has to be considered by abstracting the corresponding task by means of a FIFO component processing a bundle of streams. In the following, we describe how such an abstract FIFO component can be realized in the context of MPA.

5.1 The FIFO component

An abstract FIFO component as depicted in Figure 4(a) models a set of tasks that share an available resource in a FIFO manner. The activation patterns of the individual tasks are bounded by n arrival curves $\alpha_1, \cdots, \alpha_n$. The service curve β bounds the availability of processing resources for the component. The FIFO component computes n arrival curves $\alpha'_1, \cdots, \alpha'_n$ that characterize the outgoing event streams and a service curve β' that captures the amount of remaining processing resources.

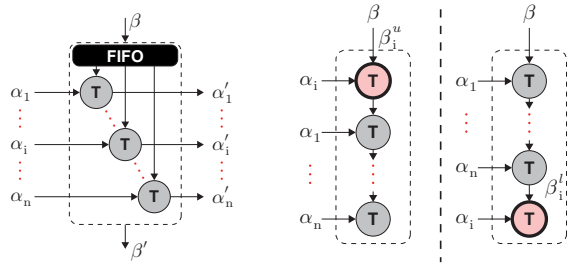
For the computation of β' we adapt the corresponding formulas of the GPC component reported in (2). In particular, as input event stream α we use the sum of all the input streams $\alpha_1, \cdots, \alpha_n$:

$$\beta'^u = (\beta^u - \sum_i \alpha_i^l) \overline{\oslash} 0 \quad ; \quad \beta'^l = (\beta^l - \sum_i \alpha_i^u) \overline{\otimes} 0 \qquad (5)$$

To determine valid upper and lower bounds for the outgoing event streams, we first look at the maximum and minimum possible amount of resources available for the processing of the corresponding input stream, respectively. In particular, for the task associated with input stream α_i we consider the best case and the worst case in a fixed assignment of priorities, as shown in Figure 4(b). The corresponding service curves are given by:

$$\beta_i^u = \beta^u \quad ; \quad \beta_i^l = (\beta^l - \sum_{j \neq i} \alpha_j^u) \overline{\otimes} 0 \qquad (6)$$

Under FIFO scheduling, the amount of resources available for



(a) Model of the FIFO Component in MPA

(b) Best-case (left) and worst-case (right) scenario for an event stream α_i under fixed priorities

Figure 4. Model of the FIFO Component

the processing of α_i can clearly only be less or equal than β_i^u and larger or equal than β_i^l and hence the two bounds are a valid abstraction. Given β_i^u and β_i^l, we can compute bounds for the corresponding outgoing event stream α'_i using the formulas for the GPC component as given in (2):

$$\begin{aligned} \alpha_i'^u &= \min\{(\alpha_i^u \otimes \beta_i^u) \oslash \beta_i^l, \beta_i^u\} \\ \alpha_i'^l &= \min\{(\alpha_i^l \oslash \beta_i^u) \otimes \beta_i^l, \beta_i^l\} \end{aligned} \qquad (7)$$

6. Event count curves

The approach presented in the previous section allows to model arbitrary compositions and decompositions of event streams. However, it has a major drawback. It is not transparent to the existing modeling components of the MPA framework. In particular, it requires to explicitly adapt all the components of a system model that process a structured event stream. In this section we present a second approach to handle structured event streams. The method is based on Event Count Curves (ECC) and is totally orthogonal to the modeling of the processing semantics of components. This means that modeled sub-systems can ignore the fact that they are processing structured event streams.

The method applies join and fork operations on event streams to compose and decompose structured streams. It relies on the fact that the order of the events in a stream is preserved no matter how many and what kind of components process the stream. Hence, when joining different event streams into one, we can store some information about the structure of the resulting stream and then use this information later, after arbitrary processing operations on the joined stream, to split it again into the corresponding sub-streams.

Obviously we cannot store the exact pattern of the composition whenever we join different event streams in a model. This would not only lead to an unbearable overhead for the composition and decomposition of event streams, but is also not feasible in the abstraction of MPA which operates in the time-interval domain. However, for any sequence of events in a joined stream, we can still bound the number of events belonging to a given sub-stream, which is the main idea behind ECCs.

6.1 Definitions

Definition 1 (Structured Event Stream)**.** *A structured event stream with event types e_i, $i \in I$ is described by a set of positive monotonically increasing arrival functions $R_i(s,t)$, $i \in I$, $s < t$. Given some times s and t, then $R_i(s,t)$ denotes the number of events of type e_i that arrived in the time interval $[s,t)$.*

In the context of MPA, we can characterize a structured event stream by a tuple of arrival curves bounding the total number of events in the stream, and by a tuple of ECCs for each of the composing sub-streams. Intuitively, an ECC bounds the number of events belonging to a particular sub-stream for a given number of consecutive events in the structured stream.

Definition 2 (Characterization of structured event streams)**.** *A structured event stream can be characterized by a tuple of upper and lower arrival curves $\alpha(\Delta) = [\alpha^l(\Delta), \alpha^u(\Delta)]$, $\Delta \geq 0$ and a set of upper and lower ECCs $\gamma_i(n) = [\gamma_i^l(n), \gamma_i^u(n)]$, $n \geq 0$, one for each event type e_i, $i \in I$. In particular, the upper and lower arrival curves bound the number of events in any time interval, i. e. for all $s < t$, $i \in I$ we have*

$$\alpha^l(t-s) \leq \sum_{i \in I} R_i(s,t) \leq \alpha^u(t-s) \qquad (8)$$

The upper and lower ECCs bound the number of events of type e_i within a certain number of events, i. e. for all $s < t$, $i \in I$ we have

$$\gamma_i^l(\sum_{j \in I} R_j(s,t)) \leq R_i(s,t) \leq \gamma_i^u(\sum_{j \in I} R_j(s,t)) \qquad (9)$$

For the single sub-streams we can again define arrival curves that bound the number of events in any time interval:

Definition 3 (Arrival curves for sub-streams)**.** *For each event type e_i, $i \in I$, of a structured event stream the arrival curves $\alpha_i(\Delta) = [\alpha_i^l(\Delta), \alpha_i^u(\Delta)]$, $\Delta \geq 0$, satisfy*

$$\alpha_i^l(t-s) \leq R_i(s,t) \leq \alpha_i^u(t-s) \qquad (10)$$

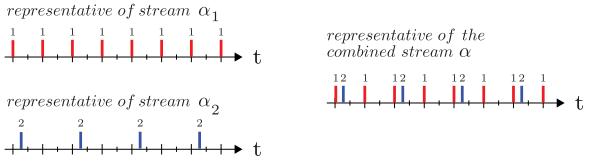
for all $s < t$.

Let us now illustrate the concept of ECCs by means of a simple example.

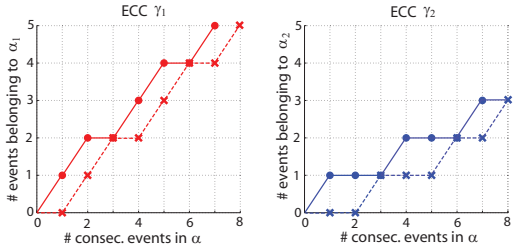
Example 1. *Consider two strict periodical event streams α_1 and α_2 with periods $p1 = 10$ and $p2 = 20$, respectively. Two example*

traces for the two streams are shown on the left side of Figure 5(a). Consider now the structured stream α obtained by joining α_1 and α_2. A representative trace for α is shown on the right side of Figure 5(a). The ECCs γ_1 and γ_2 describing the structure of the joined stream α are depicted in Figure 5(b). For any number of consecutive events in α, γ_1 and γ_2 give upper and lower bounds on the number of possible occurrences of events of type 1 and 2, respectively. For instance, for 4 consecutive events in the structured stream α, at least 2 and at most 3 events are of type 1, or in short, $\gamma_1^l(4) = 2$ and $\gamma_1^u(4) = 3$.

Note that ECCs are defined for integer values only. In the representation of Figure 5(b) the interconnecting lines are shown for illustration purposes only.



(a) Traces used in Example 1.



(b) ECCs for Example 1

Figure 5. Example traces and ECCs of Ex. 1

In the following we will also need the concept of pseudo-inverses of arrival curves and ECCs.

Definition 4 (Pseudo-inverse of arrival curve)**.** *The pseudo-inverses of upper and lower arrival curves $\alpha(\Delta) = [\alpha^l(\Delta), \alpha^u(\Delta)]$ are defined as*

$$\begin{aligned} \alpha^{-l}(n) &= \sup\{\Delta \geq 0 : \alpha^l(\Delta) \leq n\} \\ \alpha^{-u}(n) &= \inf\{\Delta \geq 0 : \alpha^u(\Delta) \geq n\} \end{aligned} \tag{11}$$

Lower and upper arrival curves $\alpha^l(\Delta)$ and $\alpha^u(\Delta)$ denote the minimum and maximum number of events that may arrive in a stream in any time interval of length $\Delta \in \mathbb{R}^+$, respectively. Their pseudo-inverses have the following meaning: $\alpha^{-l}(n)$ denotes the length of the longest time interval in which there can be n event arrivals in the stream; $\alpha^{-u}(n)$ denotes the length of the shortest time interval with n event arrivals.

Definition 5 (Pseudo-inverse of ECC)**.** *The pseudo-inverses of upper and lower ECCs $\gamma_i(n) = [\gamma_i^l(n), \gamma_i^u(n)]$ are defined as*

$$\begin{aligned} \gamma^{-l}(n_i) &= \sup\{n \geq 0 : \gamma_i^l(n) \leq n_i\} \\ \gamma^{-u}(n_i) &= \inf\{n \geq 0 : \gamma_i^u(n) \geq n_i\} \end{aligned} \tag{12}$$

Lower and upper ECCs $\gamma_i^l(n)$ and $\gamma_i^u(n)$ denote the minimum and maximum number of events of type e_i in any sequence of $n \in \mathbb{N}$ events of the structured stream, respectively. Their pseudo-inverses are interpreted as follows: $\gamma_i^{-l}(n_i)$ denotes the maximum length of an event sequence that contains n_i events of type e_i; $\gamma_i^{-u}(n_i)$ denotes the minimum length of a sequence with n_i events of type e_i.

6.2 Join and Fork of Simple Event Streams

In the following, we show how ECCs can be computed when joining streams, and how they can be applied to split structured streams into sub-streams.

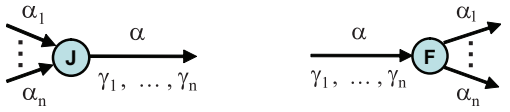


Figure 6. (a) Join operator for merging n simple event streams into one structured stream. (b) Fork operator for decomposing a structured event stream into n sub-streams.

Consider first the fork operator shown in Figure 6(b). Given the structured event stream and the various ECCs, we can derive the individual sub-streams as follows.

Theorem 1. *Given is a structured event stream with arrival curve α and ECCs γ_i, $i \in I$. Then we have*

$$\alpha_i^l(\Delta) = \gamma_i^l(\alpha^l(\Delta)) \quad ; \quad \alpha_i^u(\Delta) = \gamma_i^u(\alpha^u(\Delta)) \tag{13}$$

Proof (Sketch): The evaluation of $\alpha^l(\Delta)$ gives the minimum number n of events in the structured stream for a given interval Δ. This number can be translated using γ_i^l which by definition determines the minimum number of events of type e_i in a sequence of n events in the structured stream. The proof for $\alpha^u(\Delta)$ is analogous. □

Consider now the join operator shown in Figure 6(a). Given the individual streams $\alpha_1, \cdots, \alpha_n$ we can compute the resulting structured event stream and the individual ECCs as follows.

Theorem 2. *Given are n event streams with arrival curves $\alpha_1, \cdots, \alpha_n$ that are joined to a single event stream. Then the resulting structured event stream is characterized by the arrival curve*

$$\alpha(\Delta) = [\alpha^l(\Delta), \alpha^u(\Delta)] = \Big[\sum_i \alpha_i^l, \sum_i \alpha_i^u\Big]. \tag{14}$$

The ECCs of the structured event stream are determined by

$$\gamma_i(n) = [\gamma_i^l(n), \gamma_i^u(n)] = [\epsilon_i^{-u}(n), \epsilon_i^{-l}(n)] \tag{15}$$

with

$$\begin{aligned} \epsilon_i^u(n_i) &= n_i + \textstyle\sum_{j \neq i} \alpha_j^u(\alpha_i^{-l}(n_i)) \\ \epsilon_i^l(n_i) &= n_i + \textstyle\sum_{j \neq i} \alpha_j^l(\alpha_i^{-u}(n_i)) \end{aligned} \tag{16}$$

Proof (Sketch): The join operator forwards the events of all input streams without delay. Hence, for any time interval we have that the number of event arrivals in the structured stream is equal to the sum of the event arrivals in the individual sub-streams, which justifies (14). Formally, we can prove (14) by combining the inequalities (8) and (10).
For the computation of the ECCs, let us focus on $\gamma_i^l(n)$. The curve $\gamma_i^l(n)$ represents the minimum number n_i of events of type e_i in a sequence of n events of the structured stream. Consider now the pseudo-inverse of $\gamma_i^l(n)$ which we denote as $\epsilon_i^u(n_i)$. The curve $\epsilon_i^u(n_i)$ represents the maximum length n of an event sequence that contains n_i events of type e_i. Intuitively, we can determine $\epsilon_i^u(n_i)$ by constructing a scenario in which during a time interval there are n_i event arrivals in sub-stream α_i and as many event arrivals as possible in all the other sub-streams α_j with $j \neq i$. This is done by first considering the largest time interval Δ for which in sub-stream α_i there can be exactly n_i event arrivals, given by $\Delta = \alpha_i^{-l}(n_i)$. Successively, we determine the maximum number of event arrivals

in all the other sub-streams in a time interval of length Δ. This is done by evaluating the sum of the corresponding upper arrival curves α_j^u. At this point we can easily determine $\epsilon_i^u(n_i)$, and hence, $\gamma_i^l(n)$. The derivation of $\gamma_i^u(n)$ is analogous. □

Note that in terms of $\alpha(\Delta)$, the above described join operator is equivalent to the OR-composition of the input streams $\alpha_1, \cdots, \alpha_n$, as introduced in [5]. For a more detailed proof of the bounds $\gamma_i(n)$ we refer the reader to [12].

6.3 Hierarchical application of ECCs

In the presence of multiple successive join and fork operators, we can organize and apply ECCs in an hierarchical manner. Consider for instance the example system of Figure 2. We can model the joins and forks of streams in the system by the operators introduced in Section 6.2. In particular, by joining the streams α_{11} and α_{12} we will obtain the structured stream α_1 and two ECCs γ_{11} and γ_{12}. Similarly, the join of α_{21}, α_{22} and α_{23} will result in the structured stream α_2 and the ECCs γ_{21}, γ_{22} and γ_{23}. The processed event streams α_1' and α_2' are then joined again which yields the structured stream α' and two ECCs γ_1 and γ_2. Following the various stream compositions, we can organize the ECCs that describe the structure of the stream α' with respect to the different event types hierarchically. In particular, we can represent the hierarchy of event types by means of a tree, as shown in Figure 7. The edges of the tree represent ECCs that are applied to extract sub-streams from a structured stream

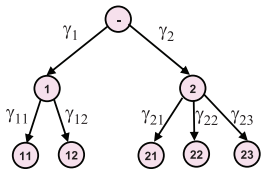


Figure 7. ECC hierarchy for the system of Figure 2

In the model of Figure 2, we can apply the described hierarchy of ECCs to fork the stream α'' into its composing sub-streams. In particular, we first apply γ_1 and γ_2 to fork α'' into α_{21}'' and α_{22}''. Successively, we fork α_{21}'' and α_{22}'' again by applying γ_{11}, γ_{12}, γ_{21}, γ_{22}, and γ_{23}.

6.4 Arbitrary Join and Fork of Structured Event Streams

The hierarchical organization of ECCs described above has a major disadvantage: The resulting structured stream can be decomposed only in the inverse order in which it has been composed. However, in order to enable the modeling of realistic systems, it is highly desirable to provide join/fork operators that allow an *arbitrary* decomposition of a structured stream into sub-streams, no matter how the structured stream was constructed. Consider again the motivational example introduced in Section 4. Assume that in the system the division of the event stream α'' is different from the one depicted in Figure 2. For instance, require that stream α_{21} needs to be processed by T3 instead of T4, that is, the composition and decomposition of the streams are not symmetrical. In such a case, with the ECC model described so far, we have no means to correctly abstract the system behavior.

In this section we tackle the above described problem and introduce more general join and fork operators that operate on *structured* event streams. The new operators allow to arbitrarily join and fork structured event streams by keeping the ECC hierarchy flat. That is to say that a structured stream is characterized only by ECCs referring to its *simple* sub-streams. ECCs referring to *structured* sub-streams will not be computed and forwarded any longer.

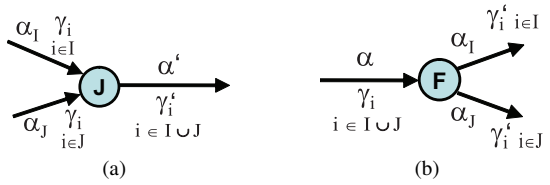


Figure 8. (a) Join operator for merging a two structured event streams. (b) Fork operator for decomposing a structured event stream into two structured sub-streams.

Consider the join operator shown in Figure 8(a). It merges two *structured* event streams α_I and α_J. In contrast to the case of a hierarchical organization of ECCs, this join operator computes a new ECC for all *simple* sub-streams composing the outgoing structured event stream. Hence, at any further point in the model it will be possible to isolate arbitrary subsets of the joined event streams. The outgoing structured event stream of the described join operator can be characterized as follows.

Theorem 3. *Assume that we join two structured event streams with arrival curves α_I and α_J and ECCs γ_i, $i \in I$ and γ_i, $i \in J$, respectively. Then the resulting structured event stream is characterized by the arrival curve*

$$\alpha'(\Delta) = [\alpha'^l(\Delta), \alpha'^u(\Delta)] = [\alpha_I^l(\Delta) + \alpha_J^l(\Delta), \alpha_I^u(\Delta) + \alpha_J^u(\Delta)] \tag{17}$$

The ECCs of the structured stream are determined by

$$\gamma_i'(n) = [\gamma_i'^l(n), \gamma_i'^u(n)] = [\gamma_i^l(\gamma_I^l(n)), \gamma_i^u(\gamma_I^u(n))] \tag{18}$$

for $i \in I$. Here, we use the partial ECCs for stream I

$$\gamma_I^l(n) = \epsilon_I^{-u}(n) \quad \gamma_I^u(n) = \epsilon_I^{-l}(n) \tag{19}$$

with

$$\epsilon_I^u(n_I) = n_I + \alpha_J^u(\alpha_I^{-l}(n_I)) \; ; \; \epsilon_I^l(n_I) = n_I + \alpha_J^l(\alpha_I^{-u}(n_I)) \tag{20}$$

The ECCs $\gamma_i'(n)$ for $i \in J$ are determined similarly.

Proof (Sketch): We have again that for any time interval the number of event arrivals in the joined output stream is equal to the sum of the event arrivals in the two input streams, which justifies (17).
For the outgoing ECCs, let us focus on $\gamma_i'^l(n)$ with $i \in I$. We first ignore that α_I and α_J are structured streams, and apply the join operator described in Theorem 2. This yields the ECC $\gamma_I^l(n)$ that defines the minimum number of events that belong to stream α_I in a sequence of n events of the structured stream α'. At this point we consider that α_I is a structured stream itself. Hence, given a sequence of events of α_I, by applying the ECC γ_i^l we can determine the minimum number of events of type e_i in the sequence. This leads to $\gamma_i'^l(n) = \gamma_i^l(\gamma_I^l(n))$. The derivation of $\gamma_i'^u(n)$ is analogous. □

The above theorem can easily be extended to more than two inputs.

Finally, consider the fork operator shown in Figure 8(b), which splits a structured event stream into two *structured* event streams α_I and α_J. The outgoing structured event streams can be characterized as follows.

Theorem 4. *Given is a structured event stream with arrival curve $\alpha(\Delta)$ and ECCs $\gamma_i(n)$, $i \in I \cup J$. Then the arrival curve $\alpha_I(\Delta) = [\alpha_I^l(\Delta), \alpha_I^u(\Delta)]$ of a sub-stream I with event types e_i, $i \in I$, is characterized by by the bounds*

$$\alpha_I^l(\Delta) \geq \sum_{i \in I} \gamma_i^l(\alpha^l(\Delta)) \tag{21}$$

$$\alpha_I^l(\Delta) \geq \max\left\{ \inf_{\lambda \geq \Delta} \left\{ \alpha^l(\lambda) - \sum_{i \in J} \gamma_i^u(\alpha^u(\lambda)) \right\}, 0 \right\} \quad (22)$$

and

$$\alpha_I^u(\Delta) \leq \sum_{i \in I} \gamma_i^u(\alpha^u(\Delta)) \quad (23)$$

$$\alpha_I^u(\Delta) \leq \sup_{0 \leq \lambda \leq \Delta} \left\{ \alpha^u(\lambda) - \sum_{i \in J} \gamma_i^l(\alpha^l(\lambda)) \right\} \quad (24)$$

The arrival curve $\alpha_J(\Delta)$ is derived in analogous manner.
The ECCs $\gamma'_i(n) = [\gamma'^l_i(n), \gamma'^u_i(n)]$ with $i \in I$ are characterized by the bounds

$$\gamma'^l_i(n) \geq g_i^{-u}(n) \qquad \gamma'^l_i(n) \geq f_i^{-u}(n) \quad (25)$$

$$\gamma'^u_i(n) \leq g_i^{-l}(n) \qquad \gamma'^u_i(n) \leq f_i^{-l}(n) \quad (26)$$

with

$$g_i^u(n_i) = n_i + \sum_{k \in I \setminus \{i\}} \gamma_k^u(\gamma_i^{-l}(n_i)) \quad (27)$$

$$f_i^u(n_i) = \sup_{0 \leq \lambda \leq n_i} \left\{ \gamma_i^{-l}(\lambda) - \sum_{k \in J} \gamma_k^l(\gamma_i^{-u}(\lambda)) \right\} \quad (28)$$

$$g_i^l(n_i) = n_i + \sum_{k \in I \setminus \{i\}} \gamma_k^l(\gamma_i^{-u}(n_i)) \quad (29)$$

$$f_i^l(n_i) = \max\left\{ \inf_{\lambda \geq n_i} \left\{ \gamma_i^{-u}(\lambda) - \sum_{k \in J} \gamma_k^u(\gamma_i^{-l}(\lambda)) \right\}, 0 \right\} \quad (30)$$

The ECCs $\gamma'_i(n)$ with $i \in J$ are characterized in analogous manner.

Due to space constraints, we do not prove the above theorem. We will, however, illustrate the idea behind the above formulae. Let us start with the bounds for $\alpha_I^l(\Delta)$. The curve $\alpha_I^l(\Delta)$ represents the minimum number of events that are seen on the upper output stream in any interval of length Δ. In order to compute $\alpha_I^l(\Delta)$, we consider $\alpha^l(\Delta)$, the minimum number of events on the input stream, and determine how many of those events do certainly belong to α_I. This can be done in two ways. We can either use the ECCs γ_i^l, $i \in I$, to determine how many events do certainly belong to the individual sub-streams of α_I and then compute their sum, as shown in (21). Alternatively, we can subtract from $\alpha^l(\Delta)$ the maximum number of events that can possibly belong to α_J, as shown in (22). For $\alpha_I^u(\Delta)$ the reasoning is analogous. Note that when we compute the difference of arrival curves, we have to use appropriate sup or inf operators to guarantee the monotonicity of the resulting curve.

Let us now consider the bounds for the outgoing ECCs. The approach for the computation of $\gamma'^l_i(n)$ relies on the same idea as adopted in Theorem 2, namely by deriving a bound for the pseudo-inverse of the unknown curve. In this case, the pseudo-inverse of $\gamma'^l_i(n)$ represents the maximum length n of an event sequence in α_I that contains n_i events of type e_i. In order to bound it, we first consider $\gamma_i^{-l}(n_i)$, the maximum length of a sequence on the input stream α that contains n_i events of type e_i, and then determine how many of those events can at most belong to α_I. This can again be done in two ways: Using the ECCs γ_k^u, $k \in I \setminus \{i\}$, as shown in (27), or using the ECCs γ_k^l, $k \in J$, as shown in (28). The ECCs $\gamma'^u_i(n)$ are determined similarly.

6.5 Analysis of Motivational Example

In this section we compare the analysis results of the different proposed approaches for the example system introduced in Section 4. We first build three new MPA models for the system of Figure 2. In the first one we apply the method described in Section 5, that is, we use FIFO components with multiple inputs instead of the depicted GPC components. In the second model, we adopt hierarchies of ECCs as described in Section 6.3. In particular, the tree of Figure 7 is used to model the structure of the streams α' and α''. The third model is based on a flat organization of ECCs as described in Section 6.4. Thus, it contains two ECCs for α_1 and α'_1, five ECCs for α' and α'', etc.

Successively, we use the models to bound the five output streams of the system. The Figures 9 and 10 depict the results achieved for the output curves α'''_{12} and α'''_{21}. The figures show that the two approaches based on FIFO scheduling and on hierarchically organized ECCs provide similar results. For the output stream α'''_{12} the FIFO approach determines tighter (i.e., less conservative) bounds. For the output stream α'''_{21} the approach based on hierarchical ECCs is tighter. The method based on the flat organization of ECCs provides the worst bounds for both output streams, with local exceptions as α'''^u_{21} shows. However, as pointed out earlier, it is the only method applicable in scenarios with unsymmetrical join and fork operations (which is not the case in the described example) and hence it is still highly useful in general.

We also compare the results of the proposed approaches with the bounds obtained when applying the method described in [13]. The figures show that the HEM of [13] provide slightly better bounds than the proposed approaches. However, it has to be noted that, in contrast to the HEM of [13], the approaches based on ECCs do not affect the compositionality of previous analysis methods and are totally transparent to the modeling of processing components.

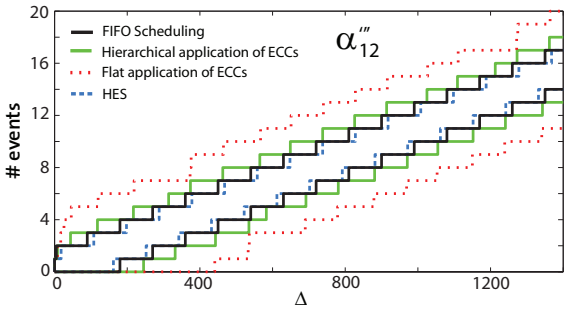


Figure 9. Results for the characterization of stream α'''_{12}

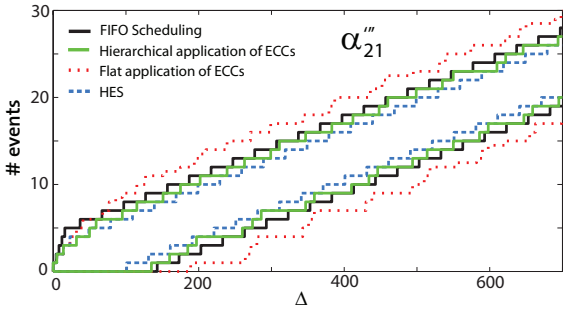


Figure 10. Results for the characterization of stream α'''_{21}

7. Case Study

In this section we show how the proposed theory can be applied to the analysis of a realistic distributed embedded system. We consider a heterogeneous avionic in-cabin entertainment/communication system which we denote shortly as HCS.

General System Description

The system consists of various devices connected by a communication network. In particular, the HCS comprises a central server (SERV), an Ethernet backbone network, and a large number of end-devices (DEV). We assume that there are n network controllers (NC) connected in a chain along the backbone network and that m distinct end-devices are connected to each NC. Figure 11 shows the architecture of the HCS for the case $n = 3, m = 3$.

We assume that there is an end-device for each seat in the aircraft cabin. The main function of the end-devices is the playback of audio streams transmitted over the backbone network from the server. We consider an on-demand audio system where each passenger can choose its individual audio content from a large database stored on the server. Hence we assume that the audio streams are transmitted in unicast mode, meaning that for each end-device there is a dedicated data stream from the server to the device. Besides on-demand audio streaming, the server executes a second application, namely the periodic broadcast of real-time flight data over the backbone network. We assume that there are a number of LCDs connected to some of the NCs in the system, displaying the transmitted flight data such as altitude, speed and current position on a map.

In this case study we focus on the *communication* between the components of the HCS. In particular, we are going to analyze the timing of the data transmissions over the links of the described network. We will neglect the *computations* carried out by the components themselves, i.e., we will not consider the execution times of the various processes on the server and the end-devices.

We consider a scenario in which the HCS provides different QoS for the two different kinds of traffic (audio streams and flight data). In particular, we assume that for the transmission of frames over the outgoing links both, the server and the NCs, implement a preemptive fixed-priority arbitration policy where audio traffic has higher priority than flight data traffic. This means that an ongoing transmission of flight data over a link will be interrupted whenever there is an audio frame to be transmitted over the same link. The interrupted transmission will be resumed as soon as the link is free again. For the sake of simplicity, we assume that the transmission of frames can be interrupted and resumed at any time without need of retransmitting previously send data.

The goal of the analysis is to determine whether the frames with flight data will reach their destination within a given deadline under the described arbitration policy. To keep the illustration of the proposed methods simple, we will restrict ourselves to the analysis of the small system architecture shown in Figure 11. In order to still show meaningful effects, we choose an accordingly small bandwidth for the Ethernet backbone network.

Detailed specification

We consider a full duplex Ethernet backbone network with a bandwidth of 5 Mbit/s. We are interested in the traffic from the server to the end-devices only. We assume that the server sends an individual audio stream to each of the nine devices shown in Figure 11. Without loss of generality we index the audio streams with the number of the corresponding destination device. We consider audio streams with a net data rate of 384 kbit/s. The server partitions the data of each audio stream in frames of constant size. The audio frames are send to the network periodically with a small jitter. The complete specification of the audio traffic in the network is given in the upper part of Table 2.

The server also periodically sends flight data to the LCD. We assume that all the needed data is transmitted in a single frame of constant size, which we shortly denote as data frame. The details for the transmission of data frames are specified in Table 2. The aim of the analysis is to clarify whether each data frame is guaranteed to reach its destination within the specified deadline.

	Size	Period	Jitter	Deadline
Audio Frames	1'518 Bytes	30 ms	5 ms	100 ms
Data Frames	106'500 Bytes	5 s	0 s	1.5 s

Table 2. Characteristics of the network traffic

Models and Analysis

In the following we are going to describe how the specified system can be modeled using the different methods proposed in this paper to handle join/fork scenarios of event streams. In order to obtain a baseline for the comparison of the new methods, we also build a rough model of the system behavior using the classic MPA abstraction, where join/fork scenarios cannot be faithfully represented.

The common basic abstraction of the models is to represent the frame traffic in the network by means of timed event streams. This allows to model communication components of the network such as data links by means of abstract components that process event streams. Figure 16 illustrates the basic modeling principle. The abstract processing components are triggered by incoming events which represent frames that need to be transmitted over the corresponding link. The processing time of an event in the abstract component corresponds to the transmission time of the frame in the concrete network component. The completion of a frame transmission is represented by the generation of an event on the output of the abstract component.

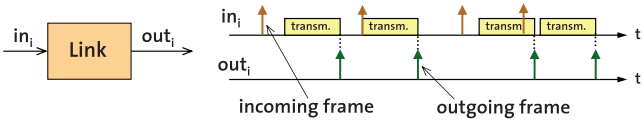


Figure 16. Frame transmission modeled as processing of timed event streams

In the MPA modeling framework event streams on the conventional time-line are abstracted by arrival curves in the time-interval domain. Hence, in the MPA models of the HCS we have to capture the timing behavior of all streams by means of appropriate arrival curves. In particular, exploiting the periodic nature of the streams, we can use the equations $\alpha^u(\Delta) = \left\lceil \frac{\Delta+\text{jitter}}{\text{period}} \right\rceil$, $\alpha^l(\Delta) = \left\lfloor \frac{\Delta-\text{jitter}}{\text{period}} \right\rfloor$ to the determine nine arrival curves $\alpha_1, \cdots, \alpha_9$ which represent the audio streams produced by the server. Similarly, we model the Data Stream by an arrival curve α_{data}. We use three service curves $\beta_1, \beta_2, \beta_3$ to represent the full availability of the three network links for the two considered traffic classes (audio streams and data stream). The fixed-priority arbitration policy described above is modeled by appropriately connecting the service curve inputs and outputs (β, β') of the abstract event processing components, as shown in Figure 12. The transmission times for the corresponding frames on the network links are derived from the frame sizes and the network bandwidth. They amount to 2.429 ms for audio frames and 170.4 ms for data frames.

Figure 12 shows a model of the HCS in the classic MPA framework, that is, without the methods introduced in this paper. The joining of the nine audio streams into a single stream is modeled by a simple sum of the corresponding arrival curves, which is a correct representation for the traffic sent over link 1. However, in contrast to the newly proposed methods, there is no means to decompose the resulting stream again into sub-streams. We can only forward the joined event stream to other components. Since in this model on the links 2 and 3 we represent more audio traffic than

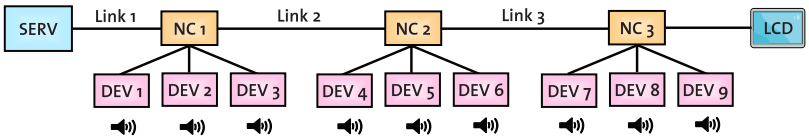


Figure 11. Application scenario

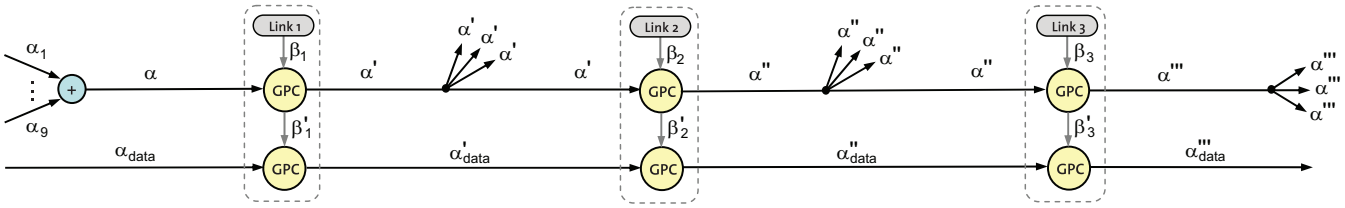


Figure 12. Model of communication system in classic MPA

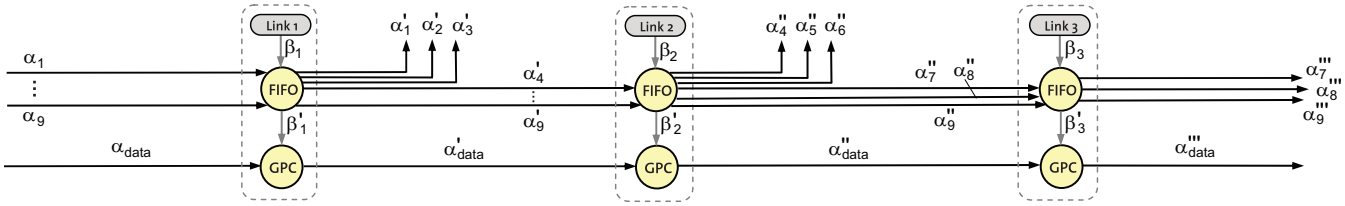


Figure 13. Model of communication system in MPA with FIFO components

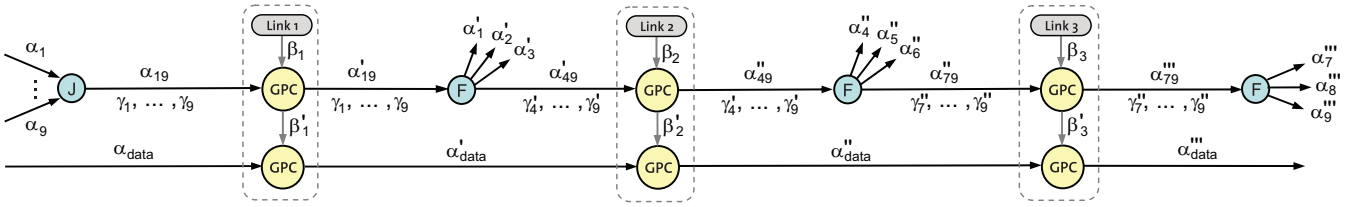


Figure 14. Model of communication system in MPA with ECCs (flat organization)

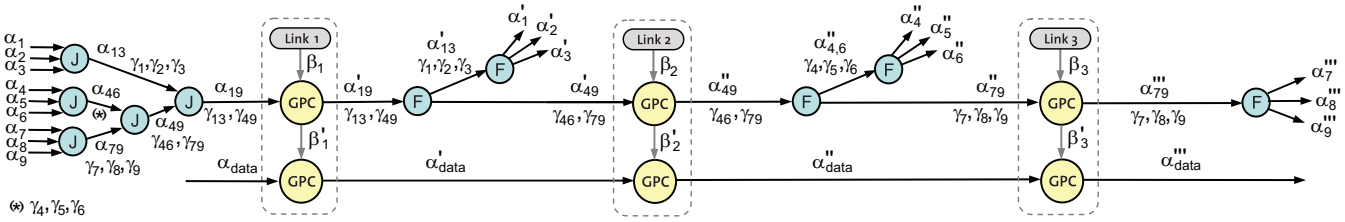


Figure 15. Model of communication system in MPA with ECCs (hierarchical organization)

actually present in the real system, we expect overly conservative results for the worst-case end-to-end delay of data frames.

In Figure 13 we report the MPA model that makes use of the FIFO scheduling component as introduced in section 5. The models of the HCS that employ ECCs are shown in Figures 14 and 15. In the model of Figure 14 we consider a flat organization of the ECCs. Hence, whenever we have to branch off sub-streams from a structured stream, this can be done with one single fork operator as shown in the figure. In contrast, in the model of Figure 15 we join and fork sub-streams in an hierarchical manner.

Results

Table 3 summarizes the results of the performance analysis. It reports the worst-case end-to-end delay for data frames predicted by the different models. For the sake of simplicity, we compute the end-to-end delay as sum of the response times of the individual links. Tighter bounds for the end-to-end delay could potentially be obtained by considering that an data frame cannot experience the worst-case interference of audio frames consecutively on all three links. However, such a holistic analysis is more involved and out of the scope of this paper.

	Classic	FIFO	ECC (flat)	ECC (hierar.)
Max. delay	1.954 s	1.255 s	1.316 s	1.248 s

Table 3. Worst-case end-to-end delay for data frames derived with the different models

The table shows that the methods proposed in this paper to model join and fork scenarios of streams lead to considerably better results for the analysis of the HCS compared to the naive modeling approach of Figure 12. In particular, based on the classic MPA analysis, we would have to reject the designed system, as we could not guarantee that all data frames meet their deadline. On the other hand, the MPA models based on the abstract FIFO component or on ECCs show that the deadline for the transmission of data frames is never violated and hence the designed system fulfills the requirement. The reason for the better results is that the newly proposed methods permit to capture the amount of audio traffic on Links 2 and 3 more precisely than the naive MPA model. In particular, the predicted worst-case availability of Links 2 and 3 for the transmission of data frames is considerably less pessimistic, which finally yields tighter delay predictions. Figure 17 shows the corresponding lower service curves β'^{l}_{2} for Link 2.

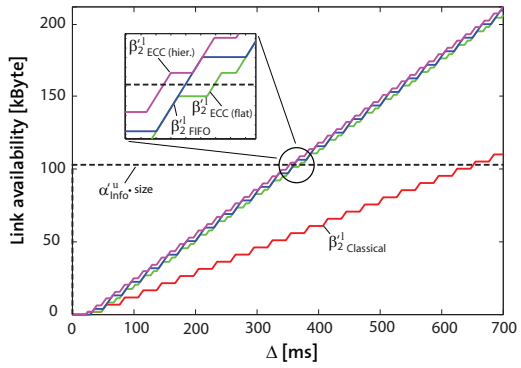


Figure 17. Minimum availability of Link 2 for the transmission of data frames (β'^{l}_{2}) determined by the different models. The dashed line (α'^{u}_{data}) represents the maximum demand of link capacity.

Table 3 and Figure 17 show that for the particular system under analysis, the model based on the hierarchical organization of ECCs turns out to be best. Nevertheless, the results achieved with a flat organization of ECCs are comparable. Moreover, we see that in terms of results there is not a major difference when using abstract FIFO components instead of structured event streams and ECCs. However, as pointed out before, the FIFO technique requires the designer to stick to a particular model for the processing semantics of components and hence it is less general.

8. Conclusion

We introduced two new methods for the modeling and the analysis of joined event streams in modular system performance analysis. The methods allow to precisely capture the structure of joined event streams that are processed by the computation and communication components of distributed embedded systems. The first approach, based on FIFO scheduling, keeps individual sub-streams separated and extends the Modular Performance Analysis framework by a new abstract processing component. The second approach, based on Event Count Curves, explicitly handles the joining and forking of event streams and is totally transparent to existing modeling components. Hence, it is highly suited for being embedded into present modular performance analysis frameworks. We further extended the approach based on Event Count Curves such that arbitrary decompositions of event streams can be represented, which considerably extends the modeling scope of the method. By means of experiments we evaluated and compared the presented methodologies, and highlighted the utility of all proposed approaches. Finally, we demonstrated the applicability of the described methods by a analyzing a realistic application scenario.

References

[1] K. Albers, F. Bodmann, and F. Slomka. Hierarchical event streams and event dependency graphs: a new computational model for embedded real-time systems. *18th Euromicro Conference on Real-Time Systems*, pages 97–106, 2006.

[2] J.-Y. L. Boudec and P. Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*, volume 2050 of *LNCS*. Springer, 2001.

[3] S. Chakraborty, S. Künzli, and L. Thiele. A general framework for analysing system properties in platform-based embedded system designs. In *Design Automation and Test in Europe (DATE)*, pages 190–195, Munich, Germany, Mar. 2003. IEEE Press.

[4] R. Cruz. A calculus for network delay, Parts 1 & 2. *IEEE Transactions on Information Theory*, 37(1), 1991.

[5] W. Haid and L. Thiele. Complex task activation schemes in system level performance analysis. In *Proc. 5th Intl Conf. on Hardware/Software Codesign and System Synthesis (CODES+ISSS 2007)*, pages 173–178, Salzburg, Austria, Oct. 2007. ACM Press.

[6] 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.

[7] K. Huang, L. Thiele, T. Stefanov, and E. Deprettere. Performance analysis of multimedia applications using correlated streams. In *Design, Automation and Test in Europe (DATE 07)*, pages 912–917, Nice, France, Apr. 2007.

[8] M. Jersak and R. Ernst. Enabling scheduling analysis of heterogeneous systems with multi-rate data dependencies and rate intervals. In *Design Automation Conference, 2003. Proceedings*, pages 454–459, 2003.

[9] J. Pasaje, M. Harbour, and J. Drake. MAST real-time view: A graphic UML tool for modeling object-oriented real-time systems. In *22nd IEEE Real-Time Systems Symposium, 2001.(RTSS 2001). Proceedings*, pages 245–256, 2001.

[10] P. Pop, P. Eles, and Z. Peng. Performance estimation for embedded systems with data and control dependencies. In *Proceedings of the 8th International Workshop on Hardware/Software Codesign*, pages 62–66. ACM New York, NY, USA, 2000.

[11] T. Pop, P. Eles, and Z. Peng. Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems. In *Proceedings of the 10th International Symposium on Hardware/Software Codesign*, pages 187–192. ACM New York, NY, USA, 2002.

[12] T. Rein, K. Lampka, and L. Thiele. Modeling hierarchical event streams in system level performance analysis. Technical Report 295, Computer Engineering and Networks Laboratory, ETH Zurich, Nov. 2008.

[13] J. Rox and R. Ernst. Modeling event stream hierarchies with hierarchical event models. In *Design, Automation and Test in Europe (DATE 2008)*, pages 492–497, March 2008.

[14] S. Schliecker and R. Ernst. A recursive approach to end-to-end path latency computation in heterogeneous multiprocessor systems. In *Proceedings of the International Conference on Hardware-Software Codesign and System Synthesis 2009*, pages 433–442, 2009.

[15] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *IEEE International Symposium on Circuits and Systems (ISCAS 2000)*, volume 4, pages 101–104, Geneva, Switzerland, Mar. 2000.

[16] E. Wandeler. *Modular Performance Analysis and Interface-Based Design for Embedded Real-Time Systems*. PhD thesis, PhD Thesis ETH Zurich, Sept. 2006.

[17] E. Wandeler, L. Thiele, M. Verhoef, and P. Lieverse. System architecture evaluation using modular performance analysis: a case study. *International Journal on Software Tools for Technology Transfer (STTT)*, 8(6):649–667, 2006.