Modeling Hierarchical Event Streams in System Level Performance Analysis

TIK Report 295

Tobias Rein, Kai Lampka, Lothar Thiele
Computer Engineering and Networks Laboratory
Swiss Federal Instisute of Technology (ETH) Zurich, Switzerland
{rein|lampka|thiele}@tik.ee.ethz.ch

December 2, 2008

Abstract

The growing complexity of embedded real-time systems requires elaborate methods in their design process. While simulation does not sufficiently cover corner cases, 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 are the limited system configurations that can be analyzed with high accuracy. In this report we extend existing methods for analyzing heterogeneous multiprocessor systems such that different types of data streams can be composed to a higher level event stream with multiple hierarchies. Furthermore, we describe how such hierarchical event streams can be decomposed into their sub-streams. Additionally, we show how these new methodologies are used to model the distributed heterogeneous communication system of the COMBEST case study provided by EADS.

1 Introduction

In the context of the COMBEST project, we will provide a formal framework for component based design of complex embedded systems. One of the milestones to reach this goal is the analysis of a case study provided by EADS Innovation Works [4] in WP 3. This case study concerns a distributed Heterogeneous Communication System (HCS) containing wired and wireless communication networks based on Ethernet. A common server is connected to different devices, such as sensors (camera, smoke, pressure, etc.) and actuators (speakers, light switches, temperature control, signs, etc.). The functions and applications of the Heterogeneous Communication System are running on common platforms that can be efficiently scaled to incorporate various requirements. Amongst others, the server broadcasts audio streams to the devices, which are equipped

with speakers to play back the audio signals. To achieve clock synchronization the PTP (Precision Time Protocol, IEEE1588) is used. Some key requirements of the systems are the following:

- The system should provide up to 190 devices
- The system should be able to transmit up to 10 audio streams broadcasted by the server
- The maximal time difference between sending a signal (e. g. speaking into the microphone connected to the server) and playback of the signal at the devices should be less than 0.1 sec.
- The audio samples have to be played at all devices with a maximum jitter of 0.1 ms (e. g. the maximal time difference of the playback time between any two devices in the whole system is less than 0.1 ms)

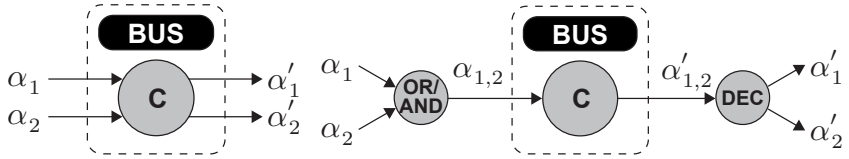
Clock synchronization as well as audio signal transmission require hard real-time constraints such as guaranteed maximum end-to-end delays. This involves both communication and computation. Because of the complexity of the overall system a compositional system analysis is necessary. In addition, precise protocols for resource sharing are not yet known. Thus, testing of various system configurations is necessary. For that reason we need a fast analysis that delivers hard bounds in terms of worst-case and best-case on various system properties. Therefore, simulation is not possible since it is neither correct in terms of hard bounds nor fast.

All these requirements including the size of the system ask for a component based design of the analysis model. In the field of performance analysis of embedded real-time systems, different techniques have been introduced in the past. SymTA/S [3] and a technique denoted as modular performance analysis (MPA) [3, 10] are both tools which perform compositional system analysis. They use local analysis techniques for analyzing the individual components, which are interconnected via event streams. As main contribution, these tools deliver hard bounds on minimum and maximum end-to-end delays, queue sizes or resource utilization.

A major problem of these methods is the restricted scope i. e. classes of systems that can be analyzed with high precision. For classes not covered by the methods, simplifications are needed resulting in a loss of accuracy of the computed bounds or in a loss of the compositionality of the methods.

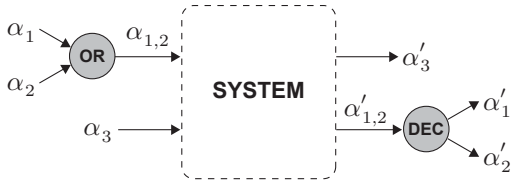
A system configuration, which is highly relevant for practice occurs in communication systems, if multiple data streams are combined in packets and transmitted over a communication channel. According to [8] the combined data stream is denoted as hierarchical event stream. In the COMBEST case study by EADS [4] the broadcast of different audio streams by the server is such a scenario. Once an audio packet has been successfully delivered and arrives at the end-point of the communication channel, the data have to be de-packetized i. e. the hierarchical event stream has to be decomposed into the individual data streams. In this report we introduce a major extension to the aforementioned methods that allows joining and forking event streams with high accuracy while keeping the compositionality of the methods. Related to the COMBEST

case study the server packetizes the audio streams and after successful transmission the devices have to de-packetize the stream to play back the audio signals. Without the concept of the hierarchical event streams, the COMBEST case study scenario could not be modeled accurately and different abstraction would be needed resulting in pessimistic worst-case bounds.



(a) Communication task activated by multiple inputs.

(b) Model of Fig. 1(a) for the MPA domain.



(c) A system in which the combined stream $\alpha_{1,2}$ may pass several tasks before being decomposed.

Figure 1: Examples of tasks with multiple input streams

As an example, we consider a system with a single communication task C activated by two streams α_1 and α_2 as shown in Fig. 1(a). The task delivers the outgoing streams α'_1 and α'_2. Associated with the COMBEST case study [4], the two streams α_1 and α_2 could be audio signals that are transmitted from one network component to the other. The characteristics of the output streams depend on the incoming data streams α_1 and α_2 as well as on the availability of the resource, i. e. the availability of the bus in this example. In the context of modular performance analysis, tasks activated by multiple input streams are normally modeled as a combination module that combines the streams followed by the task using the combined stream as input as depicted in Fig. 1(b). For the task activation pattern in the combination module one mainly distinguish between OR- and AND-activation [5, 8]. In case of an OR-activation, each event on any of the inputs implies the execution of the task e. g. transmission over the communication system. The events are put into packets but are sent individually. In the COMBEST cast study such an OR-activation is realized by combining the audio streams. Every audio segment is packetized from the server and broadcasted over the network. Contrary, the AND-component only executes if there is an event present on each input and each event is removed after processing. Therefore, every event generated by an AND-activated task, denoted as packet, contains one event of each input stream. While in the context of MPA combining streams in an AND-activated task is challenging but in detail described in [5], the decomposition of an AND-activated combined stream in the context of MPA is simple, since we only consider number of

events/packets in time intervals. As we have one event of each stream in one packet, the combined output stream does not differ from the individual output streams and therefore $\alpha'_1 = \alpha'_2 = \alpha'_{1,2}$ holds. Since AND-activated tasks are not present in the COMBEST case study, we only consider OR-activated tasks in this report.

In the following, we propose two new approaches to deal with hierarchical event streams in modular performance analysis. The first approach uses the well-know FIFO scheduling to manage event hierarchies as the second approach uses a new concept called Event Count Curves. Both methods are convenient to model the above mentioned scenario with high accuracy.

In Section 2 we give details about the Real-time Calculus [9], the framework we are using for the MPA. Furthermore, we summarize the Hierarchical Event Model introduced by the authors of [7]. In Section 3 and 4 we present two different approaches to deal with hierarchical event streams. The first approach is considering FIFO scheduling while the second uses a new concept denoted as Event Count Curves. Finally, in Section 5, we give an example how hierarchical event streams can be applied and conclude our work in Section 6.

2 Related Work

Recently, Rox et al. [8] 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 model such hierarchical streams. The key of their 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. To decompose the HES into the different input streams, one only needs to extract the inner event streams from the HES. This model allows to accurately analyzing the processing and communication on the combined as well as on the embedded individual streams. The authors demonstrated significant improvements in terms of tightness of the worst case response time by using HES instead of flat event stream models. Nevertheless, in the processing step of hierarchical event streams in a component, the HEMs have to be deeply processed and therefore each component has to be able to handle such hierarchical streams.

Albers et al. [1] used a hierarchical data structure to describe repetitively occurring patterns within event streams. Since such scenarios refer to the combination of streams, the authors also referred to their approach as hierarchical event stream, which is also justified since the pattern are given as PJD-models, so that the overall model can be viewed as hierarchical (or nested) PJD-model. Nevertheless, the topic faced in our report is not related to their work, since it does not assume any pattern of event occurrence. The approach presented here takes arbitrary input patterns.

In this report, we use MPA-RTC introduced by Chakraborty et al. [3] as the basis for our analysis. In this compositional approach, a system is modeled as a set of components interconnected by streams representing event or resource streams. However, contrary to other techniques the streams are not defined on a straight time-line, but for time intervals of length Δ. Formally, we define the bounding function $\alpha(\Delta) =$

$\big(\alpha^u(\Delta), \alpha^l(\Delta)\big)$, also commonly denoted as arrival curve, which gives the maximum and minimum number of events arriving in any time interval of length Δ. In practice such functions can be obtained from formal specifications, traffic models such as periodic with jitter (PJD-models), or from simulation traces. Analogous to the upper and lower arrival curves, one may also define an upper and lower bound for the availability of resources $\beta(\Delta) = \big(\beta^u(\Delta), \beta^l(\Delta)\big)$.

For computing the bounding functions on outgoing events $\alpha'(\Delta)$ and remaining resources $\beta'(\Delta)$ the principles of the Real-Time Calculus [9] are applied on the input curves. The method is based on the Network Calculus [2] and uses the concept of the min-plus and max-plus algebra which define the min-plus convolution ($\otimes$) and min-plus deconvolution ($\oslash$) as well as the analogons max-plus convolution ($\overline{\otimes}$) and max-plus deconvolution ($\overline{\oslash}$). The most common component used in the MPA-RTC is the greedy processing component (GPC). This component instantiates a fully preemptable task at every event arrival and active tasks are sequentially processed in a greedy manner. Since the resource availability is limited, an input queue in form of a buffer is needed. The outgoing event stream α' and the remaining resource β' of the component are given by the following relations:

$$\begin{aligned}
\alpha'^u_{GPC} &= \min\{(\alpha^u \otimes \beta^u) \oslash \beta^l, \beta^u\} \\
\alpha'^l_{GPC} &= \min\{(\alpha^l \oslash \beta^u) \otimes \beta^l, \beta^l\} \\
\beta'^u_{GPC} &= (\beta^u - \alpha^l) \,\overline{\oslash}\, 0 \\
\beta'^l_{GPC} &= (\beta^l - \alpha^u) \,\overline{\otimes}\, 0
\end{aligned} \tag{1}$$

For a system with components that are activated by multiple event streams as depicted in Fig. 1(b), we also need to know how to combine two or more event streams. In [5], the authors give a detailed derivation and come out with the following formula for an OR-activated task:

$$\alpha^u_{OR} = \sum_i \alpha^u_i \;\; ; \;\; \alpha^l_{OR} = \sum_i \alpha^l_i \tag{2}$$

3 FIFO scheduling

3.1 System Composition

In this section, we describe how to use FIFO scheduling to deal with hierarchical event streams. The key of this approach is that we keep track of the individual input streams instead of evaluating the combined stream. The method relies on the fact, that an OR-activation does not change the order of the arriving events and forwards them immediately in a FIFO manner into the output stream. For this approach, we omit the step of combining the streams completely but keep the streams separate as a bundle of streams. Each time this bundle is going to be processed in a task we replace the task with a FIFO component using the resource input of the task and the streams contained in the bundle as inputs. This FIFO component is justified since also the combined stream would behave like this according to the OR-activation as stated before. In Fig. 2

a simple example with only one task is shown on the left. If we want to model this system using FIFO scheduling, we remove the OR-activation and replace the task with a FIFO component as shown in Fig. 2 on the right.

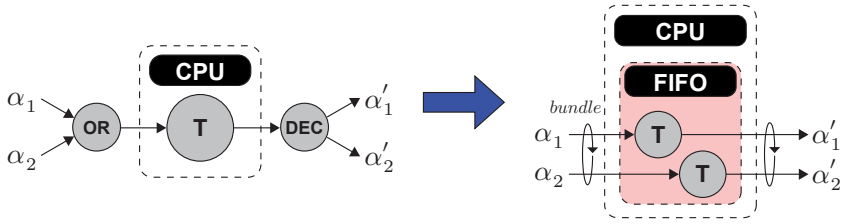


Figure 2: A simple system with an OR-activated task on the left and how it is modeled with FIFO scheduling on the right.

To evaluate properties related to the combined stream such as required buffer sizes or end-to-end delays of the packets of the combined stream we have to perform a separate analysis with the original system i. e. we have to combine the streams using (2) and to evaluate the performance analysis with this combined stream and the original tasks.

3.2 FIFO Component

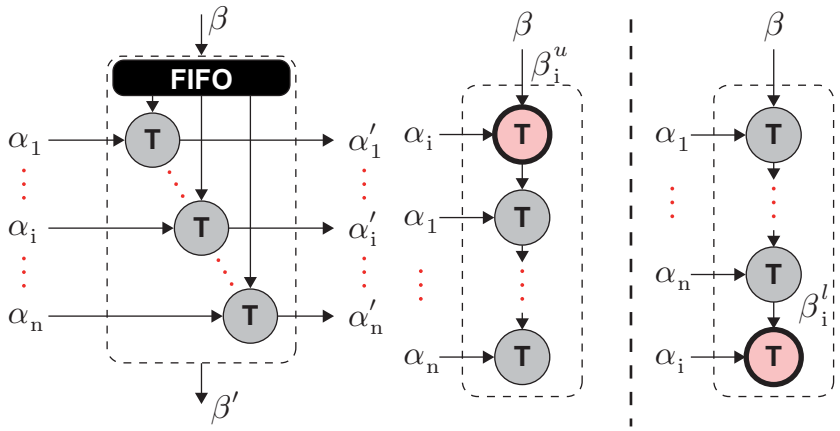
In this section, we show how a FIFO component can be implemented in the context of MPA-RTC. A FIFO component as depicted in Fig. 3(a) models a set of tasks in a real-time system that share an available resource in a FIFO manner. As input, the FIFO component has one resource stream β and n event streams $\alpha_1, \ldots, \alpha_n$.

The remaining resource β' of the FIFO-component can be calculated using a variation of the formula (1) for a GPC. As input event stream α we us the sum of all the input streams $\alpha_1, \ldots, \alpha_n$ according to (2)

$$\begin{aligned} \beta'^{u}_{FIFO} &= (\beta^u - \sum_i \alpha_i^l) \overline{\oslash} 0 \\ \beta'^{l}_{FIFO} &= (\beta^l - \sum_i \alpha_i^u) \overline{\otimes} 0 \end{aligned} \tag{3}$$

To determine the bounds of the output event streams $\alpha'_1, \ldots, \alpha'_n$, we analyze the minimum and maximum available resource for every input stream, respectively. Hence, we consider two cases for every stream α_i: a best-case, where task T_i associated with stream α_i has the highest priority and a worst-case, where the same task T_i has the lowest priority. In the first case, we observe that the maximum available resource β_i^u for task T_i is the upper bound β^u of the total resource input β of the FIFO component. The minimum resource β_i^l available for task T_i is the remaining resource after all other tasks as shown in Fig. 3(b). Using this policy, we can derive the following formulas to limit the available resource for every stream α_i:

$$\beta_i^u = \beta^u \;\; ; \;\; \beta_i^l = (\beta^l - \sum_{j \neq i} \alpha_j^u) \overline{\otimes} 0 \tag{4}$$



(a) Model of the FIFO Component in the Real-Time Calculus.

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

Figure 3: Model of the FIFO Component

Given the bounds β_i^u, β_i^l on the available resource for every task T_i, the outgoing streams of the FIFO component can then be calculated using a greedy processing components with α_i and β_i as its inputs. With (1) the outputs are given by

$$\begin{aligned} \alpha'^u_{i_{FIFO}} &= \min\{(\alpha_i^u \otimes \beta_i^u) \oslash \beta_i^l, \beta_i^u\} \\ \alpha'^l_{i_{FIFO}} &= \min\{(\alpha_i^l \oslash \beta_i^u) \otimes \beta_i^l, \beta_i^l\} \end{aligned} \tag{5}$$

4 Event Count Curves

4.1 Definition

Unlike the Hierarchical Event Model [8, 7] and the approach using FIFO scheduling introduced in the previous section, the method using Event Count Curves (ECC) does not keep track of the individual input streams. Therefore, it can be denoted as compositional and hierarchical analysis method as a flattening of the hierarchical event stream is not necessary for being processed in a tasks. With the ECCs, the focus lies on the composition and decomposition of the combined stream as shown in Fig. 4.

The basic idea of this approach relies on the fact, that the order of events in an event stream remains the same whatever operation is applied to the event stream, i. e. the sequence of the events before and after a task is the same. Therefore, if we store the

pattern how the input streams are combined at the OR-activation or packetization unit, we can later on use this information to decompose the stream into its individual sub-streams. The main advantage of this method is that besides the composition and the decomposition of the stream, the methodologies for analyzing the system do not have to be changed or modified due to the existence of hierarchical event streams. Furthermore, the sub-system between the OR-activation and its corresponding decomposition is not limited to a few components but can be of arbitrary complexity.

As mentioned earlier, an event stream is given by an upper and lower bound, which limit the occurrences of events in a given time interval. One may note that we consider the time interval domain and not the time domain. Using such a model, it is not possible to evaluate the exact pattern of the composition of the combined stream after an OR-activation of several event streams, but we can give hard bounds for the number of events related to a specific input stream with respect to the number of events in the combined stream.

Definition 1 (Event Count Curve γ)**.** *For a fixed number of consecutive events in the combined stream, the Event Count Curve* $\gamma_i(n) = \left(\gamma_i^u(n), \gamma_i^l(n)\right)$ *bounds the number of events related to input stream* α_i*.*

The Event Count Curves (ECCs) are a powerful model to describe the composition of a combined event stream. They are related to the workload curves introduced by Maxiaguine et al. [6] but clearly differ in the usage and meaning. In particular, the ECCs can be used to decompose the combined stream into its sub-streams as follows:

Theorem 1 (Decomposition with Event Count Curves)**.** *If* α' *is the combined output stream that needs to be decomposed and* γ_i *the ECC related to events of stream* α_i*, the sub-stream* α_i' *is given by the concatenation of these two curves.*

$$\alpha_i'(\Delta) = \gamma_i\left(\alpha'(\Delta)\right) \tag{6}$$

Proof. The evaluation of $\alpha'(\Delta)$ gives the minimum or maximum number of events for a given interval Δ, respectively. This number can be translated using the ECC γ_i which gives then the number of events related to sub-stream α_i' for the given interval Δ. □

Example 1 (Event Count Curves)**.** *Consider the system given in Fig. 4 with the two strict periodical input streams* α_1 *and* α_2 *with period* $p_1 = 10$ *and* $p_2 = 20$ *(jitter* $j_1 = j_2 = 0$*) which may represent periodic audio signals or synchronization sequences in the COMBEST case study . Example traces of the individual streams as well as of the combined streams are depicted in Fig. 5(a). The Event Count Curves derived at the OR-activation are shown in Fig. 5(b). The meaning of the ECC is the following; if you look at* 4 *consecutive events in the combined streams, we have at least* 2 *and at most* 3 *events from stream* α_1 *which is expressed by the following notion:* $\gamma_1^l(4) = 2$ $,\gamma_1^u(4) = 3$*.*

4.2 Join and Fork

In this section we describe how Event Count Curves can be derived from two input streams $\alpha_1(\Delta)$ and $\alpha_2(\Delta)$ in the context of the RTC. We give a closed solution how to

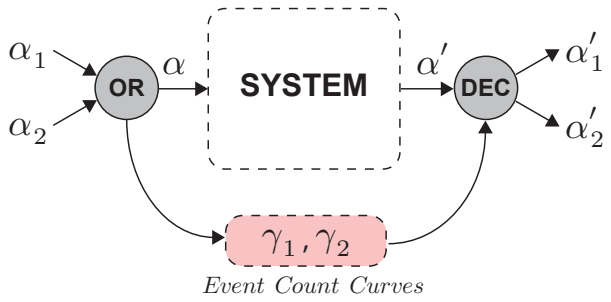


Figure 4: System used in Ex. 1.

compute the ECCs, show a detailed derivation and proof the correctness of the formula.

First we introduce the $\delta_i^{u,l}(e)$ curves that indicate the maximum or minimum time interval in which at most or at least e events occur, respectively. These curves are strongly related to the arrival curves $\alpha_i^{u,l}(\Delta)$. Le Boudec et al. [2] defined the pseudo-inverse function which exactly describes the relationship between $\alpha_i^{u,l}(\Delta)$ and $\delta_i^{u,l}(e)$:

$$\begin{aligned} \delta_i^l(e) &:= \inf_{0 \leq \Delta} \{\Delta : \alpha_i^u(\Delta) \geq e\} \\ \delta_i^u(e) &:= \sup_{0 \leq \Delta} \{\Delta : \alpha_i^l(\Delta) \leq e\} \end{aligned} \tag{7}$$

With this definition, we can give the desired closed formula to determine the ECCs.

Theorem 2 (Event Count Curves in the RTC)**.** *If α_1 and α_2 are two arrival curves, then the ECC $\gamma_1^{u,l}$ related to stream α_1 can be computed as follows*

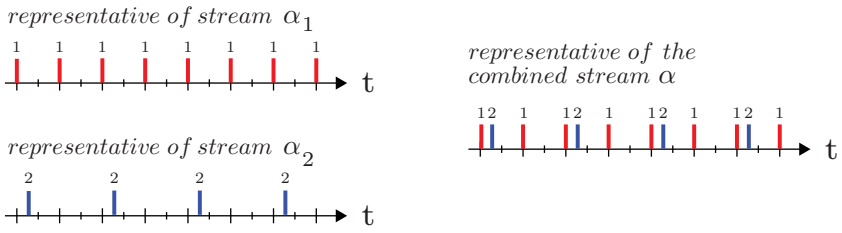
$$\begin{aligned} \gamma_1^l(e) &= \inf\{e_1 : e_1 + \alpha_2^u(\delta_1^u(e_1)) \geq e\} \\ \gamma_1^u(e) &= \sup\{e_1 : e_1 + \alpha_2^l(\delta_1^l(e_1)) \leq e\} \end{aligned} \tag{8}$$

where δ_1 is the pseudo-inverse of α_1 defined in (7).

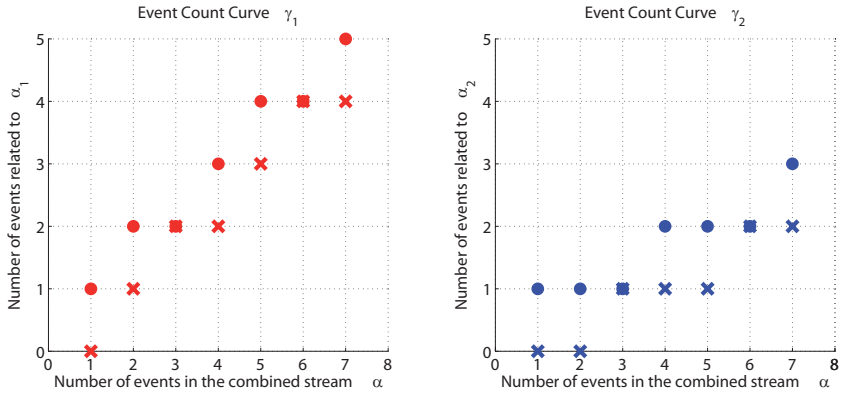
Proof. First, we define the $\epsilon_i^{u,l}(e_i)$ as pseudo-inverse curves of the event count curves $\gamma_i^{u,l}(e)$

$$\begin{aligned} \epsilon_i^l(e_i) &:= \inf\{e : \gamma_i^u(e) \geq e_i\} \\ \epsilon_i^u(e_i) &:= \sup\{e : \gamma_i^l(e) \leq e_i\} \end{aligned} \tag{9}$$

where e is the number of events in the combined stream and e_i is the number of events related to input stream α_i. $\epsilon_i^{u,l}(e_i)$ give the maximum or minimum number of consecutive events in the combined stream so that at most or at least e_i events in this sequence are related to input stream a_i, respectively. Since the pseudo-inverse is reversible we can express $\gamma_i^{u,l}(e)$ using $\epsilon_i^{u,l}(e_i)$ in the following way



(a) Example traces used in Ex. 1.



(b) Event Count Curves from Ex. 1 where dots indicate the upper and crosses the lower bounds of the ECCs

Figure 5: Example traces and event count curves of Ex. 1

$$\gamma_i^l(e) := \inf\{e_i : \epsilon_i^u(e_i) \geq e\}$$
$$\gamma_i^u(e) := \sup\{e_i : \epsilon_i^l(e_i) \leq e\} \tag{10}$$

For the proof, we are considering events related to stream α_1. $\gamma_1^{u,l}(e)$ bounds the number of events e_1 related to α_1 as follows

$$\gamma_1^l(e) \leq e_1 \leq \gamma_1^u(e) \tag{11}$$

which can be written with the pseudo-inverse $\epsilon_1^{u,l}$ as follows:

$$\epsilon_1^l(e_1) \leq e \leq \epsilon_1^u(e_1) \tag{12}$$

We also need information about the combined stream. We know, that the total number of events e is the sum of the two input streams and therefore

$$e = e_1 + e_2 \tag{13}$$

must be hold for all time. If we consider $\epsilon_1(e_1)$, the number of events e_1 is fixed and therefore we can define a time interval Δ^* that satisfies

$$\delta_1^l(e_1) \leq \Delta^* \leq \delta_1^u(e_1) \tag{14}$$

This interval indicates the time in which exactly e_1 occur in stream α_1. With this interval we can bound the occurrences of e_2 by the following bounds

$$\begin{aligned} \alpha_2^l(\Delta^*) \leq e_2 \leq \alpha_2^u(\Delta^*) \\ \alpha_2^l(\delta_1^l(e_1)) \leq e_2 \leq \alpha_2^u(\delta_1^u(e_1)) \end{aligned} \tag{15}$$

Combining (12), (13) and (15) we find a closed formula for $\epsilon_1^{u,l}(e_1)$

$$\begin{aligned} \epsilon_1^l(e_1) = e_1 + \alpha_2^l(\delta_1^l(e_1)) \\ \epsilon_1^u(e_1) = e_1 + \alpha_2^u(\delta_1^u(e_1)) \end{aligned} \tag{16}$$

With this formula and the definition of $\gamma_i^{u,l}(e_i)$ in (10) we can derive the Event Count Curve by concatenating these equations.

□

5 Case Study

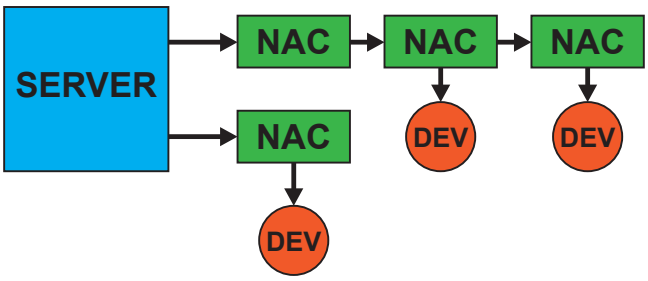


Figure 6: Distributed multimedia system with a server broadcasting audio streams to several devices (DEV) via network access controllers (NAC).

In the following section, we show how the concept of the hierarchical event streams can be applied to the COMBEST case study by EADS [4] introduced in Section 1. The system described in the case study consider a distributed multimedia cabin system with a server broadcasting different audio streams to various playback devices as depicted in Fig. 6. The server is connected to several network access controllers (NACs) that forward the streams either to another NAC or to the playback device itself. Hence, we have a daisy-chain like backbone topology in the cabin. There exists also other

traffic e. g. status and control messages for emergency signs or smoke detectors and time synchronization packets that are routed in the same way as the audio streams through the network. The devices are equipped with a speaker (output for headphones) and control logic so that the user can switch between the individual audio streams he wants to listen to.

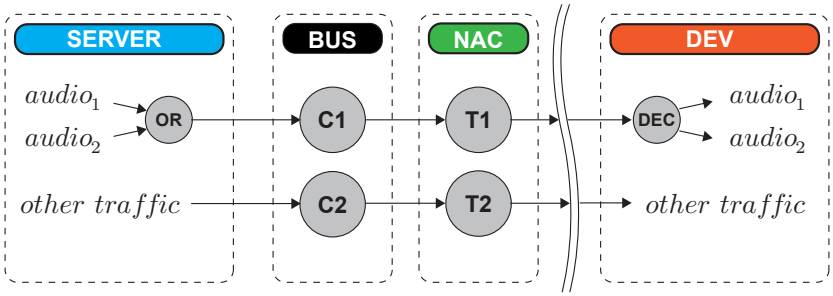


Figure 7: MPA model of the distributed multimedia system shown in Fig. 6 using hierarchical event streams.

Fig. 7 shows the MPA model of the distributed multimedia system described in the EADS case study [4] with two audio streams only. Since these audio streams are broadcast streams it is feasible to combine them to a hierarchical event stream. That means, we do not have to model a task for every audio stream individually but only one for the hierarchical event stream. Therefore, the streams are combined in the server, transmitted over the network and decomposed in the playback devices. On the communication channel, modeled as bus, as well as in the network access controllers (NACs) we only see one task for all the audio streams. This approach is highly scalable in terms of number of audio streams since only the server and the playback devices need to be changed if we increase or decrease the number of audio signals.

6 Conclusion

In this report, we showed how FIFO scheduling, a well-known technique, can be used to handle hierarchical event streams in the context of system-level performance analysis based on the MPA-RTC. In addition, we introduced the concept of the Event Count Curves which is more modular and also enables one to model hierarchical event streams. With an example system we showed the applicability of hierarchical event streams on a real-world application.

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. *Real-Time Systems, 2006. 18th Euromicro Conference on*, pages 10 pp.–, 5-7 July 2006.

[2] J.-Y. L. Boudec and P. Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*, volume 2050. 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] EADS Inovation Works. *COMBEST - Case study on distributed hererogeneous communication systems*, October 2008.

[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] A. Maxiaguine, S. Künzli, and L. Thiele. Workload characterization model for tasks with variable execution demand. In *Design Automation and Test in Europe (DATE)*, pages 1040–1045, Paris, France, feb 2004. IEEE Computer Society.

[7] J. Rox and R. Ernst. Construction and deconstruction of hierarchical event streams with multiple hierarchical layers. In *In Proceedings of the 20th Euromicro Conference on Real-Time Systems (ECRTS)*, 2008.

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

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

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