Construction and Deconstruction of Hierarchical Event Streams with Multiple Hierarchical Layers

Jonas Rox, Rolf Ernst
Institute of Computer and Communication Network Engineering
Technical University of Braunschweig
D-38106 Braunschweig / Germany
{rox|ernst}@ida.ing.tu-bs.de

Abstract

Compositional Scheduling Analysis couples local scheduling analysis via event streams. While local analysis has successfully been extended to include hierarchical scheduling strategies, event streams are still flat. In this paper, we formally define hierarchical event streams, which cannot only be constructed from flat event streams, but also from hierarchical streams allowing event streams with multiple hierarchical layers. We define an hierarchical event model and the operations to construct and deconstruct hierarchical events streams. Finally, we demonstrate how the model can be integrated in an existing analysis approach for distributed systems, enabling superior analysis results.

1. Introduction

System and communication platform integration is a major challenge and systematic analysis of the complex dynamic timing effects (scheduling, arbitration, blocking, buffering) becomes key to building safe and reliable systems. During system design, a system model is developed, that captures information about the applications and the available hardware architecture of the system, defines the mapping of tasks to computation or communication resources and specifies the scheduling and arbitration schemes used on these resources. Additionally, an environmental model that describes how a system is being used (e.g. how often will events arrive, activating the system) is needed also.

To give worst case guarantees for the timing behaviour of a distributed system, different compositional approaches for system level performance analysis have been developed [6][10]. Common to the different compositional approaches is that they use local analysis techniques for analysing the individual components, which are interconnected via event streams, where the output event stream of one task turns into the input event stream of connected tasks. Global system analysis, then iterates between local component analysis and output stream calculation. More precisely, in each global iteration of the compositional system level analysis, local analysis is performed for each component to derive response times and the timing of output event streams. Afterwards, the calculated output event streams are propagated to the connected components, where they are used as input event streams for the subsequent global iteration. The different approaches, not only use different analysis techniques at the local component level, they also use different models to describe the event streams.

While local analysis has successfully been extended to include hierarchical scheduling strategies [7][9], event streams are still flat. A recent proposal [1] introduces event hierarchies in a stream, which enable to model streams with complex event arrival patterns. But the proposed model still describes only a flat event stream. But when event streams are combined, e.g. when a task is activated by multiple event streams or when a communication channel is shared by different tasks, the existing models cannot capture the relations between the different layers of the combined stream.

Observe the small example configuration shown in Figure 1 which may be part of a greater system. The tasks T1 and T2 are both activated by multiple input streams. If we assume that different activating events also produce different output events, not only their input streams contain events from different sources, but also the events of their output streams can be related to the different input streams. Since in the illustrated example, both tasks share a communication channel to communicate their results to another resource, their output streams are again combined. Hence, the events of the input stream of the communication task

The research described was supported by the German Government (BMBF) as part of the SuReal-Project

1068-3070/08 $25.00 © 2008 IEEE
DOI 10.1109/ECRTS.2008.13

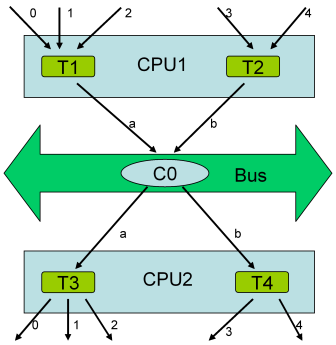

Figure 1. A motivational example system

C0 modeling the bus communication, contains events outputted by T1 and T2, which in turn can be related to the different input events of the tasks T1 and T2. Consider that T3 is only activated by those output events of C0 generated due to an activating event coming from T1 while T4 is only activated by those output events of C0 that are generated due to activating event from T2(as indicated by the small labels). Since existing event models cannot capture such inner relations of combined event streams, if used in the above example, it can only be conservatively assumed that the tasks T3 and T4 are activated on each completion of the communication task C0 and that on each completion of these tasks, an event is generated on each output.

This paper presents the following contributions:

- We define hierarchical event streams and an appropriate model to capture such streams consisting of multiple layers of embedded event streams.
- We present how hierarchical event models can be applied to model AND- and OR-combined event streams, conserving the inner relations of the combined streams. We then use these relations to determine how the inner streams are affected when the timing of a stream at a higher layer is changed.
- We integrate the proposed model into an existing system analysis tool and show the improvements that can be obtained when exploiting the captured inner stream relations.

The remainder of this paper is structured as follows. First, we will review related work (Section 2). We then give formal definitions of the existing general system models used for distributed system analysis and its elements, introduce stream hierarchies, and define a hierarchical event model to capture these stream hierarchies (Section 3). In Section 4 we introduce the used analysis frame work and in Section 5 we present how the hierarchical event models for the existing activation semantics can be constructed and how these hierarchical event models can be deconstructed. Finally, we use the new event model for analysing the introduced example systems to proof its applicability and to demonstrate its benefits (Section 6) before we draw conclusions (Section 7).

2. Related Work

The analysis framework presented by Richter [6] defines four characteristic functions $\eta^+(\Delta t), \eta^-(\Delta t), \delta^-(n)$ and $\delta^+(n)$ to describe event streams. The first two functions bound the number of events that can occur in a given time interval of size Δt and the last two functions bound the distance between n events. At the component level, formal analysis techniques based on the busy window technique proposed by Lehoczky [5] are used. Standard event models, consisting of the three parameters period (P), jitter (J) and minimum distance (d_{min}), are used as parameterized representation of the four characteristic functions. They enable a very efficient computation of the four characteristic functions and also of the output event models, but can lack in precision when it comes to approximating arbitrary event streams .

The compositional approach presented by Thiele et. al. [10] uses numerical upper and lower event arrival curves for describing load generated by event streams, and similar service curves for execution modeling. Based on [2], not only new scheduling analysis algorithms for the local components, but also the corresponding output stream calculations, were developed. As parameterized representation, Thiele et. al. propose a Piecewise Linear Approximation [9] consisting of a combination of two line segments to approximate all arrival and service curves.

There exist some other event models used for describing the timing of event streams. In [3] Gresser introduced so called event vectors and defined how a demand bound function (DBF) that models the load a task produces on a resource can be calculated with the event model. The event model was extended in [1] to a hierarchical event model, that can accurately describe the timing of complex hierarchical structured event patterns of a single stream. A finite inner event sequence described by one or several event vectors can be embedded into another outer event sequence. An event of the outer sequence doesn't stand for a single event, but for the entire inner event sequence. This allows a more accurate description of the DBF, provided that enough information is available to actually determine the specific parameters.

To handle tasks with multiple input streams, the aforementioned analysis frameworks map the input event streams on one event stream, according to the activation semantic of the task. The resulting event stream is than used as input stream of the task [4][11]. Since the resulting event

stream doesn't contain any information about the individual event streams that were combined, it is not possible to reverse the combination later. But this is exactly what is needed to enable the analysis of system setups like depicted in Figure 1 where different tasks share the same communication channel. For analysis of the communication timing, only the event stream describing the channel activation timing is of interest, which can be calculated by an appropriate combination of the output streams of the sending tasks. But for the receiving tasks, the previously combined event streams must be extracted, incorporating scheduling effects the communication channel experiences, which cannot be done with the existing event models.

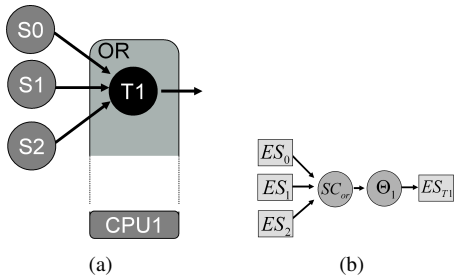

Figure 2. A simple example system (a) and the corresponding abstract system model (b)

3. General System Model

For performance analysis, a distributed system is modeled as event streams, interconnected by operations. All possible event sequences that could be observed at the input of a task, are modeled by an event stream. An event stream is modeled by a tuple of functions $F = (f_1, .., f_n)$, which bound the timing behaviour of the modeled event sequences.

Definition 1. *Event Model*
An event model defines the event stream functions F_{EM}.

Stream operations model the processing of event streams, defining a function that maps all input streams to an output stream. This function is evaluated in each global iteration step of the system analysis, after the local component analysis is completed.

Definition 2. *Stream Operation*
A stream operation Θ_{op} defines a function

$$T_{op} : F^n \rightarrow F^m$$

that delivers the corresponding function tuples of the m output event streams depending on the function tuples of the n input streams.

Tasks activated by multiple event streams are decomposed in two operations [4][11]: the first is an event stream constructor (SC), which combines the input stream according to the activation semantic and calculates one output stream that becomes the input stream of the second operation, which models the actual processing of the event streams due to the task.

Figure 2 shows a small part of a system and the corresponding model used for performance analysis, consisting of operations and event streams.

Since the output event stream ES_{T1} of the tasks T1 is modeled by a single function tuple F_{T1}, it contains no information about the timing of the combined event streams, which is needed to be able to decombine the event streams later.

Therefore we extend the system model by hierarchical event streams (HES), hierarchical stream constructors (HSC) and hierarchical event stream deconstructors. The idea is, that a hierarchical event stream has one *outer* representation in form of an event stream ES_{out} and for each combined event stream $ES_{in,i}$ it defines an event stream ES_i bounding the event occurrences of events that are related to $ES_{in,i}$.

Definition 3. *Hierarchical Event Stream*
A hierarchical event stream ES_h is the result of the combination of n input event streams $ES_{in,1}, .., ES_{in,n}$ and has one outer event stream ES_{out}, defined by a function tuple $F_{out} = (f_1, .., f_n)$ and n inner event streams, each defined by its own function tuple $F_i = (f_1^i, .., f_n^i)$.

To model a hierarchical event stream, we define a hierarchical event model:

Definition 4. *Hierarchical Event Model*
A hierarchical event model defines a parameter tuple H:

$$\begin{aligned} H = \{(F_{out}, L, C_\Omega) \,|\, & F_{out} = (f_1, .., f_n), \\ & L = (F_1, .., F_n), \\ & C_\Omega = (B_1^\Omega, .., B_k^\Omega)\} \end{aligned}$$

$F_{out} = (f_1, .., f_n)$ models the outer event stream.
$L = (F_1, .., F_n)$ is a List of function tuples modeling the inner streams.
The functions $(B_1^\Omega, .., B_k^\Omega) \in C_\Omega$ define how the inner event streams are affected when the outer event stream changes.

Definition 5. *Hierarchical Stream Constructor*
A hierarchical stream constructor Ω combines two or more event streams $ES_1, .., ES_n$ resulting in an hierarchical event stream ES_h. Formally:

$$HSC_\Omega : F^n \rightarrow H$$

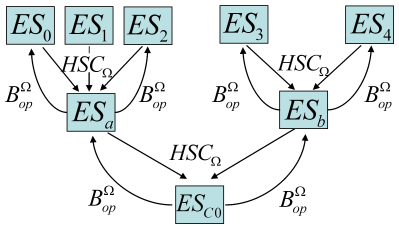

Figure 3. Relations between event streams of the hierarchical input stream of the communication task $C0$ from the introductional example

Figure 3 shows the hierarchical input event stream of the channel $C0$ of the motivational example and illustrates the relations between the different streams of the HES. The event stream ES_{C0}, that serves as input for the channel $C0$, depends on the event streams ES_a and ES_b as defined by the function HSC_Ω. ES_a and ES_b not only depend on their *inner* streams ES_0 - ES_4, but also on ES_{C0}. Every time an operation Θ_{op} is applied on ES_{C0}, the function B^Ω_{op} defines how ES_a respectively ES_b are affected. This in turn will affect the *inner* streams of ES_a and ES_b accordingly. As depicted in Figure 3, the *outer* event stream representation of an hierarchical event stream is used to combine the hierarchical stream with other event streams, forming deeper event stream hierarchies.

For extracting the *inner* event streams of a hierarchical event stream, we define a deconstructor.

Definition 6. *Hierarchical Event Stream Deconstructor*
A hierarchical event stream deconstructor Ψ, takes a hierarchical event model EM_h as input and returns the updated inner event models EM_i, modeling the individual streams, as output:

$$D_\Psi : H \rightarrow F^n$$

4. Event Streams and Event Stream Propagation

Before we define the concrete HSCs for AND-combining and OR-combining event streams, we will shortly review the basic concepts of the used system analysis and introduce our notation.

Similar to the framework presented by Richter [6] we use traditional analysis techniques at the local component level. This requires the event streams to be defined by the four characteristic functions $\eta^+(\Delta t), \eta^-(\Delta t), \delta^-(n)$ and $\delta^+(n)$. Note that $\eta^+(\Delta t)$ and $\eta^-(\Delta t)$ can be directly derived from $\delta^-(n)$ and $\delta^+(n)$:

$$\eta^+(\Delta t) = \max_{2 \le n \in \mathbb{N}} [\{n \mid \delta^-(n) < \Delta t\} \cup \{1\}] \quad (1)$$

$$\eta^-(\Delta t) = \min_{n \in \mathbb{N}_0} \{n \mid \delta^+(n+2) > \Delta t\} \quad (2)$$

Hence, in the following we will use $F = (\delta^-(n), \delta^+(n))$

We assume FIFO-buffers at the input of a task, and in-order processing of events by the tasks. Based on the worst case response time R^{max} and best case response time R^{min} obtained from the local scheduling analysis, the output stream calculation used to determine the output stream of analysed tasks can be modeled by an operation that defines a function T_τ, which calculates the function tuple F_τ modeling the output stream in the following way:

$$\begin{aligned} F_\tau = ((&\delta^-_\tau(n), \delta^+_\tau(n)) \mid \\ &\delta^-_\tau(n) = \max\{\delta^-_{in}(n) - (R^{max} - R^{min}), \\ &\qquad\qquad \delta^-_\tau(n-1) + R^{min}\}, \quad (3) \\ &\delta^+_\tau(n) = \{\delta^+_{in}(n) + (R^{max} - R^{min})\ ) \quad (4) \end{aligned}$$

In Equations 3 and 4 two different changes are applied to the timing of the event stream: First, the interval in which each event can occur is increased by $(R^{max} - R^{min})$, decreasing the minimum distance and increasing the maximum distance by the same amount. Second, a minimum inter arrival time of R^{min} between subsequent events is set, possibly increasing the distance between events. These changes obviously go in opposite directions. To separate these two steps we define a function for each one. The function $\vartheta(ES, j)$ decreases (increases) the minimum (maximum) distance between events of the event streams ES by j:

Definition 7. $\vartheta(ES, j)$
Consider an event stream ES_{in} defined by $F_{in} = (\delta^-_{in}(n), \delta^+_{in}(n))$. The function $\vartheta(ES_{in}, j)$ calculates the output model ES_ϑ defined by F_ϑ in the following way:

$$\begin{aligned} F_\vartheta = ((&\delta^-_\vartheta(n), \delta^+_\vartheta(n)) \mid \\ &\delta^-_\vartheta(n) = \max\{\delta^-_{in}(n) - j, 0\}, \\ &\delta^+_\vartheta(n) = \delta^+_{in}(n) + j\ ) \end{aligned}$$

The function $\sigma(ES, d)$ increases the minimum inter arrival time of two subsequent events of the event stream ES to d.

Definition 8. $\sigma(ES, d)$
Consider an event stream ES_{in} defined by $F_{in} = (\delta^-_{in}(n), \delta^+_{in}(n))$. The function $\sigma(ES_{in}, d)$ calculates the output model ES_σ defined by F_σ in the following way:

$$\begin{aligned} F_\sigma = ((&\delta^-_\sigma(n), \delta^+_\sigma(n)) \mid \\ &\delta^-_\sigma(n) = \max\{\delta^-_{in}(n), \delta^-_{in}(n-1) + d\}, \\ &\delta^+_\sigma(n) = \delta^+_{in}(n)\ ) \end{aligned}$$

Figure 4 illustrates the effects these two operations have, when applied to an event stream.

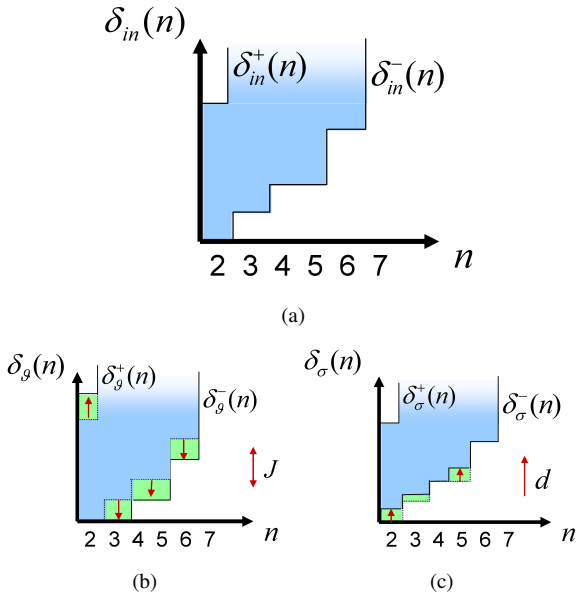

Figure 4. The δ-function modeling an event stream ES_{in} (a), the δ-function modeling the event stream $ES_\vartheta = \vartheta(ES_{in}, J)$ (b) the δ-function modeling the event stream $ES_\sigma = \sigma(ES_{in}, d)$ (c)

As can be seen in Figure 4b, applying the function $\vartheta(ES_{in}, J)$ shifts down the entire function $\delta^-(n)$ by J for all values of n. The function $\sigma(ES_{in}, d)$ shifts up the function $\delta^-(n)$ by d only for specific values of n, as depicted in Figure 4c. Essentially, $\sigma(ES, d)$ returns the same output stream as a sporadic shaper like defined in [6].

We can now also calculate the output stream ES_{out} of an analysed task τ by using these two functions.

Definition 9. *Stream Operation Θ_τ*
The stream operation Θ_τ that calculates the output stream of an analysed task defines the function T_τ as follows:

$$T_\tau(ES_{in}) = \sigma(\vartheta(ES_{in}, j), d) \tag{5}$$

Where $j = R^{max} - R^{min}$, $d = R^{min}$ and ES_{in} the input stream of task τ.

To be able to apply Equations 3 and 4 to output stream calculation, tasks may only have one input stream. In [4] two different activation semantics for tasks with multiple inputs were defined: AND-activation and OR-activation. In the following section we define corresponding HSCs and how the resulting HES can be deconstructed again.

5. Construction and Deconstruction of Hierarchical Event Streams

5.1. OR-Construction

First we define how the output stream of an OR-SC can be calculated.

Theorem 1. *An OR-SC combining m event streams to calculate the activating input stream of an OR-activated task, defines the function tuple F_{or} modeling its output stream:*

$$F_{or} = ((\delta^-_{or}(n), \delta^+_{or}(n)) \mid$$
$$\delta^-_{or}(n) = \begin{cases} 0: & n \leq m \\ \min\{\max_{\sum k_i = n}[\delta_i^-(k_i)]\}: & n > m \end{cases}$$
$$\delta^+_{or}(n) = \max\{\min_{\sum k_i = n-2}[\delta_i^+(k_i+2)]\}\ )$$
$$with\ \delta_i^-(1) \stackrel{!}{=} 0$$

Proof. We start with $n \leq m$: When we OR-combine m independent event streams, at least m events can occur simultaneously, one from each of the combined event streams. Hence the distance between the first m events is 0.

If $n > m$: To determine $\delta^-_{or}(n)$ we search for the smallest time interval Δt^- in which a total number of $n = \sum_i \eta_i^+(\Delta t^-)$ events can occur. We call $K = (k_1, ..k_m)$ with $k_i \in \mathbb{N}$ the contribution vector, that determines the number of events each event stream contributes to the total number n. Only contribution vectors with $\sum_{i=1}^m k_i = n$ are of interest. Hence, for a given contribution vector K,

$$\max_{\sum k_i = n}[\delta_i^-(k_i)], \quad with\ \delta_i^-(1) \stackrel{!}{=} 0 \tag{6}$$

gives the size of the time interval in which at least n events can occur. By evaluating Equation 6 for each possible contribution vectors and by taking the minimum of all, the searched minimum distance is obtained.

To determine $\delta^+_{or}(n)$, we search for the largest open time interval Δt^+ in which at least $n-2$ events must occur. Hence, we use a contribution vector $K = (k_1, ..k_m)$ with $k_i \in \mathbb{N}_0$ and $\sum_{i=1}^m k_i = n-2$. Since each open interval of size $\delta_i^+(k_i)$ contains at least $k_i - 2$ events of event stream ES_i,

$$\min_{\sum k_i = n-2}[\delta_i^+(k_i+2)] \tag{7}$$

gives the size of the time interval Δt^+ in which at most n events can occur according to the chosen contribution vector. Taking the maximum of all possible intervals of all possible contribution vectors leads to the searched maximum distance. □

Next, we must determine, how the *inner* streams can be defined. Observe the event sequences depicted in Figure 5. We denote $es_i^{\delta^-}$ the event sequence $[e_1, e_2, e_3, \ldots] = es_i \in ES_i$, where the distance between the event e_1 and e_k is given by $\delta_i^-(k)$. Figure 5 shows the two event sequences $es_a^{\delta^-} \in ES_a$ and $es_b^{\delta^-} \in ES_b$ in the upper part. In the lower part, the event sequence $es_{or}^{\delta^-} \in ES_{or} = \Omega_{or}(ES_a, ES_b)$ is depicted. As can be seen, the distance between events belonging to one of the event sequences in the resulting event sequence $es_{or}^{\delta^-}$ remains the same as in the individual streams. E.g. the distance between 2 events that belong to the event sequence $es_a^{\delta^-}$ equals $\delta_a^-(2)$ in both streams.

Since in general, the or-combination of several event streams doesn't change the timing of the event streams that are combined, we can define the *inner* streams in the following way:

$$F_i = ((\delta_i^-(n), \delta_i^+(n)) \mid \delta_i^-(n) = \delta_{i,in}^-(n) \; \delta_i^+(n) = \delta_{i,in}^+(n))$$

Now we must define how the *inner* event streams change,

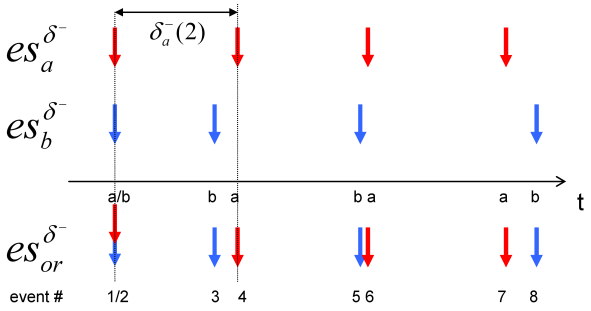

Figure 5. OR-combination of two event sequences

when an operation is applied to the *outer* event stream. For output model calculation of a task, we must consider the two operations: ϑ and σ. Accordingly, we define a function $B_\vartheta^{or}(ES_i, j)$ that is applied on each *inner* event stream when an operation $\vartheta(ES_{out}, j)$ is applied on the *outer* stream and a function $B_\sigma^{or}(ES_i, d)$ which is applied on each *inner* event stream when an operation $\sigma(ES_{out}, d)$ is applied on the *outer* stream. Figure 6 illustrates the event sequence $es_\vartheta^{\delta^-} \in ES_\vartheta$ resulting from applying the operation $\vartheta(ES_{or}, J)$ on the or-combined event stream from Figure 5. As can be seen, the distance between the event $e_1 \in es_\vartheta^{\delta^-}$ to all other events decreased by J. Obviously, the minimum distance between events that are either related to the event sequence $es_a^{\delta^-}$ or $es_b^{\delta^-}$ also decreased by J. Similar considerations lead to an increase in the maximum distance of

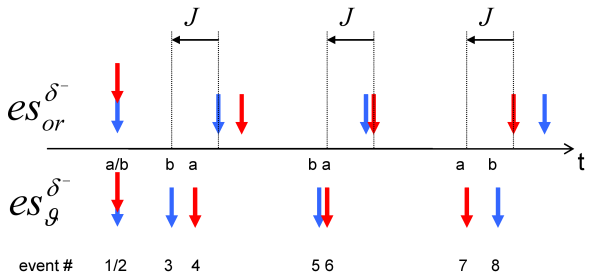

Figure 6. the event sequence $es_{or}^{\delta^-}$ and the event sequence $es_\vartheta^{\delta^-}$ after applying ϑ

events that are either related to the event streams ES_a or ES_b by J, if the operation $\vartheta(ES_{or}, J)$ is applied on the *outer* stream. Hence, we define $B_\vartheta^{or}(ES_i, j)$ in the following way:

Definition 10. *Inner Update Function* $B_\vartheta^{or}(ES_i, j)$
When an operation $\vartheta(ES_{out}, J) = ES'_{out}$ *is applied on the outer event stream, the inner update function* $B_\vartheta^{or}(ES_i, J) = ES_i'$ *is applied on each inner event stream.*

$$ES_i' = B_\vartheta^{or}(ES_i, J) = \vartheta(ES_i, J)$$

Let us now consider, that we apply the operation $\sigma(ES_\vartheta, d)$ on the event stream ES_ϑ. Figure 7 shows the event sequence $es_\vartheta^{\delta^-} \in ES_\vartheta$ and the resulting output sequence $es_\sigma^{\delta^-} \in ES_\sigma$, according to δ_σ^-.

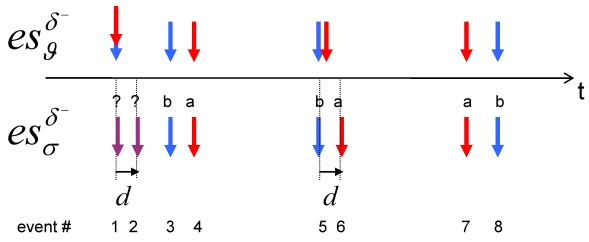

Figure 7. the event sequence $es_\vartheta^{\delta^-}$ and the event sequence $es_\sigma^{\delta^-}$ after applying σ

As can be seen, the distance between the event e_2 to the following events decreased by d. An Exception is event e_6, which in the illustrated example also suffers a delay caused by the new minimum inter arrival time. Remember that the depicted event sequence $es_\vartheta^{\delta^-} \in ES_\vartheta$ is only one possible event sequence among many others that are contained in ES_ϑ. There might be an event sequence $es \in ES_\vartheta$, where the event $e_5 \in es_\vartheta^{\delta^-}$ arrives later than the event $e_6 \in es_\vartheta^{\delta^-}$. Then, the event labeled e_6 in Figure 7 wouldn't be delayed and the minimum distance between events that are related to the event stream ES_a just decreases by d.

So, to obtain the new minimum distance $\delta_i^{'-}$ between n events that are related to a specific event stream ES_i in general, we assume the maximum possible delay D^{max} for the first event and no delay for the nth event. This leads to:

$$\delta_i^{'-}(n) = \max\{\delta_i^-(n) - D^{max}, 0\} \tag{8}$$

To obtain the new maximum distance $\delta_i^{'+}$ between n events that are related to a specific event stream ES_i, we assume the maximum possible delay D^{max} for the nth event of the sequence defined by $\delta_i^+(n)$ and no delay for the first event:

$$\delta_i^{'+}(n) = \delta_i^+(n) + D^{max} \tag{9}$$

The maximum delay D^{max} an event can suffer, when an operation $\sigma(ES_{in}, d) = ES_{out}$ is applied on the event stream ES_{in} is given by:

Theorem 2.

$$D^{max} = \max_{i>1}\{\delta_{out}^-(i) - \delta_{in}^-(i), 0\} \tag{10}$$

Proof. See delay calculation for a sporadic shaper as defined in [6]. □

After using Equations 8 and 9 to modify the *inner* streams, we have to account for the new minimum distance. We can do this by applying an operator σ on the resulting stream. This leads to the following definition of the function $B_\sigma^{or}(ES_i, d)$:

Definition 11. *Inner Update Function $B_\sigma^{or}(ES_i, d)$*
When an operation $\sigma(ES_{out}, d) = ES'_{out}$ is applied on the outer event stream, the inner update function $B_\sigma^{or}(ES_i, d) = ES'_i$ is applied on each inner event stream.

$$ES'_i = B_\sigma^{or}(ES_i, d) = \sigma(\vartheta(ES_i, D^{max}), d) \tag{11}$$

We now define the HSC Ω_{or} as follows:

Definition 12. Ω_{or}
The HSC Ω_{or} defines the function HSC_{or}:

$$\begin{aligned}
HSC_{or}(F_{in,1}, .., F_{in,n}) &= ((F_{out}, L, C) \mid \\
F_{out} &= ((\delta_{out}^-(n), \delta_{out}^+(n)) \mid \delta_{out}^-(n) = \min\{\max_{\sum k_i = n}[\delta_i^-(k_i)]\}, \\
&\qquad \delta_{out}^+(n) = \max\{\min_{\sum k_i = n-2}[\delta_i^+(k_i + 2)]\}), \\
L &= ((F_1, .., F_n) \mid F_i = ((\delta_i^-(n), \delta_i^+(n)) \mid \\
&\qquad \delta_i^-(n) = \delta_{in,i}^-(n) \\
&\qquad \delta_i^+(n) = \delta_{in,i}^+(n)\ )) \\
C_{or} &= ((B_\vartheta^{or}, B_\sigma^{or}) \mid B_\vartheta^{or}(ES_i, J) = \vartheta(ES_i, J), \\
&\qquad B_\sigma^{or}(ES_i, d) = \sigma(\vartheta(ES_i, D^{max}), d)\ ))
\end{aligned}$$

5.2. AND-Construction of Event Streams

An event stream constructor combining m event streams $ES_1, .., ES_m$ to calculate the activating input stream of an AND-activated task, defines the function tuple F_{and} modeling its output stream:

$$\begin{aligned}
F_{and} = &((\delta_{and}^-(n), \delta_{and}^+(n)) \mid \\
&\delta_{and}^-(n) = \min[\delta_i^-(n)] && (12) \\
&\delta_{and}^+(n) = \max[\delta_i^+(n)]\ ) && (13)
\end{aligned}$$

Proof. See [4]. □

Note: To ensure bounded AND-buffer sizes the combined event streams must fulfill the following requirements [4]:

1. For $\lim_{\Delta t \to \infty}$, the difference between the minimum and maximum number of input events at input i must be finite.
2. For $\lim_{\Delta t \to \infty}$, the difference between the number of input events at different inputs must be finite.

Since each event of ES_{out} is related with one event of each combined event stream, it directly follows that, the minimum distance δ_i^- between n events that are related to the event stream $ES_{in,i}$ is given by the minimum distance δ_{out}^- of the *outer* stream:

$$\begin{aligned}
F_i = &((\delta_i^-(n), \delta_i^+(n)) \mid \\
&\delta_i^-(n) = \delta_{out}^-(n), && (14) \\
&\delta_i^+(n) = \delta_{out}^+(n)\ ) && (15)
\end{aligned}$$

It also follows, that regardless of changes applied to the *outer* stream, Equations 14 and 15 still reflect the timing of the *inner* streams. Hence, we can define the inner update functions $B_\vartheta^{and}(ES_i, J)$ and $B_\sigma^{and}(ES_i, d)$ in the following way:

Definition 13. *Inner Update Function $B_\vartheta^{and}(ES_i, J)$*
When an operation $\vartheta(ES_{out}, J) = ES'_{out}$ is applied on the outer event stream of an and-combined HES, the inner update function $B_\vartheta^{and}(ES_i, J) = ES'_i$ is applied on each inner event stream resulting in ES'_i.

$$ES'_i = B_\vartheta^{and}(ES_i, J) = \vartheta(ES_i, J) \tag{16}$$

Definition 14. *Inner Update Function $B_\sigma^{and}(ES_i, d)$*
When an operation $\sigma(ES_{out}, d) = ES'_{out}$ is applied on the outer event stream of an and-combined HES, the inner update function $B_\sigma^{and}(ES_i, d) = ES'_i$ is applied on each inner event stream resulting in ES'_i.

$$ES'_i = B_\sigma^{and}(ES_i, d) = \sigma(ES_i, d) \tag{17}$$

We can now define the HSC Ω_{and} as follows:

Definition 15. Ω_{and}
The HSC Ω_{and} defines the function HSC_{and}:

$$\begin{aligned} HSC_{and}(F_1,..,F_n) = ((F_{out}, L, C_{and}) \mid \\ F_{out} = ((\delta^-_{out}(n), \delta^+_{out}(n)) \mid \delta^-_{out}(n) = \min[\delta^-_{in,i}(n)]\}, \\ \delta^+_{out}(n) = \max[\delta^+_{in,i}(n)]), \\ L = ((F_1,..,F_n) \mid F_i = ((\delta^-_i(n), \delta^+_i(n)) \mid \\ \delta^-_i(n) = \delta^-_{out}(n), \\ \delta^+_i(n) = \delta^+_{out}(n)\ )), \\ C_{and} = ((B_\vartheta, B_d) \mid B^{and}_\vartheta(ES_i, J) = \vartheta(ES_i, J), \\ B^{and}_\sigma(ES_i, d) = \sigma(ES_i, d)\ )) \end{aligned}$$

5.3. HES with multiple layers

The defined HSCs can also be used with HES as input streams as illustrated in Figure 3, whereas the *outer* streams are taken as input streams for the HSC. Since we defined the inner update functions using the operations ϑ and σ, it is assured, that changes to the top most event stream for which a corresponding inner update function is defined, can be propagated through all hierarchical layers to the event streams of the lowest layer.

As the alert reader may already have noticed, the definition of Ω_{and} brings some limitations. In contrast to the HSC Ω_{or}, the HSC Ω_{and}, changes the timing of the combined event streams as soon as it is applied (see Equations 14 and 15). The problem is, that we haven't defined an appropriate inner update function that defines how the inner streams are effected if one of the and-combined streams is an HES. Hence, as soon as we use an HSC Ω_{and} with a HES as input stream, we loose the specific information about the inner streams of this HES.

From there it follows that for this contribution, an AND-constructed hierarchical event stream can only be on the lowest hierarchical layer of an HES with multiple hierarchical layers.

5.4. Deconstruction of Hierarchical Event Streams

The way we defined hierarchical event models and the HSCs, the deconstruction of hierarchical event models turns out very simple. Remember that we update the *inner* streams of an HES every time an operation is applied on the *outer* stream of the HES. Hence, as long as we don't want to consider additional timing effects imposed on the *inner* streams by the extraction process, the extracted event streams are directly given by the updated *inner* streams.

To deconstruct a hierarchical event stream, we define the hierarchical deconstructor Ψ:

Definition 16. Ψ
The hierarchical deconstructor Ψ applied on an hierarchical event stream EM_h modeled by H defines the function D_Ψ:

$$D_\Psi(H) = ()(F_1,..,F_n) \mid F_i = L(i))$$

where $L(i)$ returns the i-th element of the list L that is contained in H.

The deconstructor essentially returns one element of the List of *inner* streams of the hierarchical model.

6. System Example

Now we will use hierarchical event models to analyse the example system shown in Figure 8, which is similar to the motivational example system. We just added some additional bus communication in form of C1, which will impose some scheduling effects on C0 and we also added another CPU (CPU3) running the two tasks T5 and T6. We want to consider the following activation rules for the tasks T3, T4, T5 and T6: The task T3 is only activated by events that were generated by the task T1 and T4 is only activated by events generated by the task T2. The task T5 is only activated by events generated by S0 and the task T6 is only activated by events generated by S1.

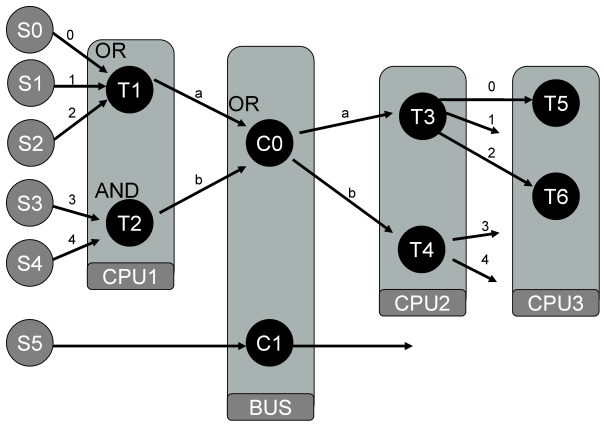

Figure 8. A hypothetical example System

Since for each event stream we can determine the function tuple $F = (\delta^-(n), \delta^+(n), \eta^+(\Delta t), \eta^-(\Delta t))$ we are not only able reuse traditional scheduling techniques, we can also easily integrate our new Model in the tool SymTA/S [8] and reuse the therein implemented analysis techniques to obtain the parameters of the stream operations Θ. Relevant system parameters like core execution time (CET), core communication time (CCT), priorities and timing of the sources are summarized in Tables 1 - 5.

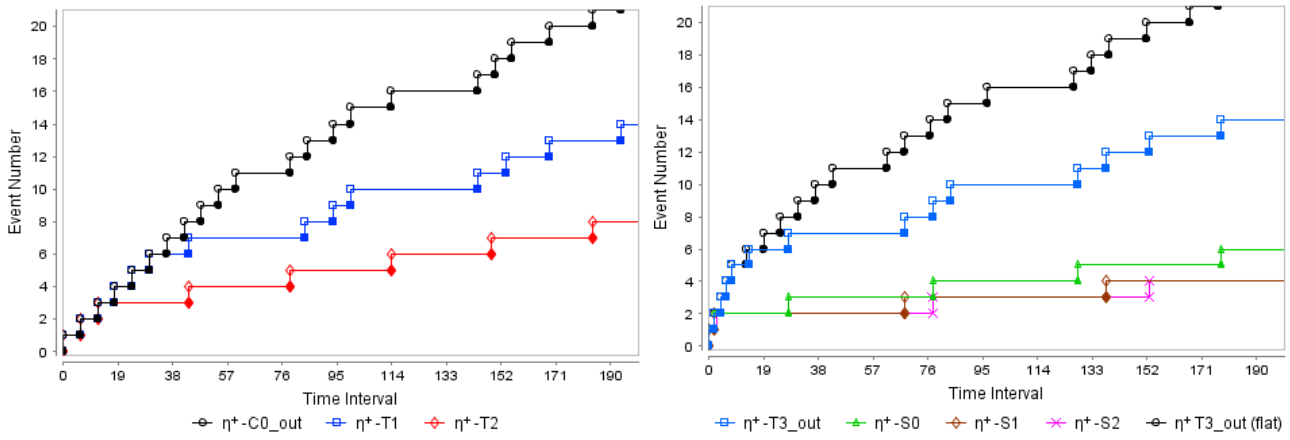

Figure 10. η^+-functions from the HES at the output of C0(a) and the η^+-functions of the HES at the output of T3(b)

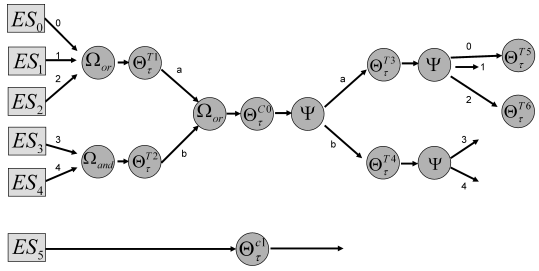

Figure 9. The system graph of the example system consisting of hierarchical constructors, stream operations and hierarchical deconstructors

For the CAN bus, we consider that for every event arrival at the input of $C0$ or $C1$, a whole CAN Frame is send.

Figure 9 shows the abstract system model for the hypothetical example system from Figure 8 consisting of the previously defined HSCs, stream operations and deconstructors. The Figures 10 (a) and (b) visualize the obtained η^+-functions of the *outer* and *inner* streams of the hierarchical output streams of C0 and T3.

The η^+-function of the *outer* event stream of the hierarchical output stream of C0 is depicted in Figure 10 (a) (marked by black cycles), together with the η^+-functions of the two *inner* streams of the hierarchical output stream of C0. These two *inner* streams bound the number of events that are related to the tasks T1 (marked by red squares) and T2 (marked by red diamonds). Since we want to consider that the task T3 is only activated by events that originated form T1, we use the corresponding *inner* event stream as input stream for T3. Using flat event streams, at the output of C0 we wouldn't have any information about the individ-

Table 1. Sources

Source	Period	Jitter
S0	50	0
S1	70	0
S2	75	0
S3	35	40
S4	35	15
S5	90	70

Table 2. CPU1 (SPP - Scheduled)

Task	CET	Priority
T1	[2, 3]	Low
T2	[3, 4]	High

Table 3. Bus (CAN)

Channel	CCT	Priority
C0	[9.2, 11.2]	High
C1	[6.8, 9.2]	Low

Table 4. CPU2 (SPP - Scheduled)

Task	CET	Priority
T3	[2, 3]	Low
T4	[4, 5]	High

Table 5. CPU3 (SPP - Scheduled)

Task	CET	Priority	R^+ SEM	R^+ HEM	Red.
T5	[2, 3]	Low	31	6	80.6%
T6	[2, 3]	High	3	3	0%

ual timings of events that came from T1 and T2. Therefore, we could only bound the activations of the receiving tasks by the total number of events produced at the output of C0, which is given by the *outer* stream.

Using the *inner* stream modeling the events that were send by T1 as input for T3, at the output of T3 we obtain a hierarchical event stream with the η^+-functions depicted in Figure 10(b) bounding its *outer* and *inner* streams. The function marked by blue squares represents the maximum number of events of the *outer* stream of the hierarchical stream at the output of the task T3, which models all events that can be output by T3 in a given time interval. The lower functions (green triangles, brown diamonds and purple crosses) represent the maximum number of events of the *inner* streams, which model the events, that originated from one of the sources S0-S2. The curve marked by black cycles represents the maximum number of events that would be obtained at the output of T3, if we used the total number of output events of C0 as activating events (which would be the case using flat event streams). Without the ability to extract the *inner* stream properties, we could again only use η^+-out (flat) (the black curve) to bound the activations of connected tasks, here for the tasks T5 and T6.

In both cases we would obviously obtain much more pessimistic analysis results. E.g., as visualized in Figure 11, the utilization of CPU2 is 60.95% and the utilization of CPU3 is 45.71%, when using flat event streams. Using HEMs, the utilization drops down to 28.57% for CPU2 and 10% for CPU3. So, with flat event streams the utilization is overestimated by more than a factor of 2 for CPU2 and by more than a factor of 4.5 for CPU3, compared to the utilization obtained using the concept of HES.

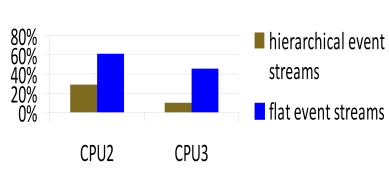

Figure 11. The utilization of CPU2 and CPU3 using flat event streams vs. HES

With the existing standard event models (SEM), the tool SymTA/S normally uses, the system analysis is much faster, but in addition to an overestimated utilization, the obtained worst case response times (WCRT) become even more conservative in comparison to the WCRT obtained with HEMs (as can be seen in Table 5). When using SEMs we are not only incapable of unpacking the different streams on the receiving side, but also the approach presented in [4] to calculate the activating event stream of a task with several input streams introduces further pessimism.

7. Conclusion

In this paper we have introduced hierarchical event streams constructed from several independent streams and defined hierarchical event models to capture the properties of such streams.

When hierarchical streams are used in an compositional analysis approach, response time analysis is only applied to the combined stream while inner update functions derive the corresponding changes to the embedded individual streams.

We also showed that hierarchical event streams can also contain other hierarchical event streams, forming several hierarchical layers. We defined the functions needed enable to construct and deconstruct hierarchical event streams.

We integrated the presented model into an existing analysis frame work and proofed its applicability by analysing an example system using existing implementations of scheduling analysis techniques.

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. In *Proceedings 18th Euromicro Conference on Real-Time Systems*, page 97106, 2006.

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

[3] K. Gresser. An event model for deadline verification of hard real-time systems. In *Proceedings 5th Euromicro Workshop on Real-Time Systems*, pages 118–123. Oulu, Finland, 1993.

[4] M. Jersak. *Compositional Performance Analysis for Complex Embedded Applications*. Phd thesis, Technical University of Braunschweig, 2004.

[5] J. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Proc. of the Real-Time Systems Symposium*, pages 201–209, 1990.

[6] K. Richter. *Compositional Scheduling Analysis Using Standard Event Models*. Phd thesis, Technical University of Braunschweig, 2004.

[7] I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees,. In *Proceedings of the Real-Time Systems Symposium (RTSS)*, page 213, 2003.

[8] Symtavision. `http://www.symtavision.com`.

[9] L. Thiele, S. Chakraborty, M. Gries, and S. Kunzli. A framework for evaluating design tradeoffs in packet processing architectures. In *Proc. 39th Design Automation Conference (DAC)*, 2002.

[10] L. Thiele, S. Chakraborty, and M. Naedele. Realtime calculus for scheduling hard real-time systems. In *Proceedings International Symposium on Circuits and Systems (ISCAS)*. Geneva, Switzerland, 2000.

[11] E. Wandeler. *Modular Performance Analysis and Interface-Based Design for Embedded Real-Time Systems*. Phd thesis, SWISS FEDERAL INSTITUTE OF TECHNOLOGY ZURICH, 2006.