Modeling Event Stream Hierarchies with Hierarchical Event Models

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 generalize the concept of a stream hierarchy to embed different types of streams in a higher level structure. We explain why this extension is a natural match to model streams generated by communication stacks that are ubiquitous in networked embedded systems. We formally define the hierarchical event model and give operations to encode, combine, and extract stream properties that can be used in flat or hierarchical local scheduling analysis. Finally, we give an example and demonstrate that the proposed model enables 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.

Currently, the use of simulation based methods for performance estimation is the state of the art in industry and several commercial simulation suites are available, as well as open simulation frameworks. The main advantage of simulation is the large modelling scope, as various dynamic and complex interactions can be taken into account. However, most simulation based performance estimation methods suffer from insufficient corner case coverage. Hence, several different abstractions for formal performance analysis have emerged, which allow to derive worst case guarantees for the timing behaviour of distributed systems.

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

Based on event streams, two different compositional approaches for system level performance analysis have been developed [7][11], the former meanwhile used in industrial practice [9]. Common to these 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. For global system analysis, it is then iterated 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.

While local analysis has successfully been extended to include hierarchical scheduling strategies[8][10], event streams are still flat. A recent proposal [1] introduces event hierarchies in a stream, still describing a single flat event stream. Such flat event streams prove to be improper to accurately capture timing effects occurring in modern communication stacks.

Contributions of this work:

- We define hierarchical event streams that allow to embed different types of streams in a higher level structure and propose a hierarchical event model (HEM) to model such hierarchical streams.
- We present how hierarchical event models can be applied to model the timing of event streams generated by a modern communication stack.
- We integrate the proposed model into an existing system analysis tool and show the improvements that can be obtained when accounting for the presence of a communication stack.

The remainder of this paper is structured as follows.

978-3-9810801-3-1/DATE08 © 2008 EDAA

First, we will review related work (section 2). We then give formal definitions of the existing system models and its elements and introduce stream hierarchies and define a hierarchical event model to capture these stream hierarchies in the analysis model (section 3). In section 4 we introduce the basic properties of the AUTOSAR COM-Layer as a practical application scenario were such stream hierarchies can be observed. Afterwards, we formally derive the needed operations to analyse such a COM-Layer (section 5). Finally, we use the new event model for analysing a distributed embedded example systems to evaluate its applicability (section 6) and draw conclusions (section 7).

2. Related Work

The analysis framework presented by Richter [7] 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 Lehocky [6] are used. Standard event models (SEM), 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. [11] uses numerical upper and lower event arrival curves for describing load generated by event streams, and similar service curves for execution modeling. Based on [3], 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 [10] 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 [4] 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.

Neither of the different event models presented so far allow to appropriately capture properties of hierarchical event streams needed to account for the presence of communication stacks in system performance analysis.

3. 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 input streams to output streams. 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.

Since the operation used in [7] and [12] to model the processing of an event stream by a task requires the task to have only one input event stream, tasks activated by multiple event streams are decomposed in two operations [5][12]: the first is an event stream constructor (SC), which combines the input streams, calculating 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 1 illustrates a simple system and the corre-

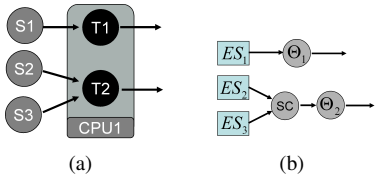

Figure 1. A simple example system (a) and the corresponding abstract system model (b)

sponding model used for performance analysis, consisting of operations and event streams.

Similar to the framework presented by Richter [7] 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 $\delta^+(n)$.

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

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

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

The output stream calculation used to determine the output stream of analysed tasks can be modeled by an operation Θ_τ that defines a function T_τ, which calculates the function tuple F_τ modeling the output stream in the following way:

$$\begin{aligned} F_\tau = (&\delta_\tau^-(n), \delta_\tau^+(n) \mid \\ &\delta_\tau^-(n) = \max\{\delta_{in}^-(n) - (r^+ - r^-), \delta_\tau^-(n-1) + r^-\} \\ &\delta_\tau^+(n) = \delta_{in}^+(n) + (r^+ - r^-)) \end{aligned}$$

Where r^- is the minimum response time and r^+ the maximum response time of the task. Similar, functions defining the output streams of other operations like e.g. shapers [7] can be defined.

In [5] two different activation semantics for tasks with multiple inputs were defined: AND-activation and OR-activation. In the following we will take a closer look at the OR-activation, since we will reuse some parts of it later.

An operation 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:

$$\begin{aligned} F_{or} = &(\delta_{or}^-(n), \delta_{or}^+(n) \mid \\ &\delta_{or}^-(n) = \min\{\max_{\sum k_i = n} [\delta_i^-(k_i)]\} \quad (3) \\ &\delta_{or}^+(n) = \max\{\min_{\sum k_i = n-2} [\delta_i^+(k_i + 2)]\}) \quad (4) \end{aligned}$$

Proof. To determine $\delta_{or}^-(n)$ we search for the smallest closed time interval Δt in which a total number of $n = \sum_i \eta_i^+(\Delta t)$ events can occur. We call $K = (k_1, ..k_m)$ 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, $\max_{\sum k_i = n}[\delta_i^-(k_i)]$ gives the size of time the interval in which n events must occur. Taking the minimum of all possible intervals leads to the searched minimum distance.

To determine $\delta_{or}^+(n)$, we search for the largest open time interval Δt in which at most $n-2$ events can occur. Hence, we use a contribution vector $K = (k_1, ..k_m)$ with $\sum_{i=1}^m k_i = n - 2$. Since each open interval of size $\delta_i^+(k_i)$ contains at most $k_i - 2$ events of event stream ES_i, $\min_{\sum k_i = n-2}[\delta_i^+(k_i + 2)]$ gives the size of the time interval Δt in which at most $n-2$ 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. □

Since the output event stream of the existing AND-combination and OR-combination is modeled by a single function tuple F, it contains no information about the timing of the combined event streams.

Therefore we extend the system model by hierarchical event streams (HES) and hierarchical stream constructors (HSC). The idea is, that a hierarchical event stream has one *outer* representation in form of an event stream and for each combined event stream it has one *inner* representation, also in form of an event stream. The relation between the *outer* event stream and the *inner* event streams depend on the HSC that combined the event streams.

Definition 3. *Hierarchical Event Stream*
A hierarchical event stream ES_h is the result of the combination of n input event streams $ES_1, .., ES_n$ and has one outer event stream ES_{out}, defined by a function tuple $F_{out} = (f_1, .., f_n)$ and k inner event streams, each defined by its own function tuple $F_i = (f_1^i, .., f_n^i)$.

Definition 4. *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:

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

Definition 5. *Hierarchical Event Model*
A hierarchical event model defines a parameter tuple H:

$$\begin{aligned} H = \{(F_{out}, L, C_\Omega) \mid &F_{out} = (f_1, .., f_n), \\ &L = (F_1, .., F_n), \\ &C = C_\Omega\} \end{aligned}$$

$F_{out} = (f_1, .., f_n)$ models the outer event stream.
$L = (F_1, .., F_n)$ is a List of function tuples modeling the inner streams.
$C = C_\Omega$ is the construction rule that combines the inner event models to the hierarchical model.

Note: for each event stream constructor generating the output stream F_{sc} a corresponding hierarchical event stream constructor can be defined that generates a hierarchical event stream with an outer event stream modeled by $F_{out} = F_{sc}$.

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:

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

Existing stream operations, like e.g. Θ_τ, assume flat event streams as input and they also generated flat event streams as output. Since we want to reuse them, we must define how they can be applied to hierarchical event streams. If an operation Θ is applied on a hierarchical event stream, the outer event stream is used as input stream and the output stream generated by the operation Θ becomes the outer event stream of the resulting hierarchical event stream. To determine the effects the operation has on the inner event streams, we define the inner update function:

Definition 7. *Inner Update Function*
For each hierarchical event model H with constructor C_Ω, the inner update function B adapts the F_i 's if the outer event model is modified by an operator Θ. Formally:

$$B_{\Theta,C_\Omega} : F \times H \rightarrow F$$

4. Application Scenario

The properties we introduce in the following are part of specification of the communication layer defined by the AUTOSAR consortium [2]. This is a complicated combination of event and register communication used to handle different signal latency requirements. Observe the system shown in figure 2. If we consider the presence of a communication stack, the events generated by the sources $S1, S2, S3$ and $S4$ don't trigger the communication directly. Instead they write their output data into a register provided by the communication layer, overwriting previous output data. Each register is assigned a fixed position in a so called frame. The communication layer triggers the sending of a frame, which then transmits all register values assigned to that frame. A frame can be of different types: *periodic*, *direct* or *mixed*. An input event stream of a frame can be either defined *triggering* or *pending*. The sending of frames is triggered according to the following rules: When the frame type is *periodic*, frames are just send periodically, not influenced by the arrival of the output events of the tasks. If the frame type is *direct*, for each arrival of a *triggering* signal, a frame is send. And finally a mixed frame is a combination of the two first ones, so it is transmitted periodically and also whenever a *triggering* event arrives.

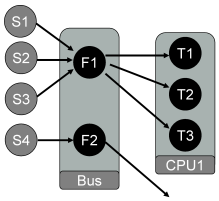

Figure 2. The system example

When a frame is received by the communication layer, it transmits the data in it into registers, again overwriting the values stored there previously. Either does the receiving task fetch the register value from time to time or each time new data is written into the register, the process is activated, e.g. by an interrupt.

To be able to analyse the bus, we first need the frame timing. Since the frame is activated whenever a *triggering* signal arrives, its equivalent to an OR-activation by all *triggering* signals, where a timer is treated as an additional *triggering* signal. Hence, we can reuse equations (3) and (4) to calculate the activation timing of the frames.

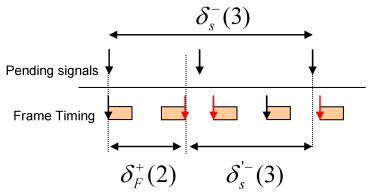

Figure 3. Frame timing vs signal timing

The second thing we will need later is the distance between frames, $\delta_i^{'-}(n)$ and $\delta_i^{'+}(n)$, that transport signals from a specific event stream ES_i. For *triggering* signals this is quite simple, since for each signal the frame is immediately activated. From there it follows that:

$$\delta_i^{'-}(n) = \delta_i^-(n) \quad (5)$$

$$\delta_i^{'+}(n) = \delta_i^+(n) \quad (6)$$

To determine the minimum distance between frames that transport n signals of a *pending* event stream, consider the scenario depicted in figure 3, where the red colored events represent frame activations that will also transmit a new *pending* signal. If we assume, that the first of the n events of the *pending* event stream just misses a frame activation, then the longest time that can pass before a new frame activation occurs is given by the maximum distance between 2 frames: $\delta_f^+(2)$. By assuming that the n-th event of the *pending* stream is transmitted immediately (although that may not be possible as in the example in figure 3), we obtain a conservative lower bound. Since each frame carries at most one signal of each assigned stream, another obvious bound is the minimum distance between n frames. Hence,

for the *pending* streams we obtain:

$$\delta_i^{'-}(n) = \max\{\delta_i^-(n) - \delta_f^+(2)), \delta_f^-(n)\} \quad (7)$$

$$\delta_i^{'+}(n) = \infty \quad (8)$$

5. Hierarchical Event Model

In this section we will define a HSC and an inner update function, corresponding to the properties of the communication layer as introduced in the previous section.

5.1 Packing the Frames

To build the hierarchical model of the hierarchical stream like it is generated by the previously introduced communication layer, we define a "pack"-HSC Ω_{pa}, reusing the equations from Ω_{or} and equations (5) - (8).

Definition 8. Ω_{pa}
A "pack"-HSC Ω_{pa} defines the hierarchical event model H as follows:

$$H = \{(F_{out}, L, C) \mid F_{out} = ($$

$$\delta_{out}^-(n) = \min_{ES_i \in T}\{\max_{\sum k_i = n}[\delta_i^-(k_i)]\}$$

$$\delta_{out}^+(n) = \max_{ES_i \in T}\{\min_{\sum k_i = n-2}[\delta_i^+(k_i+2)]\})$$

$$L = \{F_1', .., F_n'\} \mid F_i' = (\delta_i^{'-}(n), \delta_i^{'+}(n) \mid$$

$$\delta_i^{'-}(n) = \begin{cases} \delta_i^-(n) & : ES_i \in T \\ \max\{\delta_i^-(n) - \delta_{out}^+(2)), \delta_{out}^-(n)\} & : ES_i \notin T \end{cases}$$

$$\delta_i^{'+}(n) = \begin{cases} \delta_i^+(n) & : ES_i \in T \\ \infty & : ES_i \notin T \end{cases}$$

$$C = \Omega_{pa}\}$$

Where T is the set of all event streams that model triggering signals.

5.2 Analysing the Bus

To determine the output model of the frame, we use Θ_τ to determine the outer event stream. To adopt the inner event streams of an hierarchical stream constructed by a HSC Ω_{pa} we have to consider two things: First, the minimum distance between the events decreases by $(r^+ - r^-)$. Second, each event is separated by at least r^- from subsequent events. Hence, each event that previously arrived simultaneous with other events could be delayed by $(k-1) * r^-$, where k is the number of simultaneous event arrivals. With the result that the minimum distance of the delayed event to subsequent events is further reduced by $(k-1) * r^-$. Similar, the maximum distance to previous events increases by r^-. This leads to the following definition of the inner update function:

Definition 9. $B_{\Theta_\tau, C_{pa}}$

$$F_i' = (\delta_i^{'-}(n), \delta_i^{'+}(n) \mid$$
$$\delta_i^{'-}(n) = \max\{\delta_i^-(n) - (r^+ - r^-) - (k-1) * r^-,$$
$$(n-1) * r^-\},$$
$$\delta_i^{'+}(n) = \delta_i^+(n) + (r^+ - r^-) + (k-1) * r^-)$$

Where r^+ and r^- are given by Θ_τ and k is the maximum number of events of ES_{out} (before the operation was applied) that can be effected by the new minimum distance.

5.3 Unpacking the Signals

The way we defined hierarchical event models and the HSCs, makes the deconstruction of hierarchical event models turn out very simple. For extraction, we define the hierarchical deconstructor Ψ_{pa} that is applied on the hierarchical output event stream ES_h of the frame:

Definition 10. Ψ_{pa}
The hierarchical deconstructor Ψ_{pa} applied on an hierarchical event stream ES_h modeled by H defines the event stream ES_i by the following function:

$$F_i = L(i)$$

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

6. System Example

In this section, we apply hierarchical event models to analyze the performance of the system illustrated in figure 2. We extended the tool SymTA/S [9], which normally uses standard event models, with the presented hierarchical models. Since HEMs can be characterized by the four characteristic functions, similar to SEMs, the different local scheduling analysis techniques implemented in SymTA/S can directly be reused.

Relevant system parameters like core execution time (CET) and priorities are summarized in tables 1-3. For the bus we assume, that it is a CAN bus and the tasks on the CPU are scheduled according to a static priority preemptive (SPP) scheduling policy.

Table 1. Sources

Source	Period	Type
S1	250	triggering
S2	450	triggering
S3	1200	pending
S4	400	triggering

Table 2. Bus (CAN - Scheduled)

Frame	Payload size	priority
F1	[4:4]	High
F2	[2:2]	Low

Table 3. CPU (SPP - Scheduled)

Task	CET	Prio	R^+ flat	R^+ HEM	Red.
T1	[24:24]	High	24	24	0%
T2	[32:32]	Med	104	56	46,2%
T3	[40:40]	Low	266	96	63,9%

Figure 4 shows the η^+-functions of the output event stream of the frame $F1$ and of the input event streams of $T1$ - $T3$ when using hierarchical event models. The graph marked with black points pictures the maximum number of total frame arrivals for a given time interval. The other three depicted functions were obtained after unpacking the corresponding signals. The function marked with red squares bounds the maximum number of event arrivals for $T1$, the function marked by the blue squares does the same for $T2$ and the function marked with the green triangles bounds the maximum number of events activating $T3$.

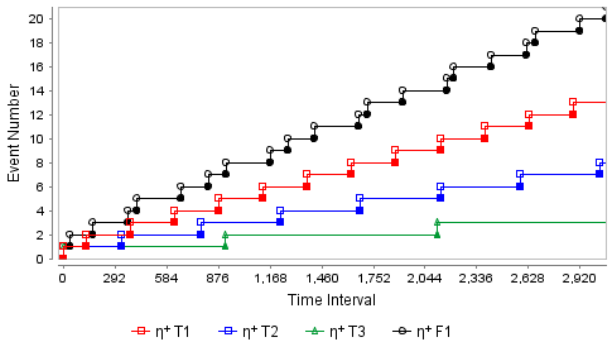

Figure 4. The η^+-functions of $T1$-$T3$ and $F1$

Obviously, using the functions obtained by unpacking the corresponding signals to bound the input event streams of $T1$, $T2$ and $T3$ leads to much less overestimation than using the function for total frame arrivals. This in turn leads to much more accurate analysis results for CPU1. Table 3 shows the obtained worst case response time (WCRT) R^+ when using flat event streams (standard event models) and when using HEMs. The last column denotes the reduction of the obtained WCRT in percent.

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. These hierarchical event models allow to accurately model and analyze the processing and communication on the combined as well as on the embedded individual streams. We define construction and deconstruction functions that enable the transition between hierarchical and flat streams. Response time analysis is only applied to the combined stream while an inner update function derives the corresponding changes to the embedded individual streams.

We applied the model to an automotive communication stack and explained the modeling of packing, unpacking, and transport of frames that transport signals from different sources. We also demonstrated, with a simple example, the significant improvements that can be obtained, exploiting the stream hierarchies.

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] AUTOSAR. Autosar specification of communication v. 2.0.1, autosar partnership, 2006.

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

[4] 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.

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

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

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

[8] I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees,. In *Proceedings of the Real-Time Systems Symposium (RTSS)*, pages 2–13, 2003.

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

[10] 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.

[11] 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.

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