Advanced Hierarchical Event-Stream Model and the Real-Time Calculus

Karsten Albers, Steffen Kollmann, Frank Bodmann, Frank Slomka

Embedded Systems / Real-Time Systems, University of Ulm

{surname.name}@uni-ulm.de

6. March 2008

Abstract - Analyzing future distributed real-time systems, automotive and avionic systems, requires compositional real-time analysis techniques. Well known established techniques as Sympta/S and the real-time calculus are candidates for solving the mentioned problem. However both techniques use quite simple event models. Sympta/S is based on discrete events the real-time calculus on continuous functions. Such simple models has been chosen because of the computational complexity of the considered mathematical operations required for real-time analysis. Advances in approximation techniques are allowing the consideration of more expressive descriptions of events. In this paper such a new expressive event model and its analysis algorithm are described. It integrates the models of both techniques. This also allows to propagate the approximation through the analysis of a distributed system leading to a much more efficient analysis.

We will also show the integration of the the hierachical event-stream model and therefore the event driven real-time analysis, the periodic, and the sporadic task model with the real-time calculus. For the event-driven real-time analysis, flexible approximative analysis approaches are proposed to allow an efficient real-time analysis. We will provide an easy but powerful approximative description model for the real-time calculus. In contrary to the existing description model the degree of approximation is choosable allowing a more accurate description.

1. Motivation

The module-based design processes make it possible to handle the complexity in software and hardware design. Systems are built using a set of closed modules. These modules can be designed and developed separately. They have only designated interfaces and connections to other modules of their set. The purpose of modularisation is to split the challenging job of designing the whole system into multiple smaller jobs, allow the reuse of modules in different designs or to include IP components of third-party vendors.

Every module-based design concept requires a well defined interface-concept for connecting the modules. Developing real-time systems requires for this interface-concept also to cover the real-time aspects of the modules. A concept for the real-time analysis is required to handle the modules separatly and allows a propagation of the real-time analysis results through the system. It is necessary to propagate the results of the real-time analysis of the different modules in an abstract way. The global analysis is built by connecting the local analyses of the single modules. Therefore it is essiential to have an expressive and efficient interface describing the influence in timing of one module to the next module. One aspect of this interface is the timing description of events which are produced by one module to trigger the next following module. Another aspect is the remaining computation capacity for the next module left over by the previous module.

Consider for example a network packet processor as shown in figure 1. The single packages are processed by chains of tasks τ which can be located on different processing elements P. The processing elements P can be processors, dedicated hardware or the communication network. The events Θ triggering the different tasks are equal to the packages flowing through the network. Each processing unit P uses a fixed-priority scheduling and the tasks on each unit are sorted by their priority level. Each task τ has, as available capacity, the capacity S' left over by the tasks τ with a higher priority located on the same processing unit.

The purpose of this paper is to provide an efficient and flexible approach for the real-time analysis of such modularized systems. Therefore a powerful and sufficient event model for describing the different time interfaces for the different aspects is necessary.

2. Related work

The most advanced approach for the real-time analysis of such a modulare network is the real-time calculus by Thiele et al. [**7**], [**16**]. It is based on the network calculus approach, especially on the concept of arrival and service curves defined by Cruz [**8**] and Parekh and Gallager [**12**].

The event pattern is modeled by an arrival curve $\alpha_f(\Delta I)$ which denotes the number of events arriving within a time interval of length ΔI, $\alpha_f^u(\Delta I)$ denoting the upper bound and $\alpha_f^l(\Delta I)$ the lower bound for this curve. These functions are sub-additive and deliver for

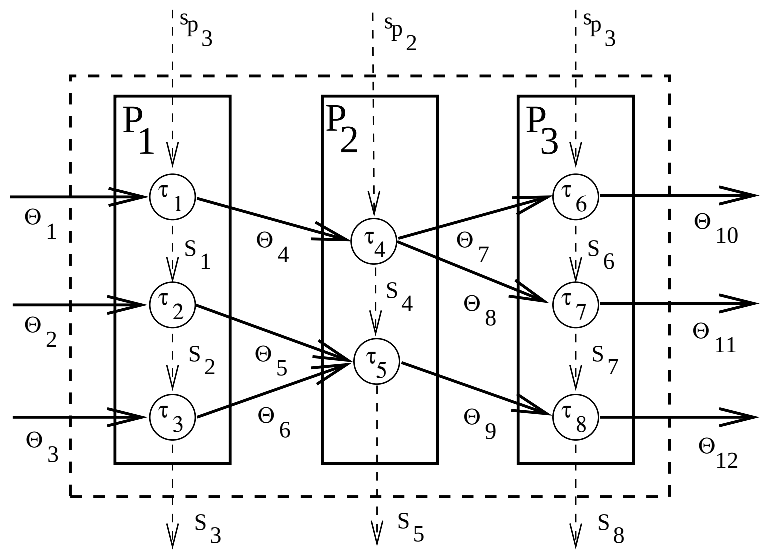

FIGURE 1. Network processor example

every t the maximum or the minimum number of events respectivly. The service curves $\beta_r^u(\Delta I)$ and $\beta_r^l(\Delta I)$ model the upper and lower bound of the computational requirements which can be handled by the ressource during a time interval of length ΔI. The real-time calculus provides equations to calculate the outgoing arrival and service curves out of the incoming curves of a task.

To make it possible to evaluate the modification equations independently from each other, a good finit description for the curves is needed. The complexity of the relationship equations depends directly on the complexity of this description. In [**11**] and [**7**] an approximation for the arrival and service curves was proposed in which each curve is described by three straight line segments. One segment describes the initial offset or arrival time, one segment the initial bursts and one segment the long time rate. As outlined in [**4**] this approach is much to simplified to be suitable for complex systems. It has only a fixed degree of exactness. No suitable description for the function is known so far.

In this paper we will propose a model for the curves having a selectable approximation error. It allows a trade-off between this degree of accuracy and the necessary effort for the analysis to become possible.

Sympta [**14**],[**15**] is another approach for the modularized real-time analysis. The idea was to provide a set of interfaces which can connect different event models. Therefore the different modules can use different event models for analysis. Unfortunatly, the event models for which interfaces are provided are quite simple. In [**14**] an event model covering all these models is described. The problem of these models is that multiple bursts or bursts with different minimum separation times cannot be handled qualitatively.

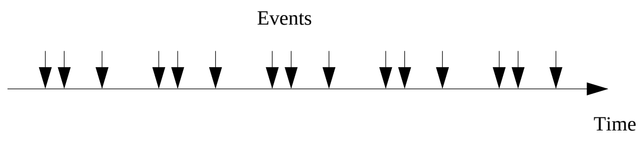

FIGURE 2. Example Event Stream

However in [**13**] a real-time analysis problem was formulated, which can't be solved by Sympta/S and the real-time calculus by each technique exclusivly. To solve it, it is necessary to integrate the models of both techniques into one powerful new model.

The event stream model proposed by Gresser [**10**] with its extension the hierachical event stream model proposed by Albers et al. [**1**] can model systems with all kinds of bursts efficiently. The problem is that it can only model discrete events and not the continious service function as needed for the real-time calculus.

2.1. Event stream model. For the event stream model a system is described by a set of communicating tasks τ. Each task is assigned to one resource ρ. The properties of each task $\tau = (\hat{\Theta}, c, d)$ are given by the worst-case execution time c_τ and the deadline d_τ of the task and an event pattern $\hat{\Theta}$ triggering the tasks activations.

The key question is to find a good model for the event pattern $\hat{\Theta}$. For real-time analysis this model has to describe the worst-case densities of all possible event patterns. They lead to the worst-case demand on computation time. Comparing these worst-case demands with the available computation time allows to predict the schedulability of a system. The event stream model gives an efficient general notation for the event bound function.

DEFINITION 2.1. ([**10, 3, 1**]) The event bound function $\Upsilon(\Delta I, \Theta)$ gives for every interval ΔI an upper bound on the number of events occuring from the event stream Θ in any interval of the lenght ΔI.

LEMMA 2.2. *([**10**]) The event bound function is a subadditive function. That means that for each interval ΔI and ΔJ the number of events generated within the interval $\Delta I + \Delta J$ is smaller or equal than the sum of the maximum number of events generated in ΔI and maximum number of events generated in ΔJ:*

$$\Upsilon(\Delta I + \Delta J, \Theta) \leq \Upsilon(\Delta I, \Theta) + \Upsilon(\Delta J, \Theta)$$

PROOF. $\Upsilon(\Delta I, \Theta)$, $\Upsilon(\Delta J, \Theta)$ return the maximum number of events possible within any ΔI or ΔJ. The events in $\Delta I + \Delta J$ have to occure either in ΔI or in ΔJ. Therefore the condition holds. □

DEFINITION 2.3. An event stream Θ is a set of event elements θ. Each event element is given by a period T and an offset a. $(\theta = (T, a))$

$\Theta_1 = \{(6,0), (6,1), (6,3)\}$ (figure 2) describes three events requiring at least an interval $\Delta I = 3$ to occure, two of them have a minimum distance of 1 time units. Θ_1 is repeated

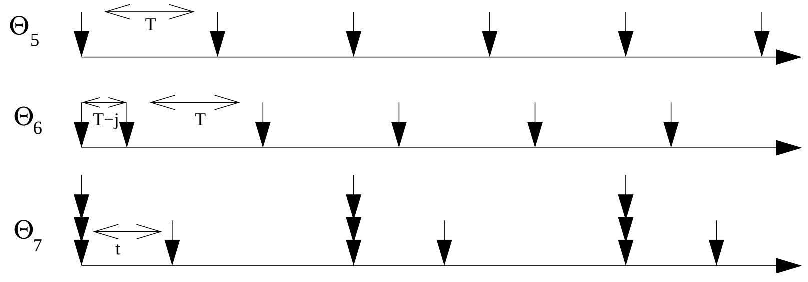

FIGURE 3. Example event streams ([**9**])

with a period of 6. In cases where the worst-case density of events is unknown for a concrete system, an upper bound can be used for the event stream. The model can describe any event sequence. Only those event sequences for which the condition of sub-additivity holds are valid event streams.

LEMMA 2.4. *([**10**]) The event bound function for an event stream* Θ *and an interval* I *is given by:*

$$\Upsilon(\Delta I,\Theta) = \sum_{\substack{\theta\in\Theta\\ \Delta I\geq a_\theta}} \left\lfloor \frac{\Delta I - a_\theta}{T_\theta} + 1 \right\rfloor$$

PROOF. See [**3**] □

It is a monotonic non-decreasing function. A larger interval-length cannot lead to a smaller number of events. An event stream is called homogenous if it contains only event elements sharing the same period or event elements having an infinit period.

In figure 3 some examples for event streams can be found. The first one $\Theta_5 = \{(T,0)\}$ has a strictly periodic stimulus with a period T. The second example $\Theta_6 = \{(\infty,0), (T,T-j)\}$ shows a periodic stimulus in which the single events can jitter within a jitter interval of size j. In the third example Θ_7 = $\{(T,0), (T,0)$, $(T,0), (T,t)$ $\}$ three events occur at the same time and the fourth occurs after a time t. This pattern is repeated with a period of T. Event streams can describe all these examples in an easy and intuitive way. The offset value of the first event element is always zero. The reason is that this value models the shortest interval in which one single event can occur.

For the real-time analysis for this model let us first repeate the demand bound function definition for the event streams:

$$\Psi(\Delta I,\Gamma) = \sum_{\forall\tau\in\Gamma} \Upsilon(\Delta I - d_\tau, \Theta_\tau) c_\tau = \sum_{\forall\tau\in\Gamma} \sum_{\substack{\forall\theta\in\Theta_\tau\\ \Delta I\geq a_\theta + d_\tau}} \left\lfloor \frac{\Delta I - a_\theta - d_\tau}{T_\theta} + 1 \right\rfloor c_\tau$$

Let θ be an event element belonging to the event stream Θ which belongs to the task τ.

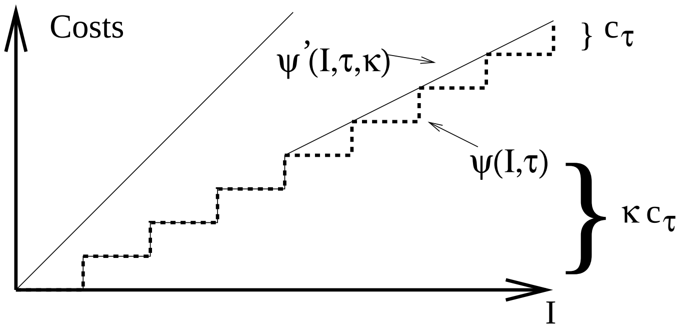

FIGURE 4. Approximated event stream element

The demand bound function allows a schedulability analysis for single processor systems by testing whether for every interval ΔI the demand is equal or smaller than the available capacity for this interval. Formally it is tested:

$$\forall \Delta I : \Psi(\Delta I, \Gamma) \leq \mathscr{C}(\Delta I)$$

Often an idealized capacity function $\mathscr{C}$ with $\mathscr{C}(\Delta I) = \Delta I$ is assumed. For an efficient analysis an approximation is necessary.

2.2. Approximation of event streams.

DEFINITION 2.5. *(*[**3**]*) The approximated event-bound-function*

Let τ be the task triggered by the event stream Θ having the event stream element θ and k be a chosen number of steps which should be considered for the task exactly. Let $\Delta I_{\theta,k} = d_\tau + a_\theta + kT$. We call $\Psi'(\Delta I, \theta, \tau, k)$ with

$$\Psi'(\Delta I, \theta, k) = \begin{cases} \Psi(\Delta I_{\theta,k}, \theta) + \frac{c_{\tau_\theta}}{T_\theta}(\Delta I - \Delta I_{\theta,k}) & \Delta I > \Delta I_{\theta,k} \\ \Psi(\Delta I, \theta) & \Delta I \leq \Delta I_{\theta,k} \end{cases}$$

the approximated event bound function for task τ.

The function is shown in figure 4. The first k events are evaluated exactly, the remaining events are approximated using the specific utilization $U_\theta = \frac{c_{\tau_\theta}}{T_\theta}$. The interesting point of this function is that the error can be bounded to $\varepsilon_{\theta,k} = \frac{1}{k}$ and therefore does depend on the selectable number of steps only, and is independent of the concrete values of the parameters of the tasks.

The complete approximated event-bound function for the event stream model is the sum of the approximated event-bound function for the single task:

$$\Psi'(\Delta I, \Gamma, k) = \sum_{\forall \tau \in \Gamma} \sum_{\forall \theta \in \Theta_\tau} \Psi'(\Delta I, \theta, k)$$

The hierachical event stream model [**1**] extends the event stream model and allows a more efficient description of bursts. In this model an event element describes the arrival not only for just one periodic event but also of a complete set of periodic events. This set of events can also be modeled by an event sequence having a limitation in the number of events generated by this event sequence. One limit of this model is that it can only describe discrete events. For the approximation it would be appropriate for the model to be capable also to describe the continuous part of the approximated event bound function.

3. Contribution

In this paper we will present an event model covering both, the discrete event model of Sympta/S and the continuous functions of the real-time calculus. It makes the elegant description of event bursts in a more tighter way than in the Sympta/S approach possible and allows a tighter modeling of the continuous function of the real-time calculus by integrating an approximation with a chooseable degree of exactness into the model. This does not only lead to more flexible and simpler analysis algorithms, but it also allows to propagate the approximation together with the event models through the distributed system leading to an efficient, flexible and powerful analysis methodology for the distributed real-time systems. The new model can, of course, also model the service functions of the real-time calculus in the same flexible way and allows therefore the integration of the discrete event model of Sympta/S with the continuous service functions.

We will also propose a simple but flexible and powerful approximative model for the explicit description of the curves of the real-time calculus. This model combines the description of arrival and service curves efficiently and allows to model them with a selectable degree of exactness. This approximation follows the same scheme than the existing approximation for event models as proposed in [**3**]. Therefore it is possible to transfeer previously existing event models, like the periodic or the sporadic task model, the event stream model, the sporadically task model, the model of Sympta/S or the hierarchical event stream model in this new model. This allows the integration of the approximative analysis for the event models and the real-time calculus to a new powerful overall analysis for the distributed systems.

We will outline this transfer methods for the various event models and the resulting real-time analysis for the new model for EDF and static priority scheduling. For the real-time calculus the new model provides a flexible and efficient approximative description of the curves. We will give the concrete algorithm for this model for all operationen necessary to implement the real-time calculus. This is the first concrete implementation of the real-time calculus which is not based on three line-segments for each curve only.

4. Model

We will define the hierachical event sequence first. The hierarchical event stream is only a specialised hierachical event sequence fulfilling the condition of sub-additivity and can therefore be described by the same model.

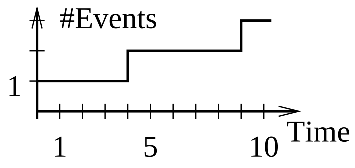

FIGURE 5. Example of a simple periodic event sequence

DEFINITION 4.1. *A hierachical event sequence $\hat{\Theta} = \{\hat{\theta}\}$ consists of a set of hierarchical event elements $\hat{\theta}$ each describing a pattern of events or of demand which is repeated periodically. The hierarchical event elements are described by a 5-tuples:*

$$\hat{\theta} = (T, a, l, G, \hat{\Theta})$$

where T_θ is the period, a_θ is the offset, l_θ is the limitation of the number of events or the amount of demand generated by this element during one period, $G_{\hat{\theta}}$ and $\hat{\Theta}_{\hat{\theta}}$ are the time pattern showing how the events respectively how the demand is generated.

The gradient $G_{\hat{\theta}}$ describing a constantly growing set of events, gives the number of events occurring within one time unit. A value $G_{\hat{\theta}} = 1$ means that after one time unit one event has occured, after two time units two events and so on. The gradient allows modeling approximated event streams as well as modeling the capacity of resources. Both cases can be described by a number of events which occurs respectively can be processed within one time unit. $\hat{\Theta}_{\hat{\theta}}$ is again a hierarchical event stream (child event stream) which is recursively embedded in $\hat{\theta}$.

CONDITION 4.2. *Either $\hat{\Theta}_{\hat{\theta}} = \emptyset$ or $G_{\hat{\theta}} = 0$.*

Due to this condition it is not necessary to distribute the limitation between the gradient and the sub-element. This simplifies the analysis without restricting the modelling capabilities.

The arrival of the first event occurs after a time units and at $a+T$, $a+2T$, $a+3T$, ..., $a+iT$ $(i \in N)$ the other events occurs.

DEFINITION 4.3. *A hierarchical event stream fulfills for every $\Delta I, \Delta J$ the condition $\Upsilon(\Delta I + \Delta J, \hat{\Theta}) \leq \Upsilon(\Delta I, \hat{\Theta}) + \Upsilon(\Delta J, \hat{\Theta})$*

In the following we will give a few examples to show the usage and the possibilities of the new model. A simple periodic event stream as outlined in figure 5 with period 5 can be modeled by :

$$\hat{\Theta}_1 = \{(5, 0, 1, 0, e)\}$$

LEMMA 4.4. *Let Θ be an event stream with $\Theta = \{\theta_1, ..., \theta_n\}$. Θ can be modeled by $\hat{\Theta} = \{\hat{\theta}_1, ..., \hat{\theta}_n\}$ with $\hat{\theta}_i = (T_{\theta_i}, a_{\theta_i}, 1, \infty, \emptyset)$*

PROOF. Each of the hierarchical event elements generates exactly one event at each of its periods following the pattern of the corresponding event element. □

$\hat{\Theta}_1$ approximated after 10 events would be modeled by:

$$\hat{\Theta}_1^{10} = \{(\infty, 0, 10, 0, \{(5, 2, 1, 0, e)\}), (\infty, 47, \infty, \frac{1}{5}, \emptyset)\}$$

Note that $47t.u. = 2t.u. + (10-1) \cdot 5t.u.$ ($t.u. = time unit$) is the point in time in which the last regular event occurs and therefore the start of the approximation.

One single event is modeled by:

$$\hat{\Theta}_2 = \{(\infty, 0, 1, \infty, \emptyset)\}$$

A gradient of ∞ would lead to an infinite number of events but due to the limitation only one event is generated. An event bound function requiring constantly 0.75 time units processor time within each time unit can be described by:

$$\hat{\theta}_2 = (\infty, 0, \infty, 0.75, \emptyset)$$

With the recursively embedded event sequence any possible pattern of events within a burst can be modeled. The pattern consists of a limited set of events repeated by the period of the parent hierarchical event element. For example a burst of five events in which the events have an intra-arrival rate of 2 time units which is repeated after 50 time units can be modeled by:

$$\hat{\Theta}_3 = \{(50, 0, 5, 0, \{(2, 0, 1, \infty, \emptyset)\})\}$$

The child event stream can contain grand-child event streams. For example if $\hat{\Theta}_3$ is used only for 1000 time units and than a break of 1000 time units is required would be modeled by

$$\hat{\Theta}_4 = \{(2000, 0, 100, 0, \hat{\Theta}_3)\}$$

The length $\Delta I_{\hat{\theta}}$ of the interval for which the limitation of $\hat{\theta}$ is reached can be calculated using a interval bound function $\mathscr{I}(x, \hat{\Theta}) = min(\Delta I | x = \Upsilon(\Delta I, \hat{\Theta}))$ which is the inverse function to the event bound function ($\mathscr{I}(l, \emptyset) = 0$):

$$\Delta I_{\hat{\theta}} = \mathscr{I}(l, \hat{\Theta}_{\hat{\theta}}) + \frac{l_{\hat{\theta}}}{G_{\hat{\theta}}}$$

Note that this calculation requires the condition of the model that either $G_{\hat{\theta}} = 0$ or $\hat{\Theta}_{\hat{\theta}} = \emptyset$ and that the calculation of the interval bound function requires the distribution of $l_{\hat{\theta}}$ on the elements of $\hat{\Theta}_{\hat{\theta}}$.

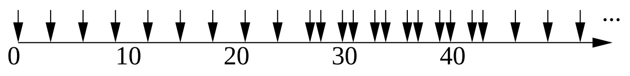

FIGURE 6. Example for overlapping events of different periods

4.1. Assumptions and Conditions. For the analysis it is useful to restrict the model to event sequences having no overlapping periods. For example (figure 6):

$$\hat{\theta}_5 = \{(28, 0, 15, 0, \{(3, 0, 1, \infty, \emptyset)\})\}$$

The limitation interval $\Delta I_{\hat{\theta}_6}$ has the length:

$$\Delta I_{\hat{\theta}_6} = (15 - 1) \cdot 3 = 42$$

The first period $[0, 42]$ and the second period $[28, 70]$ of the event sequence element overlap.

CONDITION 4.5. *(Separation Condition) $\hat{\theta}$ fulfills the separation condition if the interval in which events are generated by $G_{\hat{\theta}}$ or $\hat{\Theta}_{\hat{\theta}}$ is equal or smaller than its period $T_{\hat{\theta}}$:*

$$\mathscr{I}(l_{\hat{\theta}}, \hat{\Theta}_{\hat{\theta}}) + \frac{l_{\hat{\theta}}}{G_{\hat{\theta}}} \leq T_{\hat{\theta}}$$

or

$$T_{\hat{\theta}} \leq \Upsilon(T_{\hat{\theta}}, \hat{\Theta}_{\hat{\theta}}) + \frac{T_{\hat{\theta}}}{G_{\hat{\theta}}}$$

The condition 4.5 does not reduce the space of event patterns that can be modeled by a hierarchical event sequence.

LEMMA 4.6. A hierarchical event sequence element $\hat{\theta}$ that does not meet the separation condition can be exchanged with a set of event sequence elements $\hat{\theta}_1, ..., \hat{\theta}_k$ with $k = \left\lceil \frac{\mathscr{I}(l_{\hat{\theta}}, \hat{\theta})}{T_{\hat{\theta}}} \right\rceil$ and $\hat{\theta}_i = (kT_{\hat{\theta}}, (i-1)T_{\hat{\theta}} + a_{\hat{\theta}}, l_{\hat{\theta}}, G_{\hat{\theta}}, \hat{\Theta}_{\hat{\theta}})$.

PROOF. The proof is obvious and therefore skipped. □

$\hat{\Theta}_5$ can be transferred into $\hat{\Theta}'_5$ meeting the separation condition:

$$\hat{\Theta}'_5 = \{(56, 0, 15, 0, \{(3, 0, 1, \infty, \emptyset)\}), (56, 28, 15, 0, \{(3, 0, 1, \infty, \emptyset)\})\}$$

The separation condition prohibits events of different event sequence elements to overlap. We also do not allow recursion, so no event element can be the child of itself (or a subsequent child element).

4.2. Hierarchical Event Bound Function. The event bound function calculates the maximum number of events generated by $\hat{\Theta}$ within ΔI.

LEMMA 4.7. Hierarchical Event Bound Function $\Upsilon(\Delta I, \Theta)$:
Let for any $\Delta I, T$ define $mod(\Delta I, T) = \Delta I - \left\lfloor \frac{\Delta I}{T} \right\rfloor T$ and $\Upsilon(\Delta I, \emptyset) = 0$.

$$\Upsilon(\Delta I, \hat{\Theta}) = \sum_{\substack{\hat{\theta} \in \hat{\Theta} \\ \Delta I \geq a_{\hat{\theta}}}} \Upsilon(\Delta I, \hat{\theta})$$

$$\Upsilon(\Delta I, \hat{\theta}) = \begin{cases} l_{\hat{\theta}} & T_{\hat{\theta}} = \infty, G_{\hat{\theta}} = \infty \\ \left\lfloor \frac{\Delta I - a_{\hat{\theta}}}{T_{\hat{\theta}}} + 1 \right\rfloor l_{\hat{\theta}} & T_{\hat{\theta}} \neq \infty, G_{\hat{\theta}} = \infty \\ min(l_{\hat{\theta}}, (\Delta I - a_{\hat{\theta}}) G_{\hat{\theta}} & \\ \quad + \Upsilon(\Delta I - a_{\hat{\theta}}, \hat{\Theta}_{\hat{\theta}})) & T_{\hat{\theta}} = \infty, G_{\hat{\theta}} \neq \infty \\ \left\lfloor \frac{\Delta I - a_{\hat{\theta}}}{T_{\hat{\theta}}} \right\rfloor l_{\hat{\theta}} + min(l_{\hat{\theta}}, & \\ \quad mod(\Delta I - a_{\hat{\theta}}, T_{\hat{\theta}}) G_{\hat{\theta}} & \\ \quad + \Upsilon(mod(\Delta I - a_{\hat{\theta}}, T_{\hat{\theta}}), \hat{\Theta}_{\hat{\theta}})) & T_{\hat{\theta}} \neq \infty, G_{\hat{\theta}} \neq \infty \end{cases}$$

PROOF. Due to the separation condition it is always possible to include the maximum allowed number of events for completed periods $\left(\left\lfloor \frac{\Delta I - a_{\hat{\theta}}}{T_{\hat{\theta}}} \right\rfloor l_{\hat{\theta}} \right)$. Only the last incomplete fraction of a period has to be considered separately $(\min(\ldots))$. This remaining interval is given by subtracting all complete periods, and the offset a from the interval ΔI $\left(mod(\Delta I - a_{\hat{\theta}}, T_{\hat{\theta}} \right)$. It has to be distinguished whether the gradient or the child event stream generates the events. In case of the child event stream, the number of events is calculated by using the same function with the remaining interval and the new embedded event sequence. In case of the gradient the number of events is simply the product of the gradient and the interval length. The limitation bounds both values. □

Independently of the hierarchical level on which an event sequence element is located it is considered only once during the calculation for one interval. This allows bounding the complexity of the calculation. It is not necessary for the sequences to be homogeneous.

EXAMPLE 4.8. $\hat{\Theta}_6 = \{(20, 6, 10, 0, \{(3, 0, 2, 1, \emptyset)\}$. $\Upsilon(\Delta I, \hat{\Theta}_7)$ is shown in figure 7. $\Upsilon(33, \hat{\Theta}_6)$ is given by

$$\begin{aligned} \Upsilon(33, \hat{\Theta}_6) &= \left\lfloor \frac{27}{T_{\hat{\theta}}} \right\rfloor l_{\hat{\theta}} + \min(l_{\hat{\theta}}, mod(27, T_{\hat{\theta}}) G_{\hat{\theta}} + \Upsilon(mod(27, T_{\hat{\theta}}), \hat{\Theta}_{\hat{\theta}})) \\ &= \left\lfloor \frac{27}{20} \right\rfloor \cdot 10 + \min(10, 0 + \Upsilon(7, \hat{\Theta}_{\hat{\theta}})) = 10 + \min(10, \Upsilon(7, \hat{\Theta}_{\hat{\theta}})) \\ \Upsilon(7, \hat{\Theta}_{\hat{\theta}}) &= \Upsilon(7, \hat{\theta}') = \left\lfloor \frac{7}{3} \right\rfloor \cdot 2 + min(2, mod(7, 3) \cdot 1 + 0) = 4 + 1 = 5 \\ \Upsilon(33, \hat{\Theta}_6) &= 10 + \min(10, 5) = 15 \end{aligned}$$

4.3. Reduction and Normalization. In the following we will reduce event streams to a normal form. The hierarchical event stream model allows several different description

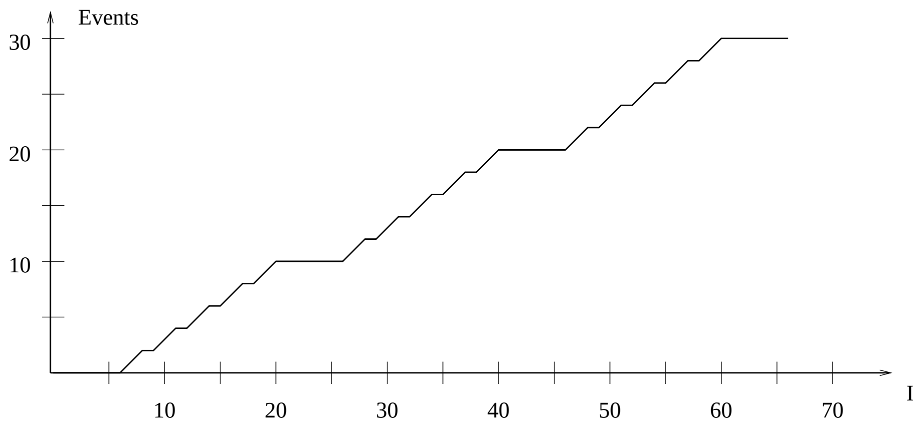

FIGURE 7. Hierarchical event sequence $\hat{\Theta}_6$

for the same event pattern. For example an event stream

$$\hat{\Theta} = \{(100, 0, 20, 0, \hat{\Theta}_a)\}$$

with

$$\hat{\Theta}_a = \{(5, 0, 2, \infty, \emptyset), (7, 2, 3, 1, \emptyset)\}$$

can be rewritten as

$$\hat{\Theta} = \{(100, 0, 10, 0, \hat{\theta}_{a,1}), (100, 0, 10, \hat{\theta}_{a,2})\}$$

with

$$\hat{\theta}_{a,1} = (5, 0, 2, \infty, \emptyset)$$

and

$$\hat{\theta}_{a,2} = (7, 2, 3, 1, \emptyset)$$

Event streams having child event streams with several event elements can be transformed into streams having only child streams with one element. This allows a better comparison between different hierarchical event streams:

LEMMA 4.9. *An event stream $\hat{\Theta}_a = \{(T_a, a_a, l_a, 0, \hat{\Theta}'_a)\}$ with a child element $\hat{\Theta}'_a = \{(T'_1, a'_1, l'_1, G'_1, \hat{\Theta}_1), ..., (T'_k, a'_k, l'_k, G'_k, \hat{\Theta}_k)\}$ can be transferred into an equivalent event stream $\hat{\Theta}_b$ with $\hat{\Theta}_b = \{\hat{\theta}_{a,1}, \hat{\theta}_{a,2}, ..., \hat{\theta}_{a,n}, \hat{\theta}_{a,x}\}$ having only child event sequences with one element where*

$$\begin{aligned}
\hat{\theta}_{b,i} &= (T, a, \Upsilon(\Delta I_a, \hat{\theta}'_{a,i}), 0, \hat{\theta}'_{a,i}) \\
\Delta I_a &= \lim_{\substack{\varepsilon \to 0 \\ \varepsilon > 0}} (\mathscr{I}(l_a, \hat{\Theta}'_a) - \varepsilon) \\
\hat{\theta}_{a,x} &= (\infty, \mathscr{I}(l_a, \hat{\Theta}'_a), l_a - \sum_{\forall \hat{\theta} \in \hat{\Theta}'_a} \Upsilon(\Delta I_a, \hat{\theta}'_{a,i}), \infty, \emptyset)
\end{aligned}$$

PROOF. We have to distribute the limitation l_a on the elements of the child event sequence. First we have to find the interval $\Delta I'$ for which the limitation of the parent element l_a is reached by the child event sequence $\hat{\Theta}'_a$. $\Delta I'$ is given by $\mathscr{I}(l_a, \hat{\Theta}'_a)$. We have to calculate the costs required for each of the child event sequence elements for $\Delta I'$. It is given by $\Upsilon(\Delta I', \hat{\theta}_i)$. The problem is that several elements can have a gradient of ∞ exactly at the end of $\Delta I'$. In this situation the sum of $\Upsilon(\Delta I', \hat{\theta})$ may exceed the allowed limitation l_a of the parent element. The total costs is bounded by the global limitation l_a rather than the limitations l'_i. To take this effect into account we exclude the costs occurring exactly at the end of $\Delta I'$ for each hierarchical event element and we handle these costs seperately modeling them with the hierarchical event element $\hat{\theta}_{a,x}$. To do so we calculate the limitation not by $\Upsilon(\Delta I', \hat{\theta}'_i)$ but by $\Upsilon(\Delta I' - \varepsilon, \hat{\theta}'_i)$ where ε is an infinitly small value excluding only the costs occurring at the end of $\Delta I'$ exactly. □

4.4. Capacity Function. The proposed hierarchical event stream model can also model the capacity of processing elements and allows to describe systems with fluctuating capacity over the time. In the standard case a processor can handle one time unit execution time during one time unit real time. For many resources the capacity is not constant. The reasons for a fluctuating capacity can be such as operation-system tasks or variable processor speeds due to energy constraints.

Assuming the capacity as constant also does not support a modularization of the analysis. This is especially needed for hierarchical scheduling approaches. Consider for example a fixed priority scheduling. In a modular approach each priority level gets the capacity left over by the previous priority level as available capacity. The remaining capacity can be calculated step-wise for each priority level taking only the remaining capacities of the next higher priority level into account. Such an approach is only possible with a model that can describe the left-over capacities exactly.

DEFINITION 4.10. The service function $\beta(\Delta I, \rho)$ gives the minimum amount of processing time that is available for processing tasks in any interval of size ΔI for a specific resource ρ for each interval ΔI . It can also be modeled with the hierarchical event sequence model.

The service function is superadditiv and fulfills the inequation $\beta(\Delta I + \Delta J) \geq \beta(\Delta I) + \beta(\Delta J)$ for all $\Delta I, \Delta J$. The definition matches the service curves of the real-time calculus. We propose to use the hierarchical event stream model as an explicit description for service curves.

In the following we will show, with a few examples, how to model fluctuating service functions with the hierarchical event streams. The constant capacity, as shown in 8 a) can be modeled by: $\beta_{basic} = \{(\infty, 0, \infty, 1, \emptyset)\}$

Blocking the service for a certain time t (figure 8 b) is done by: $\beta_{block} = \{(\infty, t, \infty, 1, \emptyset)\}$

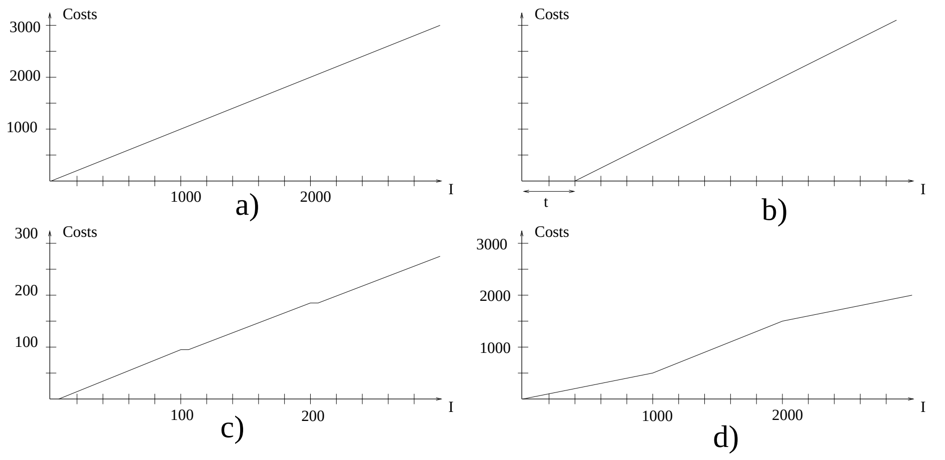

FIGURE 8. Example service bound functions

A constantly growing service curve in which the service is blocked periodically every 100 time units for 5 time units (for example by a task of the operating system): $\beta_{Tblock} = \{(100, 5, 95, 1, \emptyset)\}$ (figure 8 c))

The service for a processor that can handle only 1000 time units with full speed and than 1000 time units with half speed (figure 8 d)):

$$\beta_{vary} = \{(2000, 1000, 500, \frac{1}{2}, \emptyset), (2000, 0, 1000, 1, \emptyset)\}$$

These are only a few examples for the possibilities of this new model.

4.5. Operations. In the following we will introduce some operations on hierarchical event sequences and streams. These are operations to add sequences, to shift them by certain time values, for example the deadlines, and to scale them with for example costs values. The operations are necessary for the schedulability tests.

4.5.1. *Adding* $(+)$. The add operation for two event streams can be simply realized by a union of the sets of event elements of the two event streams:

DEFINITION 4.11. ($+$ operation) Let $\hat{\Theta}_A, \hat{\Theta}_B, \hat{\Theta}_C$ be hierarchical event streams. If $\hat{\Theta}_C$ is the sum of $\hat{\Theta}_A$ and $\hat{\Theta}_B$ ($\hat{\Theta}_C = \hat{\Theta}_A + \hat{\Theta}_B$) than for each interval ΔI the equation $\Upsilon(\Delta I, \hat{\Theta}_C) = \Upsilon(\Delta I, \hat{\Theta}_A) + \Upsilon(\Delta I, \hat{\Theta}_B)$ is true.

LEMMA 4.12. *($+$ operation) The sum $\hat{\Theta}_C = \hat{\Theta}_A + \hat{\Theta}_B$ can be calculated by the union of the event stream elements of $\hat{\Theta}_A, \hat{\Theta}_B$:*

$$\hat{\Theta}_C = \hat{\Theta}_A \cup \hat{\Theta}_B$$

PROOF.

$$\Upsilon(\Delta I, \hat{\Theta}_C) \quad = \quad \Upsilon(\Delta I, \hat{\Theta}_A) + \Upsilon(\Delta I, \hat{\Theta}_B)$$

$$
\begin{aligned}
&= \sum_{\hat\theta \in \hat\Theta_A} \Upsilon(\Delta I, \hat\theta) + \sum_{\forall \hat\theta \in \hat\Theta_B} \Upsilon(\Delta I, \hat\theta) \\
&= \sum_{\forall \hat\theta \in \hat\Theta_A \cup \hat\Theta_B} \Upsilon(\Delta I, \hat\theta) \\
&= \Upsilon(\Delta I, \hat\Theta_A \cup \hat\Theta_B)
\end{aligned}
$$

□

The operation works with hierarchical event sequences as well as with hierarchical event streams. It inherits the properties of the $+$ operation, so the function is commutative as well as associative.

4.5.2. *Shift Operation* $(\leftarrow, \rightarrow)$. The shift operation can be realized by adding or subtracting the shift-value from each offset of all top-level elements of the event stream. When subtracting, the shift value has not necessarily to be equal or smaller than the smallest offset. The event bound function $\Upsilon(\Delta I, \hat\Theta)$ with $\Delta I \geq 0$ can handle negative offsets even though that negative intervals are not defined.

DEFINITION 4.13. ($\rightarrow$ shift-operation) Let $\hat\Theta$ be an event sequence that is shifted right by the value t resulting in the event sequence $\hat\Theta' = \hat\Theta \rightarrow t$. Thus the event bound functions have the following relationship:

$$
\Upsilon(\Delta I, \hat\Theta') = \begin{cases} \Upsilon(\Delta I - t, \hat\Theta) & \Delta I \geq t \\ 0 & else \end{cases}
$$

LEMMA 4.14. $\Upsilon(\Delta I, \hat\Theta) \rightarrow t = \Upsilon(\Delta I, \hat\Theta')$ *if* $\hat\Theta'$ *contains and only contains for each element* $\hat\theta$ *of* $\hat\Theta$ *an element* $\hat\theta' \in \hat\Theta'$ *having the following relations to* $\hat\theta$*:* $T_{\hat\theta'} = T_{\hat\theta}$, $a_{\hat\theta'} = a_{\hat\theta} + t$, $n_{\hat\theta'} = n_{\hat\theta}$, $\hat\Theta_{\hat\theta'} = \hat\Theta_{\hat\theta}$, $G_{\hat\theta'} = G_{\hat\theta}$

The operation $\hat\Theta' = \hat\Theta \rightarrow t$ can be performed by adding the value t to the offset $a_{\hat\theta}$ for each event element $\hat\theta \in \hat\Theta$ for its corresponding counter-element $\hat\theta' \in \hat\Theta'$.

PROOF.

$$
\begin{aligned}
\Upsilon(\Delta I - t, \hat\Theta) &= \sum_{\substack{\hat\theta \in \hat\Theta \\ \Delta I \geq t}} \Upsilon(\Delta I - t, \hat\theta) \\
&= \sum_{\substack{\hat\theta \in \hat\Theta \\ \Delta I \geq a_{\hat\theta} + t}} \Upsilon'(\Delta I - t - a_{\hat\theta}, \hat\theta) \\
&= \sum_{\substack{\hat\theta \in \hat\Theta \\ \Delta I \geq a_{\hat\theta} + t}} \Upsilon'(\Delta I - (a_{\hat\theta} + t), \hat\theta) \\
&= \sum_{\substack{\hat\theta \in \hat\Theta \\ \Delta I \geq a_{\hat\theta} + t}} \Upsilon'(\Delta I - a_{\hat\theta'}, \hat\theta) \\
&= \Upsilon(\Delta I, \hat\Theta')
\end{aligned}
$$

□

The operation to shift a value left by the value t ($\hat{\Theta} \leftarrow t$) can be defined in a similar way.

DEFINITION 4.15. ($\leftarrow$ shift-operation) Let $\hat{\Theta}$ be an event sequence that is shifted left by the value t resulting in the event sequence $\hat{\Theta}' = \hat{\Theta} \leftarrow t$. The event bound functions have the following relationship:

$$\Upsilon(\Delta I, \hat{\Theta}') = \Upsilon(\Delta I + t, \hat{\Theta})$$

LEMMA 4.16. $\Upsilon(\Delta I, \hat{\Theta}) \leftarrow t = \Upsilon(\Delta I, \hat{\Theta}')$ *if* $\hat{\Theta}'$ *contains and only contains for each element* $\hat{\theta}$ *of* $\hat{\Theta}$ *an element* $\hat{\theta}' \in \hat{\Theta}'$ *having the following relations to* $\hat{\theta}$*:* $T_{\hat{\theta}'} = T_{\hat{\theta}}$, $a_{\hat{\theta}'} = a_{\hat{\theta}} - t$, $n_{\theta'} = n_{\theta}$, $\hat{\Theta}_{\hat{\theta}'} = \hat{\Theta}_{\hat{\theta}}$, $G_{\hat{\theta}'} = G_{\hat{\theta}}$

PROOF.

$$\begin{aligned}
\Upsilon(\Delta I + t, \hat{\Theta}) &= \sum_{\hat{\theta} \in \hat{\Theta}} \Upsilon(\Delta I + t, \hat{\theta}) \\
&= \sum_{\substack{\hat{\theta} \in \hat{\Theta} \\ \Delta I \geq a_{\hat{\theta}} - t}} \Upsilon'(\Delta I + t - a_{\hat{\theta}}, \hat{\theta}) \\
&= \sum_{\substack{\hat{\theta} \in \hat{\Theta} \\ \Delta I \geq a_{\hat{\theta}} - t}} \Upsilon'(\Delta I - (a_{\hat{\theta}} - t), \hat{\theta}) \\
&= \sum_{\substack{\hat{\theta} \in \hat{\Theta} \\ \Delta I \geq a_{\hat{\theta}} - t}} \Upsilon'(\Delta I - a_{\hat{\theta}'}, \hat{\theta}) = \Upsilon(\Delta I, \hat{\Theta}')
\end{aligned}$$

□

This operation also works on both sequences and streams. It is associative with the $(+)$ operation so we have $(\hat{\Theta}_A + \hat{\Theta}_B) \rightarrow t = (\hat{\Theta}_A \rightarrow t) + (\hat{\Theta}_B \rightarrow t)$ and $(\hat{\Theta}_A + \hat{\Theta}_B) \leftarrow t = (\hat{\Theta}_A \leftarrow t) + (\hat{\Theta}_B \leftarrow t)$ as well as $(\hat{\Theta}_A - \hat{\Theta}_B) \rightarrow t = (\hat{\Theta}_A \rightarrow t) - (\hat{\Theta}_B \rightarrow t)$ and $(\hat{\Theta}_A - \hat{\Theta}_B) \leftarrow t = (\hat{\Theta}_A \leftarrow t) - (\hat{\Theta}_B \leftarrow t)$. Having $(\hat{\Theta} \rightarrow t) \rightarrow v$ we can rewrite it as $\hat{\Theta} \rightarrow (t + v)$, having $(\hat{\Theta} \rightarrow t) \leftarrow v$ we can rewrite it as $\hat{\Theta} \rightarrow (t - v)$.

4.5.3. *Scaling with a cost value* $(\cdot)$. Another operation on event streams is to scale the total stream by a cost value. This is, for example, necessary for the integration of the worst-case execution times into the analysis. If the event stream uses the number of events as unit, it is necessary to scale it for analysis with the worst-case execution time.

DEFINITION 4.17. Let $\hat{\Theta}'$ be the hierarchical event stream $\hat{\Theta}$ scaled by the cost value c ($\hat{\Theta}' = c\hat{\Theta}$). Than for each interval ΔI the corresponding event bound functions have the relationship

$$\Upsilon(\Delta I, \hat{\Theta}') = c\Upsilon(\Delta I, \hat{\Theta})$$

LEMMA 4.18. $\Upsilon(\Delta I,\hat{\Theta}') = c\Upsilon(\Delta I,\hat{\Theta})$ *if the child set of* $\hat{\Theta}'$ *contains and only contains for each element* $\hat{\theta}$ *of the child set of* $\hat{\Theta}$ *an element* $\hat{\theta}' \in \hat{\Theta}'$ *having the following relations to* $\hat{\theta}$*:* $T_{\hat{\theta}'} = T_{\hat{\theta}}$, $a_{\hat{\theta}'} = a_{\hat{\theta}}$, $n_{\hat{\theta}'} = cn_{\hat{\theta}}$, $\hat{\Theta}_{\hat{\theta}'} = c\hat{\Theta}_{\hat{\theta}}$, $G_{\hat{\theta}} = cG_{\hat{\theta}}$

All parts of the hierarchical event sequence elements related to the amount of events are scaled by the variable c.

PROOF.

$$c\Upsilon(\Delta I,\hat{\Theta}) = \sum_{\substack{\hat{\theta}\in\hat{\Theta}\\ \Delta I\geq a_{\hat{\theta}}}} c\Upsilon(\Delta I,\hat{\theta})$$

$$\begin{aligned}
c\Upsilon(\Delta I,\hat{\theta}) &= \begin{cases} min(cl_{\hat{\theta}}, cG_{\hat{\theta}}(\Delta I - a_{\hat{\theta}}) + c\Upsilon(\Delta I,\hat{\Theta}_{\hat{\theta}}) & T_{\hat{\theta}} = \infty \\ cl_{\hat{\theta}} & |G_{\hat{\theta}}| = \infty \\ \left\lfloor \frac{\Delta I - a_{\hat{\theta}}}{T_{\hat{\theta}}} \right\rfloor cl_{\hat{\theta}} + min(cl_{\hat{\theta}}, cG_{\hat{\theta}} mod(\Delta I - a_{\hat{\theta}}, T_{\hat{\theta}}) + & \\ \qquad\qquad + c\Upsilon(mod(\Delta I - a_{\hat{\theta}}, T_{\hat{\theta}}), \hat{\Theta}_{\hat{\theta}})) & T_{\hat{\theta}} \neq \infty \end{cases} \\
&= \begin{cases} min(l_{\hat{\theta}'}, G_{\hat{\theta}'}(\Delta I - a_{\hat{\theta}}) + \Upsilon(\Delta I,\hat{\Theta}_{\hat{\theta}'}) & T_{\hat{\theta}} = \infty \\ l_{\hat{\theta}'} & |G_{\hat{\theta}}| = \infty \\ \left\lfloor \frac{\Delta I - a_{\hat{\theta}}}{T_{\hat{\theta}}} \right\rfloor l_{\hat{\theta}'} + min(l_{\hat{\theta}}, G_{\hat{\theta}'} mod(\Delta I - a_{\hat{\theta}}, T_{\hat{\theta}}) + & \\ \qquad\qquad + \Upsilon(mod(\Delta I - a_{\hat{\theta}}, T_{\hat{\theta}}), \hat{\Theta}_{\hat{\theta}'})) & T_{\hat{\theta}} \neq \infty \end{cases} \\
&= \Upsilon(\Delta I,\hat{\theta}')
\end{aligned}$$

□

The operation works with both, streams and sequences.

4.6. Utilization.

LEMMA 4.19. *The utilization* U_Γ *of a task set in which the event generation patterns are described by hierarchical event streams is given by* $((\forall\tau\in\Gamma)\Lambda\,(\forall\hat{\theta}\in\hat{\Theta}_\tau)|(l_{\hat{\theta}} \neq \infty \vee T_{\hat{\theta}} = \infty))$*:*

$$U_\Gamma = \sum_{\forall\tau\in\Gamma}\sum_{\substack{\forall\hat{\theta}\in\hat{\Theta}_\tau\\ T_\tau\neq\infty}} \frac{n_{\hat{\theta}}}{T_{\hat{\theta}}} + \sum_{\forall\tau\in\Gamma}\sum_{\substack{\forall\hat{\theta}\in\hat{\Theta}_\tau\\ l_{\hat{\theta}}=\infty\\ T_{\hat{\theta}}=\infty}} \left(U_{\hat{\Theta}_{\hat{\theta}}} + G_{\hat{\theta}}\right)$$

Note that event-elements with an infinite period and a finite limitation do not contribute to the utilization.

5. Schedulability tests

For the schedulability tests of uni-processor system using the hierarchical event stream model analysis, we can integrate the approximation and the available capacity into the analysis.

In the following we will show how an efficient schedulability test can be realized with the introduced model and operations. We will first discuss the schedulability test for a uniprocessor system using EDF (Earliest Deadline First) scheduling. Later we will extend the result to fixed priority scheduled systems.

5.1. Schedulability tests for dynamic priority systems. The general schedulability analysis for EDF is the processor demand criterion but using the demand bound function for the hierarchical event streams. A system scheduled with EDF is feasible if for all intervals ΔI the demand bound function does not exceed the service function $\Psi(\Delta I) \leq \mathscr{C}(\Delta I, \rho)$. Both, the demand bound and the service function can be described and calculated out of hierarchical event streams. This leads to the test:

$$\sum_{\forall\tau\in\Gamma}\sum_{\forall\hat{\theta}\in\hat{\Theta}_\tau} \Upsilon(\Delta I - d_\tau, \hat{\theta})c_\tau \leq \mathscr{C}(\Delta I, \rho)$$

The analysis can be done using the approximation as proposed in [**3**]. For the exact analysis an upper bound for ΔI, a maximum test interval is required to limit the run-time of the test. For the hierarchical event stream model one maximum test interval available is the busy period. An upper bound for it is given by:

$$\mathscr{B}(\Gamma) = \min(\Delta I | \mathscr{C}(\Delta I) \geq \sum_{\forall\tau\in\Gamma} \Upsilon(\Delta I, \hat{\Theta}_\tau)c_\tau)$$

5.2. Response-time calculation for static priority scheduling. In the following we will show how a worst-case response time analysis for scheduling with static priorities can be performed with the new model. The request bound function calculates the amount of computation time of a higher priority task that can interfere and therefore delays a lower-priority task within an interval ΔI. In contrast to the event bound function the request bound function does only contain the events of the start, not the events of the end point of the interval. The request bound function can be calculated using the event bound function in the following way:

$$\Phi(\Delta I, \tau) = \lim_{\substack{\Delta \to \Delta I \\ 0 \leq \Delta < \Delta I}} (\Upsilon(\Delta, \Theta_\tau)c_\tau)$$

For the hierarchical model it is only necessary to handle the cases $\Delta I = 0$ differently than in the calculation of the event bound function: $\Phi(\Delta I, \Gamma) = \sum_{\forall\tau\in\Gamma} c_\tau \sum_{\forall\hat{\theta}\in\hat{\Theta}_\tau} \Phi(\Delta I, \hat{\theta}, \tau)$ with

$$\Phi(\Delta I, \hat{\theta}, \tau) = \begin{cases} \left\lceil \frac{\Delta I - a_{\hat{\theta}}}{T_{\hat{\theta}}} \right\rceil l_{\hat{\theta}} & T_{\hat{\theta}} = \infty \\ 0 & \Delta I - a_{\hat{\theta}} \leq 0 \\ \left\lfloor \frac{\Delta I - a_{\hat{\theta}}}{T_{\hat{\theta}}} \right\rfloor l_{\hat{\theta}} + \min(l_{\hat{\theta}}, G_{\hat{\theta}}(\Delta I - a_{\hat{\theta}} + & \\ \quad \Phi(mod(\Delta I - a_{\hat{\theta}}, T_{\hat{\theta}}), \hat{\Theta}_{\hat{\theta}})) & else \end{cases}$$

With this function it is possible to calculate the worst-case response times for the tasks:

LEMMA 5.1. *Let τ be scheduled with fixed priorities and $\Gamma_{hp(\tau)}$ containing all task with a higher priority than τ. The response time $r(\tau_{i,1})$ for the first event of τ_i is given by:*

$$r(\tau_{i,1}) = \min(\Delta I | \mathscr{C}(\Delta I) \geq c_\tau + \Phi(\Delta I, \Gamma_{hp(\tau)}))$$

The value for ΔI can be calculated by a fix-point iteration starting with $\Delta I = c_\tau$. To calculate the maximum response time it is necessary to do the calculation for all events within the busy period.

The busy period of a task set is the maximum interval in which the resource is completely busy, so in which does not exists idle time for the resource:

$$\mathscr{B}(\Gamma) = \min(\Delta I | \mathscr{C}(\Delta I) \geq \Phi(\Delta I, \Gamma))$$

LEMMA 5.2. *The worst-case response time of τ can be found in the busy period of any task set containing τ and $\Gamma_{hp(\tau)}$. It is the maximum response time of all $r(J, \tau)$ where:*

$$\begin{aligned} r(J,\tau) &= \min_{\forall 0 \leq \Delta I < \infty} (\Delta I | \mathscr{C}(J + \Delta I) \geq \Upsilon(J) c_\tau + \Phi(J + \Delta I, \Gamma_{hp(\tau)})) \\ r(\tau) &= max_{\forall 0 \leq J \leq \mathscr{B}(\Gamma)} (r(J,\tau) \end{aligned}$$

J is less or equal than the busy period ($J \leq \mathscr{B}(\Gamma)$). This minimum response time has to be lower than the deadline of the task.

6. Approximation

To limit the number of test intervals and therefore the computational complexity we integrate the approximation approach of [**3**]. We can now integrate the approximation directly into the model. We allow the approximation of an event element to start after the necessary number of test intervals are reached globally for this element, independently in which period of the parent event element this happens. In case that the event element $\hat{\theta}$ is a child element of another (parent) event element $\hat{\theta}'$ we have to distinguish for $\hat{\theta}'$ between those periods in which $\hat{\theta}$ is evaluated exactly and those in which $\hat{\theta}$ is approximated. To do this it is necessary to split $\hat{\theta}'$ at the last exactly considered interval of $\hat{\theta}$.

6.1. Case simple sequence with gradient. Let us consider first a simple hierarchical event element: $\hat{\theta} = \{(T, a, l, G, \emptyset)\}$

$\hat{\theta}^k$ is the approximative counter-part for $\hat{\theta}$ starting with the approximation after k exactly considered test intervals. $\hat{\theta}^k$ is modeled by:

$$\hat{\theta}^k = \{(\infty, 0, l_A, 0, \hat{\theta}), (\infty, a_A, l, G, \emptyset), (\infty, a_B, \infty, \frac{l}{T}, \emptyset)\}$$

with $l_A = kl$, $a_A = a + kT$, $a_B = a_A + \frac{l}{G}$. For the special case with $G = \infty$ we have $a_A = a_B$.

EXAMPLE 6.1. Let us, for example consider $\hat{\Theta} = \{(10, 0, 3, \frac{1}{2}, \emptyset)\}$. The approximation $\hat{\Theta}^5$ for $\hat{\Theta}$ after $k = 5$ exactly considered test intervals is given by

$$\hat{\Theta}^5 = \{(\infty, 0, 15, 0, \{(10, 0, 3, \frac{1}{2}, \emptyset)\}), (\infty, 50, 3, \frac{1}{2}, \emptyset), (\infty, 56, \infty, \frac{3}{10}, \emptyset)\}$$

where $l_A = 5\cdot 3 = 15$, $a_A = 0+5\cdot 10 = 50$, $a_B = 50+\frac{3}{\frac{1}{2}} = 56$. We can simplify this example to:

$$\hat{\Theta}^5 = \{(\infty,0,18,0,\{(10,0,3,\frac{1}{2},\emptyset)\}),(\infty,56,\infty,\frac{3}{10},\emptyset)\}$$

Consider another example $\hat{\Theta} = \{(10,2,3,\infty,\emptyset)\}$ with $G=\infty$. The approximation $\hat{\Theta}^{10}$ is given by:

$$\hat{\Theta}^{10} = \{(\infty,0,30,0,\{(10,2,3,\infty,\emptyset)\},(\infty,103,3,\infty,\emptyset),(\infty,103,\infty,\frac{3}{10},\emptyset)\}$$

or shorter:

$$\hat{\Theta}^{10} = \{(\infty,0,33,0,\{(10,2,3,\infty,\emptyset)\},(\infty,103,\infty,\frac{3}{10},\emptyset)\}$$

6.2. Approximation of one-level child element. Let us consider a hierarchical event sequence with one child element:

$$\begin{aligned}\hat{\theta} &= (T,a,l,0,\hat{\theta}')\\ \hat{\theta}' &= (T',a',l',G',\emptyset)\end{aligned}$$

$\hat{\Theta}^k$ is given in this case by:

$$\begin{aligned}\hat{\Theta}^k = \; & \{(\infty,0,l_A,0,\hat{\theta}^\circ),\\ & (\infty,a_A,kl-l_A,0,\{(T,a',l',G',\emptyset),\\ & (T,a'+\frac{l'}{G'},l-l',\frac{l'}{T'},\emptyset)\}),(\infty,a_B,x,\infty,\emptyset),(\infty,a_B,\infty,\frac{l}{T},\emptyset)\}\end{aligned}$$

The first element of $\hat{\Theta}^k$ models the part in which the child-element $\hat{\theta}'$ is considered exactly. In case that the first possible approximation interval for $\hat{\theta}'$ occures within the first period of $\hat{\theta}$, we have to start the approximation within this first period of $\hat{\theta}$. Otherwise it would not be possible to find a reasonable bound for the number of considered test intervals for $\hat{\theta}'$. So $\hat{\theta}^\circ$ depends on whether $l \le kl'$ or $l > kl'$. We have

$$\begin{aligned}\hat{\theta}^\circ &= \begin{cases}\hat{\theta} & l \le kl'\\ \{(T,0,l,0,\hat{\theta}'^k)\} & l > kl'\end{cases}\\ \hat{\theta}'^k &= \{(\infty,0,kl_{\hat{\theta}},0,\hat{\theta}),(\infty,kT_{\hat{\theta}},l_{\hat{\theta}},G_{\hat{\theta}},\emptyset),(\infty,kT_{\hat{\theta}}+\frac{l_{\hat{\theta}}}{G_{\hat{\theta}}},\infty,\frac{l_{\hat{\theta}}}{T_{\hat{\theta}}},\emptyset)\}\end{aligned}$$

The calculation of l_A, l_B, a_A, a_B and a_C are done as follows:

$$\begin{aligned}l_A &= \begin{cases}\left\lceil\frac{kl'}{l}\right\rceil l & l \le kl'\\ l & l > kl'\end{cases}\\ a_A &= \begin{cases}\left\lceil\frac{kl'}{l}\right\rceil T+a & l \le kl'\\ T+a & l > kl'\end{cases}\\ a_B &= kT+a+a'\end{aligned}$$

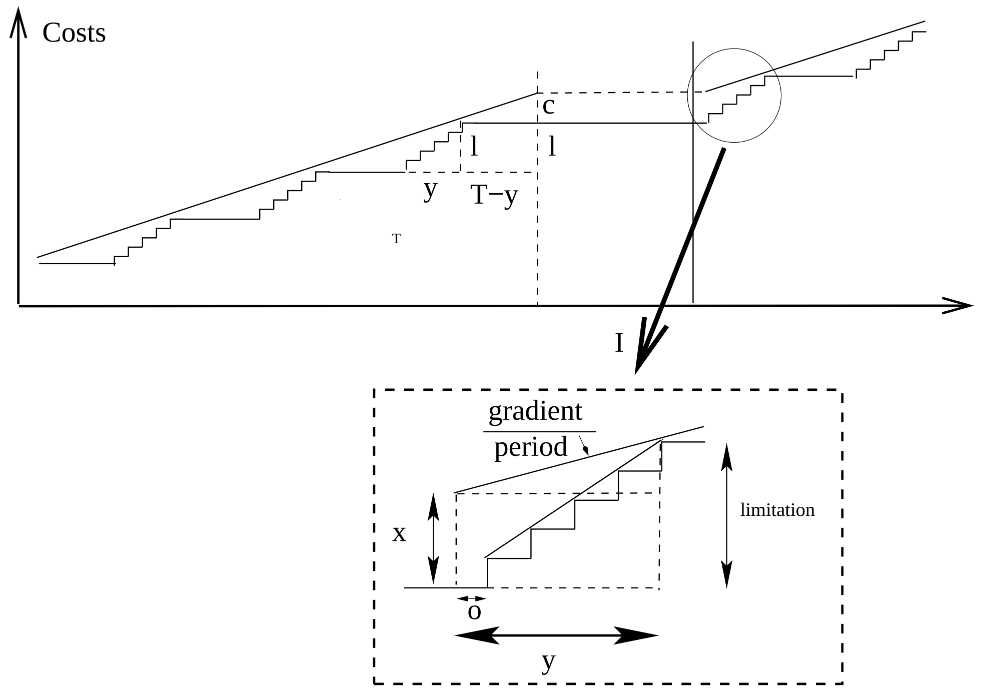

FIGURE 9. Case $\hat{\theta}'$ approximated, $\hat{\theta}$ not approximated

The approximation of $\hat{\theta}'$ can be done by an element $\hat{\theta}'^k$ with a gradient $G_{\hat{\theta}'^k} = \frac{l'}{T'}$.

When starting finally the approximation of $\hat{\theta}$ a cost-offset x is required to ensure that the approximated function $\Upsilon(\Delta I, \hat{\theta}^k)$ is always equal or higher than the exact function $\Upsilon(\Delta I, \hat{\theta})$. Figure 9 outlines this situation. This cost-offset is necessary as a new period of the parent element splits the approximation of the child element. The calculation of x can be done as follows:

$$\begin{aligned} l - x &= y\frac{l}{T} \\ x &= l\left(1 - \frac{y}{T}\right) \end{aligned}$$

y gives the interval between the start of the child element $\hat{\theta}'$ and the point in time in which the limitation of $\hat{\theta}$ is reached. The reaching of the limitation is calculated using the approximative description of the child elements of $\hat{\theta}$ with the seperate consideration of every first event of $\hat{\theta}$. For a simple child element $\hat{\theta} = \{(T, a, l, 0, \hat{\theta}')\}$ with $\hat{\theta}' = \{(T', a', l', \infty, \emptyset)\}$ this value y is given by

$$\begin{aligned} (y - a') \cdot \left(\frac{l'}{T'}\right) &= l - l' \\ y &= \frac{l - l'}{\frac{l'}{T'}} + a' = T'\frac{l}{l'} - T' + a' \end{aligned}$$

Hence for x:

$$x = l - \frac{T'l^2}{Tl'} + \frac{T'l}{T} - \frac{a'l}{T}$$

EXAMPLE 6.2. Let us consider the example hierarchical event sequence:

$$\hat{\Theta} = \{(80,2,16,0,\hat{\Theta}')\}, \hat{\Theta}' = \{(10,2,3,\infty,\emptyset)\}$$

For the approximation $\hat{\Theta}^{10}$ we get the values:

$$\begin{aligned}
l_A &= \left\lceil \frac{kl'}{l} \right\rceil l = \left\lceil \frac{10\cdot 3}{16} \right\rceil 16 = 32 \\
a_A &= \left\lceil \frac{kl'}{l} \right\rceil T + a = \left\lceil \frac{10\cdot 3}{16} \right\rceil 80 + 2 = 162 \\
a_B &= kT + a = 10\cdot 80 + 2 = 802 \\
y &= T'\frac{l}{l'} - T' + a' = 10\frac{16}{3} - 10 + 2 = 45.3333 \\
x &= l\left(1 - \frac{y}{T}\right) = 16\left(1 - \frac{45.333}{80}\right) = 6.9333
\end{aligned}$$

$$\begin{aligned}
\hat{\Theta}^{10} = \; & \{(\infty,0,32,0,\{(80,2,16,0,\{(10,2,3,\infty,\emptyset)\})\}), \\
& (\infty,162,128,0,\{(\infty,2,3,\infty,\emptyset),(\infty,2,\infty,\frac{3}{80},\emptyset), \\
& (80,2,13,\frac{3}{10},\emptyset)\},(\infty,802,6.9333,\infty,\emptyset),(\infty,802,\infty,\frac{16}{80},\emptyset)\}
\end{aligned}$$

6.3. Approximation of n-level child element. Let us consider the following hierarchical event element with two levels of child elements

$$\begin{aligned}
\hat{\theta} &= \{(T,a,l,0,\hat{\theta}')\} \\
\hat{\theta}' &= \{(T',a',l',0,\hat{\theta}'')\} \\
\hat{\theta}'' &= \{(T'',a'',l'',G'',\emptyset)\}
\end{aligned}$$

We consider the approximation $\hat{\theta}^k$. $\hat{\theta}^k$ is given by

$$\begin{aligned}
\hat{\theta}^k = \; & \{(\infty,0,l_A,0,\hat{\theta}^\circ{}_1),(\infty,a_A,l_B,0,\hat{\theta}^\circ{}_2), \\
& (\infty,a_B,l_C,0,\{(T,a',x',\infty,\emptyset),(T,a',l-x',\frac{l'}{T'},\emptyset)\}), \\
& (\infty,a_C,x,\infty,0),(\infty,a_C,\infty,\frac{l}{T},\emptyset)\}
\end{aligned}$$

$\hat{\theta}^\circ{}_1$ depends on whether $l \le kl''$ or $l > kl''$. We have

$$\hat{\theta}^\circ{}_1 = \begin{cases} \hat{\theta} & l \le kl'' \\ \{(T,0,l,0,\hat{\theta}'^k)\} & l > kl'' \end{cases}$$

$$\hat{\theta}'^k = \{(\infty,0,kl_{\hat{\theta}},0,\hat{\theta}),(\infty,kT_{\hat{\theta}},l_{\hat{\theta}},G_{\hat{\theta}},\emptyset),(\infty,kT_{\hat{\theta}}+\frac{l_{\hat{\theta}}}{G_{\hat{\theta}}},\infty,\frac{l_{\hat{\theta}}}{T_{\hat{\theta}}},\emptyset)\}$$

$\hat{\theta}^{\circ}{}_2$ depends on whether $l \leq kl'$ or $l > kl'$. We have

$$\hat{\theta}^{\circ}{}_2 = \begin{cases} \emptyset & l \leq kl' \\ \{(T', a'', l'', G'', 0), (T', a'' + \frac{l''}{G''}, l' - l'', \frac{l''}{T''}, \emptyset)\} & l > kl' \end{cases}$$

The calculation of l_A, a_A and l_B is easy and straight forward:

$$\begin{aligned} l_A &= \begin{cases} \left\lceil \frac{kl''}{l} \right\rceil l & l \leq kl'' \\ l & l > kl'' \end{cases} \\ l_B &= \begin{cases} \left\lceil \frac{kl'}{l} \right\rceil l - l_A & l \leq kl' \\ 0 & l > kl' \end{cases} \\ l_C &= kl - (l_A + l_B) \\ a_A &= \begin{cases} \left\lceil \frac{kl''}{l} \right\rceil T + a' + a & l \leq kl'' \\ T + a' + a & l > kl'' \end{cases} \\ a_B &= \begin{cases} \left\lceil \frac{kl'}{l} \right\rceil T & l \leq kl' \\ T & l > kl' \end{cases} \\ a_C &= kT + a \end{aligned}$$

The calculation of x' is the same as the calculation for x in the previous section. We have

$$\begin{aligned} y' &= T'' \frac{l'}{l''} - T'' + a'' \\ x' &= l' \left(\frac{T' - y'}{T'} \right) \end{aligned}$$

The calculation of x and y is similar but using the approximation of $\hat{\theta}''$. We have

$$\begin{aligned} (y - a) \cdot (\frac{l'}{T'}) &= l - x' \\ y &= \frac{lT'}{l'} - \frac{x'T'}{l'} + a' \\ x &= l \left(\frac{T - y}{T} \right) \end{aligned}$$

Note that, if setting $x'' = l''$ the calculation of x' and y' on the one side and x and y on the other side are the same. Therefore the proposed description for $\hat{\Theta}^k$ can be generalized to handle event sequences with n-level child event sequences. The calculation is visualized in figure 9.

In the following we will demonstrate a example by extending the example 6.2.

EXAMPLE 6.3. Let us consider the example hierarchical event sequence $\hat{\Theta}$:

$$\hat{\Theta} = \{(1000, 10, 100, 0, \hat{\Theta}')\}$$

$$\begin{aligned}
\hat{\Theta}' &= \{(80,2,16,0,\hat{\Theta}'')\}\\
\hat{\Theta}'' &= \{(10,2,3,\infty,\emptyset)\}
\end{aligned}$$

For an approximation $\hat{\Theta}^{10}$ in which $k = 10$ test intervals are considered exactly we get the values:

$$\begin{aligned}
y' &= \frac{16-3}{\left(\frac{3}{10}\right)} + 2 = 45.3333\\
x' &= 16 \cdot \left(\frac{80-45.3333}{80}\right) = 6.9333\\
y &= \frac{100-6.9333}{\left(\frac{16}{80}\right)} + 2 = 467.333\\
x &= 100 \cdot \left(\frac{1000-67.3335}{1000}\right) = 53.2667
\end{aligned}$$

$$\begin{aligned}
\hat{\Theta}^{10} &= \{(\infty,0,100,0,\hat{\Theta}^{10}_{2,1}),(\infty,1012,100,0,\{(\infty,2,3,\infty,\emptyset),(\infty,2,\infty,\frac{3}{80},\emptyset),\\
&(80,2,13,\frac{3}{10},\emptyset)\}),(\infty,2010,800,0,\{(\infty,2,6.9333,\infty,\emptyset),(\infty,2,\infty,\frac{6.9333}{1000},\emptyset),\\
&(1000,2,93.0667,\frac{16}{80},\emptyset)\}),(\infty,10010,53.2667,\infty,\emptyset),(\infty,10010,\infty,\frac{100}{1000},\emptyset)\}\\
\hat{\Theta}^{10}_{2,1} &= \{(\infty,0,32,0,\{(80,2,16,0,\{(10,2,3,\infty,\emptyset)\})\}),(\infty,162,3,\infty,\emptyset),\\
&(\infty,162,\infty,\frac{3}{80},\emptyset),(80,162,13,\frac{3}{10},\emptyset)\}
\end{aligned}$$

6.4. Approximation of element with several child elements. A hierarchical event sequence with several child elements can be transferred into a normalized hierarchical event sequence in which each event sequence element has only one child element. Each element matches one of the previous pattern and can therefore be approximated following the rules for this pattern. The overall approximation of the event sequence is than only a merge of the event sequences of the single pattern.

6.5. Required number of test intervals. In those cases in which the approximation of the child element starts within the completion of the first period of the parent element we cannot postpone it until the first period of the parent. It would not be possible to bound the number of test intervals for the child hierarchical event element.

EXAMPLE 6.4. Consider the following example:

$$\begin{aligned}
\hat{\theta}_{10} &= \{10000,0,4000,0,\{\hat{\theta}_{11}\}\}\\
\hat{\theta}_{11} &= \{10,0,5,\infty,\emptyset\}
\end{aligned}$$

Again the approximation may start after 100 test-intervals. The approximated event element can be written as follows:

$$\begin{aligned}
\hat{\theta}_{10}^{100} &= \{(\infty,0,4000,0,\{\hat{\theta}_{11}^{100}\}),\\
&\quad(\infty,10000,396000,0,\{(\infty,0,5,\infty,\emptyset),(\infty,0,\infty,\frac{5}{10000},\emptyset),\\
&\quad(10000,0,3995,\frac{5}{10},\emptyset)\}),\\
&\quad(\infty,1000000,804,\infty,\emptyset),(\infty,1000000,\infty,\frac{4000}{10000},\emptyset)\}\\
\hat{\theta}_{11}^{100} &= \{(\infty,0,500,\{(10,0,5,\infty,\emptyset)\}),(\infty,1000,5,\infty,\emptyset),(\infty,1000,\infty,\frac{5}{10},\emptyset)\}
\end{aligned}$$

Postponing the approximation of the child up to the end of the first period of the parent would cost 3000 additional test intervals. We can still find a simple bound on the required number of test intervals. For those cases in which the approximation does not start within the first period, the number of test intervals for one period of the parent event element has to be less than the approximation bound k. Otherwise the approximation would be allowed somewhere within the first period. Therefore the maximum number of test intervals we have to additionally consider due to the postponing is bounded also by k, leading to a total bound of $2k$.

6.6. Splitting points. The splitting points are the points in which the parent element is splitted to destinguish between the non-approximated and the approximated part of one of its child elements. In general, the parent element is splitted at the first of its completed period which is greater than the first possible approximation interval of the child element. Each element can require as many splitting points as its total child-set has members. The total child-set contains its children, the children of its children and so on. The parent chain contains the parent element of an element, the parent of the parent element and so on.

For reason of simplification we consider only normalized hierarchical event sequences, in which each $\hat{\theta}$ can only have one direct child element at most.

Let $\hat{\theta}_1$ be the lowest-level child element and $\hat{\theta}_n$ be the highest level parent element. The splitting point for an element $\hat{\theta}_i$ is determined by the upper-most member $\hat{\theta}_j$ of a parent chain for which the first possible approximation interval for k exactly considered test intervals $t_{\hat{\theta}_i,k}$ of $\hat{\theta}_i$ is larger than the end of the first completed period of $\hat{\theta}_j$. This first complete period is given by $a_{\hat{\theta}_j} + T_{\hat{\theta}_j}$, so $t_{\hat{\theta}_i} > a_{\hat{\theta}_j} + T_{\hat{\theta}_j}$. The splitting point is the first start of a new period of $\hat{\theta}_t$ after $t_{\hat{\theta}}$, so

$$s_{i,j}^{k} = min(\Delta I|\Delta I = a_{t_i} + kT_{t_i} \wedge \Delta I \geq t_{\hat{\theta}_j,k})$$

It is necessary to split each element of the parent-child chain between $\hat{\theta}_t$ and $\hat{\theta}_c$ at this point. All members of the parent chain of $\hat{\theta}_{t_i}$, which are of cause also member of the parent chain of $\hat{\theta}_i$, are splitted at their first period instead, so

$$\forall j > t \ | \ s_{i,j} = a_{\hat{\theta}_j} + T_{\hat{\theta}_j}$$

In general we get a matrix of possible splitting points:

LEMMA 6.5. *(Splitting points) Let $\hat{\theta}_1, .., \hat{\theta}_n$ be a set of hierarchical event elements with $\hat{\theta}_1 = (T_1, a_1, l_1, G_1, \emptyset)$ and $\hat{\theta}_i = \{T_i, a_i, l_i, 0, \hat{\theta}_{i-1})$ for $0 < i \leq n$. Let $s^k_{i,j}$ be the splitting points for element j on the event element $\hat{\Theta}_i$ with the minimum number of k test-intervals considered exactly for $\hat{\theta}_j$. Let $t_{j,k}$ denote the first possible approximated test interval of $\hat{\theta}_j$ after k exact test intervals. $s^k_{i,j}$ can be calculated:*

$$\begin{aligned}
s^{k'}_{i,j} &= min(x | x = a_i + yT_i, y \in N, x \geq t_{j,k}) \\
s^k_{i,j} &= \begin{cases} s^{k'}_{i,j} & s^k_{i,j} < a_{i+1} + T_{i+1} \\ s^k_{i+1,j} & else \end{cases} \\
s^k_{i,0} &= a_i \\
s^k_{n,j} &= s^{k'}_{n,j}
\end{aligned}$$

PROOF. The first completed period of the hierarchical event element $\hat{\theta}_i$ after the first possible approximation start for the hierarchical event element $\hat{\theta}^k_j$ gives the potential splitting point $s^{k'}_{i,j}$. The resulting splitting point $s_{i,j}$ is only in those cases identical to the potential splitting point $s^{k'}_{i,j}$ in which either $\hat{\theta}_i$ is the top-level parent element $(i = n)$ or $s^{k'}{}_{i,j}$ is smaller than the end of the first period of the parent element $\hat{\theta}_{i+1}$. In all other cases, the completion point $s^k_{i,j}$ is identical to the corresponding completion point of the parent element of $\hat{\theta}_i$, $s_{i+1,j}$, which can again be identical to the splitting points of the $(i+1)$-th parent element and so on. □

We can calculate the approximated hierarchical event streams using these splitting points.

LEMMA 6.6. *Let us consider a chain of hierarchical event streams $\hat{\Theta}_1, ..., \hat{\Theta}_n$ with $\underset{j<n}{\hat{\Theta}_j} = \{(T_{\hat{\theta}_j}, a_{\hat{\theta}_j}, l_{\hat{\theta}_j}, 0, \hat{\Theta}_{j+1})\}$ and $\hat{\Theta}_n = \{(T_{\hat{\theta}_j}, a_{\hat{\theta}_j}, l_{\hat{\theta}_j}, G_{\hat{\theta}_n}, \emptyset)\}$. The approximated event elements are given by the following equations ($s_{0,j} = 0$):*

$$\begin{aligned}
\hat{\Theta}^k_j &= \{\hat{\theta}'_{i,j} | i + j \leq n \wedge s_{i,j} \neq s_{i,j-1}\} \cup \hat{\Theta}_{j,j+1} \\
\hat{\Theta}^k_{i,i} &= \{(\infty, s_{i,j-1}, x_{\hat{\theta}_i}, \infty, \emptyset), (\infty, s_{i,j-1}, \infty, \frac{l_{\hat{\theta}_i}}{T_{\hat{\theta}_i}}, \emptyset)\} \\
\hat{\Theta}^k_{i,i+1} &= \{(\infty, s_{i,i}, x_{\hat{\theta}_i}, \infty, \emptyset), (\infty, s_{i,i}, \infty, \frac{l_{\hat{\theta}_j}}{T_{\hat{\theta}_j}}, \emptyset)\}
\end{aligned}$$

$$\hat{\theta}_{i,j} = \begin{cases} (\infty, s_{i,j-1}, \frac{s_{i,j}-s_{i,j-1}}{T_{\hat{\theta}_i}} l_{\hat{\theta}_i}, 0, & s_{i,j} \neq s_{i+1,j} \\ \quad \{(T_{\hat{\theta}_i}, 0, l_{\hat{\theta}_i}, G_{\hat{\theta}_i}, \hat{\Theta}_{i-1,j} & \\ (T_{\hat{\theta}_i}, a_{\hat{\theta}_i}, l_{\hat{\theta}_i}, G_{\hat{\theta}_i}, \hat{\Theta}_{i-1,j} & s_{i,j} = s_{i+1,j} \end{cases}$$

$$\hat{\Theta}'_{i,j} = \begin{cases} \{\hat{\theta}'_{i,j}\} & s_{i+1,j} \neq s_{i+1,j+1} \\ \{\hat{\theta}'_{i,j}\} \cup \hat{\Theta}'_{i,j+1} & s_{i+1,j} = s_{i+1,j+1} \end{cases}$$

$$x_{\hat{\theta}_i} = l_{\hat{\theta}_i}\left(1 - \frac{y_{i,j}}{T_{\hat{\theta}_i}}\right)$$

$$y_{\hat{\theta}_i} = \frac{l_{\hat{\theta}_i} - x_{\hat{\theta}_{i-1}}}{\frac{l_{\hat{\theta}_{i-1}}}{T_{\hat{\theta}_{i-1}}}} + a_{\hat{\theta}_{i-1}}$$

$$x_{\hat{\theta}_1} = l_{\hat{\theta}_1}$$

PROOF. Only for those splitting points $s^k_{i,j}$ being different from their predecessor splitting point $s^k_{i,j-1}$ a hierarchical event element can be constructed. The other splitting points would lead to elements generating no events. For the construction of the element we have to distinguish, whether the splitting point is identical to the corresponding splitting point of the parent element or whether it is a new value on its own. In the first case $(s^k_{i,j} = s^k_{i+1,j})$, the limitation is simply inherited from the parent element, in the second case $(s^k_{i,j} \neq s^k_{i+1,j})$, the limitation has to be calculated by distributing the previous limitation on the new parts. Note that $\frac{s^k_{i,j}-s^k_{i,j-1}}{T_{\hat{\theta}_i}} \in N$ by definition and therefore the limitation of the new parts are multiple of the limitation of the single elements. □

The lemma summarizes (and simplifies) the results of the previous sections. Each element of the top-parent event sequence and therefore each chain of elements can be considered seperately.

7. Example

EXAMPLE 7.1. Fig. 10 shows the advanced approximation for the event bound function of the event stream $\hat{\Theta}_7 = \{(20, 0, 10, 0, (2, 0, 2, \infty, \emptyset))\}$ and compares it with the description by Sympta/S and by the real-time calculus. For Sympta/S we have used an execution time of 2, a period of 4, a jitter of 10 and a minimum distance between two events of 2 time units. The lines of Sympta/S and the real-time calculus are nearly identical with the exception that Sympta/S models discrete events. The line for the new model in is exact form is always equal or below both other lines and in its approximated form it is below and the beginning and than equal to the real-time calculus curve. The degree of approximation is freely selectable. Note, that the event discrete modeling of the Sympta/S approach requires additional effort for the analyis.

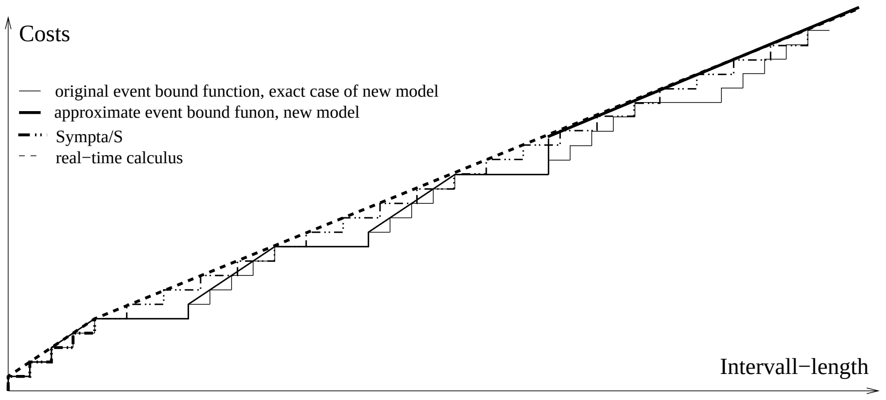

FIGURE 10. Approximated hierarchical event bound function

The event stream embodies of bursts with five events. The advanced approximated event stream with an approximation after three events is has the following separation points: $s_{1,0} = 0$, $s_{1,1} = 20$, $s_{2,0} = 0$, $s_{2,1} = 4$, $s_{2,2} = 60$.

For x and y we have the values:

$$y = \frac{l - l'}{\frac{l'}{T'}} + a' = \frac{10-2}{\frac{2}{2}} + 0 = 8$$

$$x = l\left(1 - \frac{y}{T}\right) = 10\left(1 - \frac{8}{20}\right) = 6$$

It is given by the following description:

$$\hat{\Theta}_7^3 = \{(\infty, 0, 10, 0, \{(20, 0, 6, 0, (2, 0, 2, \infty, \emptyset)), (\infty, 10, 4, 1, \emptyset)\}), (\infty, 20, 12, 0, \{(20, 0, 2, \infty, \emptyset), (20, 0, 8, \frac{2}{20}, \emptyset)\}), (\infty, 60, 6, \infty, \emptyset), (\infty, 60, \infty, \frac{1}{2}, \emptyset)\}$$

Such a description limits the maximum number of test intervals for each hierarchical event element separately. In the example five test intervals for the child element and four test intervals for the parent element are required. This approximation does also hold with event sequences as child elements instead of a simple event element as in the example.

EXAMPLE 7.2. We have an event element $\hat{\theta}_8 = \{1000, 0, 150, 0, \{\hat{\theta}_{8,b}\}\}$ with a child event element $\hat{\theta}_{8,b} = \{10, 0, 5, \infty, \emptyset\}$. The approximation is allowed after 100 test intervals, that means within the 4-th period of the $\hat{\theta}_8$.

$$y = \frac{l_{\hat{\theta}_8} - l_{\hat{\theta}_{8,b}}}{\frac{l_{\hat{\theta}_{8,b}}}{T_{\hat{\theta}_{8,b}}}} - a_{\hat{\theta}_{8,b}} = \frac{150-5}{\frac{5}{10}} - 0 = 290$$

$$x = l_{\hat{\theta}_8}\left(1 - \frac{y}{T_{\hat{\theta}_8}}\right) = 150\left(1 - \frac{290}{1000}\right) = 106.5$$

For the example the resulting approximated event stream $\hat{\theta}_8^{100}$ reads as follows:

$$\begin{aligned}\hat{\theta}_8^{100} \quad = \quad & \{(\infty,0,600,0,\hat{\theta}_8),(\infty,4000,14400,0,\{(\infty,4000,5,\infty,\emptyset),(\infty,4000,\infty,\frac{5}{1000},\emptyset), \\ & (1000,0,145,\frac{5}{10},\emptyset)\}),(\infty,100000,106.5,\infty,\emptyset),(\infty,100000,\infty,\frac{150}{1000},\emptyset)\}\end{aligned}$$

8. Integration with Real-Time Calculus

In the following we will propose a new approximative model for the curves of the real-time calculus allowing a less pessimistic modelling of the curves. It guarantees the approximation error. In [**3**] such an approximation was proposed for the periodic task model with EDF scheduling. It is now extended to distributed systems and is integrated in the model itself.

8.1. Model. We model each curve of the real-time calculus by a test list $Te = \{te\}$ consisting of a set of test-list elements $te = (\Delta I, c, G)$ each modelling one segment of the curve. ΔI is an interval determining the start point of the segment, c are costs additionally occuring at the start of the segment and G determines the gradient within the segment and is the increment between the gradient within the segment and the gradient within the previous segment. The total gradient is the sum of all gradients of previous test list elements with an interval $\Delta I' < \Delta I$.

For example four events with a distance of 10 to each other and a execution time of 2 can be modeled by a test list $Te = \{(0,2,0),(10,2,0),(20,2,0),(30,2,0)\}$. The proposed model is not limited to model time discrete events, it can also model the capacity and allows to describe systems with varying capacity over the time. The gradient is usefull to model the capacities or the remaining capacities of processing units (PUs). The standard case in which a PU can handle one time unit execution time in one time unit can be modled by $te = (0,0,1)$. More sophisticated service functions like a case in which only half of the processor capacity is available during the first 100 time units can also be described by a few elements $Te = \{(0,0,\frac{1}{2}),(100,0,\frac{1}{2})\}$. Note that the gradients are always only the differences between the resulting gradient and the previous gradient. Therefore in the example the functions has a gradient of $\frac{1}{2}$ for the first 100 time units and after them a resulting gradient of 1 for the remaining time.

8.1.1. *Approximation.* General event models generates an infinit set of events and would therefore require an infinit number of test-list elements. In the periodic task model for example each task $\tau = (T,c,d)$ represents an infinit number of jobs sharing the same worst-case execution time c and relative deadline d and having a periodic release pattern with period T. An approximation is necessary to bound this number of elements and to allow a fast analysis. The idea for the approximation is to consider the first n jobs of a task exactly and to approximation the following jobs by the specific utilization of the task. This approximation can be represented by the test-list model. The selection of the parameter n

allows a trade-off between the exactness and the analysis effort. For example a task $\tau = (10,2,6)$ is represented by a test list $Te = \{(0,2,0),(10,2,0),(20,2,0),(30,2,\frac{2}{10})\}$with 4 as degree of exactness.

DEFINITION 8.1. *([**3**]) Let Γ be any taskset bound on any resource ρ. Let ρ_l be the resource with the minimum capacity on which Γ is scheduable. An approximation with approximation error ε is a test algorithm which*

(1) *returns "non-scheduable" in those cases in which Γ on ρ is non-scheduable*
(2) *returns scheduable in all those scheduable cases in which $\mathscr{C}(\rho) \geq \frac{1}{1-\varepsilon}\mathscr{C}(\rho_l)$*
(3) *can returns either "scheduable" or "non-scheduable" in all cases with $\mathscr{C}(\rho_l) \leq \mathscr{C}(\rho) \leq \frac{1}{1-\varepsilon}\mathscr{C}(\rho_l)$*

This idea can be used in a similar way for all other task and event models. Formally, a periodic task τ with $\tau = (T,c,d)$ and a degree of exactness of n can be transfeered into a test list Te with the elements $Te = \{(0,c_\tau,0),(T_\tau,c_\tau,0),(2T_\tau,c_\tau,0),...,(nT_\tau,c_\tau,\frac{c_\tau}{T_\tau})\}$ with deadline d_τ. We can transfeer this test list further in a test representing the demand bound function $\Psi(\Delta I,\tau)$ for τ by shifting it by the deadline $(Te' = \{(d_\tau,c_\tau,0),(T_\tau+d_\tau,c_\tau,0),...,(nT_\tau+d_\tau,c_\tau,\frac{c_\tau}{T_\tau})\})$.

The service functions might also require an approximation. But in contrary to above it is necessary to underestimate the original functions. A service function of a processor which is not available every 100 time units for 2 time units due to operation system processes can be modeled with an degree of exactness of 4 by $Te = \{(2,0,1)$, $(100,0,-1)$, $(102,0,1)$, $(200,0,-1)$, $(202,0,1)$, $(300,0,-1)$, $(302,0,\frac{98}{100})\}$.

8.1.2. *Event bound function.* The amount of events occuring in some intervals ΔI, therefore the value of the real-time calculus curves can be calculated with the following event bound function.

DEFINITION 8.2. An event bound function $\Upsilon(\Delta I)$ gives the amout of event which can occure at most in any interval of length ΔI.

The calculation can be done as follows:

$$\Upsilon(\Delta I, Te) \quad = \sum_{\substack{\forall te_i \in Te \\ \Delta I_{te_i} \leq \Delta I}} [(\Delta I - \Delta I_{te_i})G_{te_i} + c_{te_i}]$$

8.2. Real-Time Analysis. In the following we will show how an efficient schedulability analysis can be realized with the introduced model easily.

8.2.1. *EDF.* Schedulability analysis for EDF can be done using the processor demand criteria which was introduced by Baruah et al. [**5**], [**6**].

DEFINITION 8.3. *([**5**]) The demand bound function $\Psi(\Delta I,\Gamma)$ gives the cummulated execution requirement of those jobs having release time and deadline within ΔI.*

LEMMA 8.4. A task set scheduled with EDF keeps all deadlines if for every intervals $\Delta I > 0$ the demand bound function $\Psi(\Delta I, \Gamma)$ does not exceed the available capacity $\mathscr{C}(\Delta I, \rho)$ for ΔI:

$$\Psi(\Delta I, \Gamma) \le \mathscr{C}(\Delta I, \rho)$$

This can be rewritten as:

$$\mathscr{C}(\Delta I, \rho) - \Psi(\Delta I, \Gamma) \ge 0$$

PROOF. See [**5**] and [**3**] □

Both, the demand bound and the service function can be described by test lists as we have already seen. $\mathscr{C}(\Delta I, \Gamma) - \Psi(\Delta I, \Gamma)$ can be simplified to one test list. The overall demand bound function of the taskset is the sum of the demand bound functions of the single tasks:

$$\Psi(\Delta I, \Gamma) = \sum_{\forall \tau \in \Gamma} \sum_{\forall te \in Te_\tau} \Psi(\Delta I, Te)$$

The demand bound function of a single task can be derived out of the events bound function of this task by shifting this function by the value of the deadline:

$$\Psi(\Delta I, \Gamma) = \Upsilon(\Delta I - d, \Gamma)$$

So the resulting analysis for EDF reads:

$$\forall \Delta I \ge 0 \quad \Upsilon(\Delta I, Te') = \mathscr{C}(\Delta I, \rho) - \sum_{\forall \tau \in \Gamma} \sum_{\forall te \in Te_\tau} \Upsilon(\Delta I - d_\tau, Te) \ge 0$$

For the demand bound function a test list can be calculated out of the test lists of the event bound functions using the shift and add functions as we will defined in section 8.3.

In algorithm 1 we give the short implementation to proof the condition $\Upsilon(\Delta I, Te) \ge 0$ for all $\Delta I \ge 0$ and therefore to do the real-time analysis.

The best way to do this is to calculate and check the intervals of the test-list elements step-wise in rising order starting by $\Delta I = 0$. We have to test each element twice, once after the costs resulting of the previous gradient are added and once after the costs of the element are added. Otherwise, the situation can occure that the costs value can compensate a negative value of the functions which would therefore be undetectable.

8.2.2. *Analysis for static priorities.* The real-time analysis of systems with static priority scheduling requires another function, the request bound function $\Phi(\Delta I, \Gamma)$.

DEFINITION 8.5. *([**5**]) The request bound functions $\Phi(\Delta I)$ contains the amount of execution time requested by those events having occured within ΔI.*

Events occuring exactly at the end of ΔI are excluded:
$\Phi(\Delta I, Te) = \lim_{\Delta I' \to \Delta I} \Upsilon(\Delta I', Te) = \sum_{\substack{\forall te_i \in Te \\ \Delta I_{te_i} < \Delta I}} [(\Delta I - \Delta I_{te_i}) \cdot G_{te_i} + c_{te_i}]$

For the analysis it is necessary to consider each task seperatly.

Algorithm 1 Feasibility Analysis

```
Algorithm Superposition
Given: testList $Te$ (sorted with rising $\Delta I$)
$r = 0$; $G = 0$; $\Delta I_{old} = 0$;
FOR ALL ($te \in Te$)
   $r := r + (a_{te} - \Delta I_{old})G$
   IF ($r < 0$) THEN
      $\Rightarrow$not scheduable
   END IF
   $r := r + c_{te}$
   IF ($r < 0$) THEN
      $\Rightarrow$not scheduable
   END IF
   $\Delta I_{old} := \Delta I_{te}$; $G := G + G_{te}$
END WHILE
IF ($G < 0$) THEN
   $\Rightarrow$not scheduable
ELSE
   $\Rightarrow$scheduable
END IF
```

LEMMA 8.6. (similar to [**8**]) The worst-case response time of a task is given by:

$$r_\tau = min(\Delta I | \forall \Delta I' > 0 : \mathscr{C}(\Delta I', \tau) - \Phi(\Delta I', \tau) \geq 0)$$

Schedulability for a job of a task τ is given if $r_\tau \leq d_\tau$.

PROOF. See [**5**]. □

The schedulability analysis can also simply be done by checking for each $\Delta I \geq 0$ and each $\tau \in \Gamma$: $\Psi(\Delta I, \tau) \leq \mathscr{C}(\Delta I, \tau)$

$\mathscr{C}(\Delta I, \tau)$ denotes the capacity available for task τ within ΔI. For the task with the highest priority this is the capacity of the resource $\mathscr{C}(\Delta I, \rho)$. For all other task it is the remaining part of the capacity after all tasks with a higher priority have been processed. The calculation of this remaining capacity can be done for each task seperatly. The problem is that an amout of capacity reached for some intervals ΔI is also available for each larger interval $\Delta I'$ even if between ΔI and $\Delta I'$ a large amount of computation request occures, so that $\Phi(\Delta I', \tau) - \Phi(\Delta I, \tau) > \mathscr{C}(\Delta I', \tau) - \mathscr{C}(\Delta I, \tau)$. No part of this requested computation time can be processed within ΔI as this would require to process it before it is requested.

For the calculation of this remaining capacity the exceeding costs function is useful:

DEFINITION 8.7. ([**2**]) *Exceeding costs* $\Upsilon(\Delta I, \Gamma)$ *denotes those part of the costs requested within the interval* ΔI *by the taskset* Γ *which cannot be processed within* ΔI *with either scheduling due to the late request times.*

See figure 11 for some examples for exceeding costs. For example for the job $\psi_{1,i}$ arriving at time 18 and requesting 4 time units computation time at least 2 time units cannot

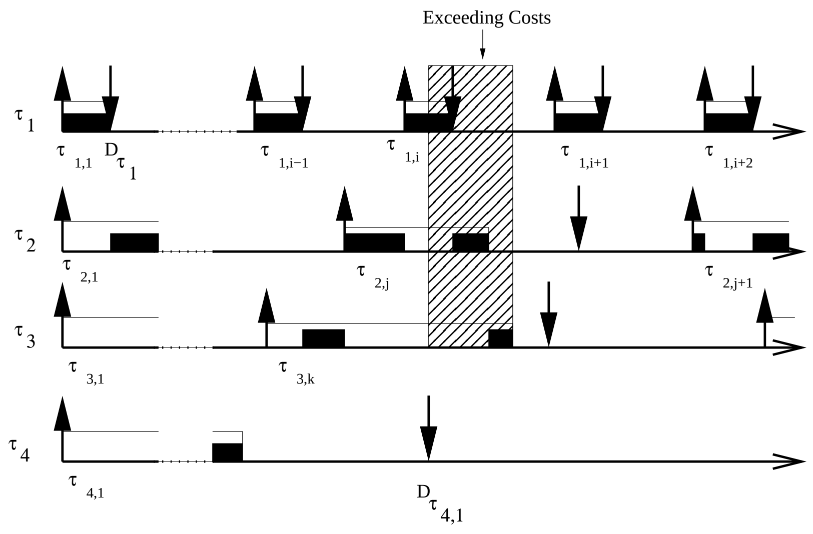

FIGURE 11. Exceeding costs

be processed within $\Delta I = 20$ even if the job fully gets the remaining processor time. The exceeding costs gets an even higher value taking other jobs into account. Job $\psi_{2,j}$ alone would not contribute to the exceeding costs, but together with job $\psi_{1,i}$ the contribution gets even higher than the contribution of job $\psi_{1,i}$ alone. The reason is that the jobs steal the capacity from each other. Only the sum of the exceeding computation time is relevant not from which task it is requested. The value and the calculation of the exceeding costs is independent of the concrete scheduling.

The exceeding cost function can be used for a simple schedulability analysis for systems with static priorities [**2**].

LEMMA 8.8. A task set Γ is scheduable if for each task $\tau \in \Gamma$ and each $\Delta I > 0$:

$$\Psi(\Delta I, \tau) + \Phi(\Delta I, hp(\tau)) - \Upsilon(\Delta I, hp(\tau)) \leq \mathscr{C}(\Delta I, \rho)$$

or if τ_{i-1} is the task with next higher priority than τ_i:

$$\Psi(\Delta I, \tau_i) + \Phi(\Delta I, \tau_{i-1}) - \Upsilon(\Delta I, \tau_{i-1}) \leq \mathscr{C}(\Delta I, \tau_{i-1})$$

The calculation of the remaining capacity can be therfore done by

$$\mathscr{C}(\Delta I, \tau_i) = \mathscr{C}(\Delta I, \tau_{i-1}) - \Phi(\Delta I, \tau_{i-1}) + \Upsilon(\Delta I, \tau_{i-1})$$

PROOF. See [**2**]. □

This allows a step-wise calculation of the remaining capacity and also an integration of the analysis for EDF and for fixed priority scheduling to one hierachical schedulability analysis. The algorithm 2 generates the test-list for the exceeding costs funtion $\Upsilon(\Delta I, \tau)$.

Algorithm 2 Exceeding-costs calculation

```
Algorithm exceeding cost calculation
Given: $Te_{in}$ // The result of $\Phi - \mathscr{C}$
// initialize values
$Te_{result} := \{\}$; $G := 0$; $c := 0$; $\Delta I_{old} := 0$
FOR all $te \in Te_{in}$
    IF $((c > 0) \vee (G > 0))$ THEN
        $c := c + (\Delta I_{te} - \Delta I_{old})G$
    END IF
    IF$(c \leq 0)$ THEN
        $\Delta I_{split} := \Delta I_{te} + \frac{c}{(-G)}$
        $Te_{result} := Te_{result} \cup (\Delta I_{split}, 0, -G)$
        $c := 0$
    END IF

    $c_{new} := c + c_{te}$
    IF $(c > 0 \wedge c_{new} > 0)$ THEN
        $Te_{result} := Te_{result} \cup te$
    ELSE IF $(c \leq 0 \wedge c_{te} > 0)$
        $Te_{result} := Te_{result} \cup (\Delta I_{te}, c_{te}, G_{te} + G)$
    ELSE IF $(c > 0 \wedge c_{new} \leq 0)$
        $Te_{result} := Te_{result} \cup (\Delta I_{te}, -c, -G)$
    ELSE IF $(c > 0 \wedge c_{new} \leq 0 \wedge G + G_{te} < 0)$
        $Te_{result} := Te_{result} \cup (\Delta I_{te}, -c, -G)$
    ELSE IF $(c > 0 \wedge c_{new} \leq 0 \wedge G + G_{te} \geq 0)$
        $Te_{new} := Te_{new} \cup (\Delta I_{te}, -c, G_{te})$
    ELSE IF $(c \leq 0 \wedge c_{new} \leq 0 \wedge G + G_{te} \geq 0)$
        $Te_{new} := Te_{new} \cup (\Delta I_{te}, 0, G + G_{te})$
    END IF
    $G := G + G_{te}$
    $c := max(c_{new}, 0)$
    $\Delta I_{old} := \Delta I_{te}$
END FOR
RETURN $Te_{result}$
```

Figure 12 visualizes its calculation. The exceeding cost function starts equal to the difference of the request bound function and the available capacity ($\Phi(\Delta I, \tau) - \mathscr{C}(\Delta I, \tau)$). It remains equal to this function until it drops below zero for the first time, e.g. more capacity is available than required by requested jobs. Than the exceeding cost function remains zero until the difference function starts rising again, e.g. new request arrives. Than the exceeding costs function will also rise and run further in parallel to the difference function but with a higher value.

8.2.3. *Practical issues.* Blocking time, scheduling overhead and the priority inheritance protocol can easily be integrated in the above equations. A blocking time b can

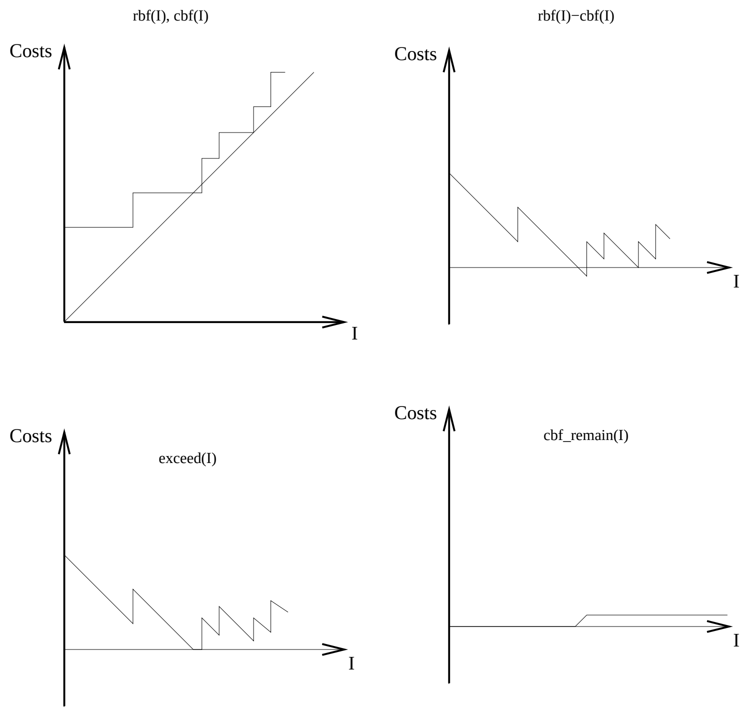

FIGURE 12. Calculation of the exceeding cost functions

be integrated by either adding b to the equations or by integrating the test-list element $te = (0, b, 0)$.

8.3. Operations and Basic Functions. In the following we will introduce some operations on test-lists and their implementations.

8.3.1. *Adding* $(+)$. The add-operation for two test lists can be simply realized by a union of the sets of test list elements of the two test lists:

DEFINITION 8.9. ($+$ operation) Let Te_A, Te_B, Te_{new} be test lists. If Te_{new} is the sum of Te_A and Te_B ($Te_{new} = Te_A + Te_B$) than for each interval ΔI the equation $\Upsilon(\Delta I, Te_{new}) = \Upsilon(\Delta I, Te_A) + \Upsilon(\Delta I, Te_B)$ is true.

LEMMA 8.10. *($+$ operation) The sum $Te_{new} = Te_A + Te_B$ can be calculated by the union of the event stream elements of Te_A, Te_B:*

$$Te_{new} = Te_A \cup Te_B$$

PROOF. The proof can be done using the definition of the hierarchical event bound function:

$$\begin{aligned}
\Upsilon(\Delta I, Te_{new}) &= \Upsilon(\Delta I, Te_A) + \Upsilon(\Delta I, Te_B) \\
&= \sum_{\substack{\forall te_i \in Te_A \\ \Delta I_{te_i} \leq \Delta I}} [(\Delta I - \Delta I_{te_i}) G_{te_i} + c_{te_i}] + \sum_{\substack{\forall te_i \in Te_B \\ \Delta I_{te_i} \leq \Delta I}} [(\Delta I - \Delta I_{te_i}) G_{te_i} + c_{te_i}] \\
&= \sum_{\substack{\forall te_i \in Te_A \cup Te_B \\ \Delta I_{te_i} \leq \Delta I}} [(\Delta I - \Delta I_{te_i}) G_{te_i} + c_{te_i}] = \Upsilon(\Delta I, Te_A \cup Te_B)
\end{aligned}$$

□

8.3.2. *Substracting* $(-)$. The subtraction can be led back to the addition by exchanging Te_B with its negation $-Te_B$. So we only need to define the negation of the test list.

DEFINITION 8.11. ($-$ operation) Let $Te' = -Te$. The negation of Te is defined by the negation of its corresponding hierarchical event bound function $\Upsilon(\Delta I, -Te) = -\Upsilon(\Delta I, Te)$. It is therefore only the substraction of two functions.

LEMMA 8.12. *($-$ operation) $Te' = -Te$ if for each test list element te of Te exists a corresponding counter element te' of Te' and vice versa differing only in the negation of the one-time costs and the gradient. We have $\Delta I_{te'} = \Delta I_{te}$, $c_{te'} = -c_{te}$ and $G_{\hat{\theta}'} = -G_{\hat{\theta}}$.*

In other words for each test list element te of Te the costs c_{te} and the gradient G_{te} are exchanged by their negation $-c_{te}$ and $-G_{te}$.

PROOF.

$$\begin{aligned}
-\Upsilon(\Delta I, Te) &= -\sum_{\substack{\forall te_i \in Te \\ \Delta I_{te_i} \leq \Delta I}} [(\Delta I - \Delta I_{te_i}) G_{te_i} + c_{te_i}] \\
&= \sum_{\substack{\forall te_i \in Te \\ \Delta I_{te_i} \leq \Delta I}} [(\Delta I - \Delta I_{te_i})(-G_{te_i}) + (-c_{te_i})] \\
&= \sum_{\substack{\forall te_i \in Te' \\ \Delta I_{te_i} \leq \Delta I}} [(\Delta I - \Delta I_{te_i}) G_{te_i} + c_{te_i}] \\
&= \Upsilon(\Delta I, Te')
\end{aligned}$$

□

The operation $(+)$ is cumulative, same as with the hierarchical event sequences, so we have $Te_A + Te_B = Te_B + Te_A$ and also $(Te_A + Te_B) + Te_C = Te_A + (Te_B + Te_C)$.

8.3.3. *Shift Operation* $(\rightarrow, \leftarrow)$. The shift operation can be realized by adding or subtracting the shift-value from each interval of all test list elements. We may get test list elements with negative intervals, which can be handled despite that the negative interval values are not defined.

DEFINITION 8.13. ($\rightarrow$ shift-operation) Let Te be a test list that is shifted right by the value t resulting in the test list $Te' = Te \rightarrow t$. The event bound functions have the following relationship:

$$\Upsilon(\Delta I, Te') = \begin{cases} \Upsilon(\Delta I - t, Te) & \Delta I \geq t \\ 0 & else \end{cases}$$

LEMMA 8.14. *$\Upsilon(\Delta I, Te) \rightarrow t = \Upsilon(\Delta I, Te')$ if Te' contains and only contains for each element te of Te an element $te' \in Te'$ having the following relations to Te: $\Delta I_{te'} = \Delta I_{Te} + t$, $c_{te'} = c_{te}$, $G_{te'} = G_{te}$*

The operation $Te' = Te \rightarrow t$ can be performed by only adding the value t to the interval ΔI_{te} of each event element $te \in Te$ for its corresponding counter-element te' of Te':

$$\Delta I_{te'} = t + \Delta I_{te}$$

PROOF.

$$\begin{aligned} \Upsilon(\Delta I - t, Te) &= \sum_{\substack{\forall te_i \in Te \\ \Delta I_{te_i} \leq \Delta I}} [((\Delta I - t) - \Delta I_{te_i}) G_{te_i} + c_{te_i}] \\ &= \sum_{\substack{\forall te_i \in Te \\ \Delta I_{te_i} \leq \Delta I}} [(\Delta I - (\Delta I_{te_i} + t)) G_{te_i} + c_{te_i}] \\ &= \sum_{\substack{\forall te_i \in Te' \\ \Delta I_{te_i} \leq \Delta I}} [(\Delta I - \Delta I_{te_i}) G_{te_i} + c_{te_i}] \\ &= \Upsilon(\Delta I, Te') \end{aligned}$$

□

The operation to shift a value left by the value t ($Te \leftarrow t$) can be defined in an equal way.

DEFINITION 8.15. ($\leftarrow$ shift-operation) Let Te be a test list that is shifted left by the value t resulting in the test list $Te' = Te \leftarrow t$. The event bound functions have the following relationship:

$$\Upsilon(\Delta I, Te') = \Upsilon(\Delta I + t, Te)$$

LEMMA 8.16. *$\Upsilon(\Delta I, Te) \leftarrow t = \Upsilon(\Delta I, Te')$ if Θ' contains and only contains for each test list element te of Te an element $te' \in Te'$ having the following relations to Te: $\Delta I_{te'} = \Delta I_{Te} - t$, $c_{te'} = c_{te}$, $G_{te'} = G_{te}$*

PROOF. The proof is similar to the proof for the right shift .

$$\Upsilon(\Delta I + t, Te) = \sum_{\substack{\forall te_i \in Te \\ \Delta I_{te_i} \leq \Delta I}} [((\Delta I + t) - \Delta I_{te_i}) G_{te_i} + c_{te_i}]$$

$$\begin{aligned}
&= \sum_{\substack{\forall te_i \in Te \\ \Delta I_{te_i} \leq \Delta I}} [(\Delta I - (\Delta I_{te_i} - t))G_{te_i} + c_{te_i}] \\
&= \sum_{\substack{\forall te_i \in Te' \\ \Delta I_{te_i} \leq \Delta I}} [(\Delta I - \Delta I_{te_i})G_{te_i} + c_{te_i}] \\
&= \Upsilon(\Delta I, Te')
\end{aligned}$$

□

8.3.4. *Scaling with a cost value* $(\cdot)$. Another operation on test lists is to scale it by a cost value. This is, for example, necessary for the integration of the worst-case execution times into the analysis. If the test lists uses the number of events as unit, it is necessary to scale it for analysis with the worst-case execution time.

DEFINITION 8.17. Let Te' be the test list Te scaled by the cost value c. $(Te' = cTe)$ Than for each interval ΔI the corresponding event bound functions have the relationship

$$\Upsilon(\Delta I, Te') = c\Upsilon(\Delta I, Te)$$

LEMMA 8.18. $\Upsilon(\Delta I, Te') = c\Upsilon(\Delta I, Te)$ *if* Te' *contains and only contains for each test lists element* θ *of the child set of* Te *an element* $te' \in Te'$ *having the following relations to* Te*:* $\Delta I_{te'} = \Delta I_{Te}$, $c_{te'} = cc_{Te}$, $G_{te'} = cG_{te}$

All parts of the test list elements related to the amount of events are scaled by the variable c.

PROOF. The proof of this lemma is quite similar to the proofs above.

$$\begin{aligned}
\Upsilon(\Delta I, Te)c &= \sum_{\substack{\forall te_i \in Te \\ \Delta I_{te_i} \leq \Delta I}} [(\Delta I - \Delta I_{te_i})G_{te_i} + c_{te_i}]c \\
&= \sum_{\substack{\forall te_i \in Te \\ \Delta I_{te_i} \leq \Delta I}} [(\Delta I - \Delta I_{te_i}) \cdot (G_{te_i}c) + (c_{te_i}c)] \\
&= \sum_{\substack{\forall te_i \in Te' \\ \Delta I_{te_i} \leq \Delta I}} [(\Delta I - \Delta I_{te_i})G_{te_i} + c_{te_i}] \\
&= \Upsilon(\Delta I, Te')
\end{aligned}$$

□

8.3.5. *Utilization.* An important value for the feasibility analysis is always the utilization of an task set.

LEMMA 8.19. *The utilization* U_{Te} *of a test-list is given by:*

$$U_{Te} = \sum_{te \in Te} G_{te}$$

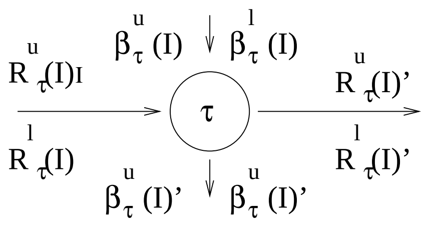

FIGURE 13. Real-Time Calculus of single task

Note that only the total gradient determines the utilization. All test list elements only influence a certain interval being small compared to the infinite interval.

PROOF. The proof is based on the fact that in the long run the contribution of the last period gets infinite small compared to the rest.

$$\begin{aligned} U_{Te} &= \lim_{\Delta I \to \infty} \left(\frac{\sum_{\substack{\forall te \in Te \\ \Delta I_{te} \leq \Delta I}} [(\Delta I - \Delta I_{te}) G_{te} + c_{te}]}{\Delta I} \right) \\ &= \sum_{\forall te \in Te} \lim_{\Delta I \to \infty} \left(\frac{[(\Delta I - \Delta I_{te}) G_{te} + c_{te}]}{\Delta I} \right) \\ &= \sum_{\forall te \in Te} \lim_{\Delta I \to \infty} \left(G_{te} - \frac{\Delta I_{te}}{\Delta I} + \frac{c_{te}}{\Delta I} \right) \\ &= \sum_{\forall te \in Te} G_{te} \end{aligned}$$

□

8.3.6. *Operations of the real-time calculus.* A scheduling network is a system consisting of several chains of tasks and a set of resources. Each task τ of the task chain is mapped to one resource ρ. Tasks mapped on the same resource are scheduled with fixed priority scheduling. Different tasks of a chain can be mapped on different resources. In the figure 1 the tasks τ_1, τ_4, τ_6 forms a task chain and the tasks τ_1, τ_4, τ_7 forms another task chain. Each task τ is triggered by an upper and lower arrival curve $R^u_\tau(\Delta I)$ and $R^l_\tau(\Delta I)$ and the available computational effort for this task is described by an upper and lower service curve $\beta^u_\tau(\Delta I)$ and $\beta^l_\tau(\Delta I)$.

Figure 13 gives a closer look at one single task τ and their curves. For each task we have an incoming (upper and lower) arrival curve $R^u_\tau(\Delta I)$ and $R^l_\tau(\Delta I)$ modelling the workload for τ. It includes and is based on the arrival times of those events generating workload for τ. We also have an (upper and lower) service curve $\beta^u_\tau(\Delta I)$ and $\beta^l_\tau(\Delta I)$ modeling the amount of workload that can be handled by the task.

The analysis of a task generates outgoing (upper and lower) arrival ($R^u_\tau(\Delta I)'$ and $R^l_\tau(\Delta I)'$) and service curves ($\beta^u_\tau(\Delta I)'$ and $\beta^l_\tau(\Delta I)'$). The outgoing arrival curve is a modification of the incoming arrival curves and is also the incoming arrival curve of the following

Algorithm 3 inf-split

```
Algorithm inf-split // $\inf_{0\leq\Delta I'\leq\Delta I}(R(\Delta I') + \beta(\Delta I - \Delta I'))$
testlist $R, \beta$
testlist $S = \emptyset$
FOR all $te \in R$ and all $te \in \beta$
   $S := min(S, subAddOneList(R, \Delta I_{te}, \beta))$
   $S := min(S, subAddOneList(\beta, \Delta I_{te}, R))$
END FOR
RETURN $S$
Algorithm subAddOneList $Te$, $\Delta I$, $Te'$
testlist $tmp := \emptyset$
$tmp := Te + \Delta I$
$c := \sum_{\substack{\forall te' \in Te' \\ \Delta I_{te'} < \Delta I_{te}}} [c_{te'} + (\Delta I - \Delta I_{te'})G_{te'}]$
$tmp := tmp \cup \{(\Delta I, c, 0)\}$
RETURN $tmp$;
```

task in the chain. The outgoing service curve is the incoming service curve reduced by the workload handled by the task. It is the incoming service curve for the task with the next lower priority on the same resource.

The real-time calculus provides the equations to describe the relationships between the incoming and outgoing curves [**7**]. For the calculation the functions sup and inf are needed provinding upper and lower bounds. Their value can be reachable, but need not to be.

The outgoing service curves, giving the available capacity for the task with the next lower priority on the same processor, can be calculated by:

$$\begin{aligned}\beta^l_\tau(\Delta I)' &= \min(\sup_{0\leq\Delta I'\leq\Delta I}\{\beta^l_\tau(\Delta I') - R^u_\tau(\Delta I)\}, 0)\\ \beta^u_\tau(\Delta I)' &= \sup_{o\leq\Delta I'\leq\Delta I}\{\beta^u_\tau(\Delta I') - R^l_\tau(\Delta I')\}\end{aligned}$$

For our model we have already provided equation for calculating the remaining capacity based on the exceeding cost function. They can be used in the real-time calculus:

$$\begin{aligned}\mathscr{C}^l(\Delta I, \tau_i) &= \mathscr{C}^l(\Delta I, \tau_{i-1}) - \Phi^u(\Delta I, \tau_{i-1}) + \Upsilon^u(\Delta I, \tau_{i-1})\\ \mathscr{C}^u(\Delta I, \tau_i) &= \mathscr{C}^u(\Delta I, \tau_{i-1}) - \Phi^l(\Delta I, \tau_{i-1}) + \Upsilon^l(\Delta I, \tau_{i-1})\end{aligned}$$

We can set $\beta^x_\tau(\Delta I) = \mathscr{C}^x(\Delta I, \tau)$ and $R^x_\tau(\Delta I) = \Phi^x(\Delta I, \tau)$.

The outgoing lower arrival curve is given by:

$$R^l_\tau(\Delta I)' = \inf_{0\leq\Delta I'\leq\Delta I}\{R^l_\tau(\Delta I') + \beta^l_\tau(\Delta I - \Delta I')\}$$

In algorithm 3 gives a concrete implementation for this operation based on test lists. The idea is to keep either $\Delta I'$ or $\Delta I - \Delta I'$ fixed, calculate the fixed value for either $R^l_\tau(\Delta I')$ or $\beta^l_\tau(\Delta I - \Delta I')$ and complete this value to every possible interval ΔI with the test list of the other function. The resulting test list for this completion operation can be calculated and the overall resulting test list is given by the infimum over the test lists of all possible fixed values for $\Delta I'$ and $\Delta I - \Delta I'$. Necessary for them is a algorithm to find the step-wise minimum or infimum of two test lists. The implementation of such an algorithm is very straight forward and therefore skipped here. It is simply processing both list in the ascending order of their test-list elements and to register always the dominating element (the element leading to the lower overall cost value). In case of different gradients of the corresponding elements the domination can change at an interval $\Delta I'$ between two intervals. The calculation of $\Delta I'$ is simply the calculation of the crossing point of two lines. The outgoing upper arrival curve is given by:

$$R^u_\tau(\Delta I)' = \min(\inf_{0 \le \Delta I' \le \Delta I}\{\sup_{0 \le v \le \infty}[R^u_\tau(\Delta I' + v) - \beta^l_\tau(v)] + \beta^u_\tau(\Delta I - \Delta I')\}, \beta^u_\tau(\Delta I))$$

We define the sup-add operation handling the inner part of the equation:

$$\sup_{0 \le v \le \infty}[R^u_\tau(\Delta I' + v) - \beta^l_\tau(v)]$$

Its implementation for test lists is given in algorithm 4.

The idea is similar as for the inf-split operation, we also hold an interval and build a test list for all possible completions. But we uses v here always as fixed value. The implemetation of the sup or maximums operation is similar to the inf or minimum operation.

9. Conclusion

In this work we presented a new advanced event model especially suitable for the modeling of distributed systems. Such a system consists of several tasks bound on different processing elements, triggering each other. To divide the problem of real-time analysis of the whole system to a problem of real-time analysis of the single tasks, a model efficiently describing the densities of the events triggering the tasks (incoming events) and those events generated by the tasks to trigger other tasks (outgoing events) was required. Additionally, a model for the capacity of the processing elements available for the tasks was necessary. This is especially complicated in the cases with a higher priority tasks already having used up a part of the capacity. In this paper we proposed an unified model as a whole. Additionally this model is capable to introduce approximations into the description of the event densities which guarantees a fast evaluation as well as an upper bound on the approximation error.

The new model integrates the efficient modeling of periodic and aperiodic events, burst of events in various kinds, approximated event streams, the initial capacity and the remaining capacites of processors in one single model. It can be seen as an explicit description for the arrival, service and capacity curves of the real-time calculus having the necessary modeling capabilies for them. We have presented the real-time analysis for this model for both, systems with dynamic or static priorities.

Algorithm 4 sup-add

```
Algorithm sup-add // $\sup_{0\leq v<\infty}(R(I+v)-\beta(v))$
TestList $R,\beta$
testList $S=\emptyset$
FOR all $te\in R$ and all $te\in\beta$
// $\Delta I_{te}=v$
    $diff=$
$\sum_{\substack{te'\in R\\ \Delta I_{te'}\leq\Delta I_{te}}}[c_{te'}+(\Delta I_{te}-\Delta I_{te'})G_{te'}]-\sum_{\substack{te'\in\beta\\ \Delta I_{te'}<\Delta I_{te}}}[c_{te'}+(\Delta I_{te}-\Delta I_{te'})G_{te'}]$
    $//diff=R(\Delta I_{te})-\lim_{\substack{\Delta I'\to\Delta I_{te}\\ \Delta I'<\Delta I_{te}}}\beta(\Delta I')$

    // Hold the point of $\beta$
    $G_R=\sum_{\substack{\forall te'\in R\\ \Delta I_{te'}\leq\Delta I_{te}}}G_{te'}$
    $Te_{tmp}:=\{te'|te'\in R\wedge\Delta I_{te'}>\Delta I_{te}\}$
    $Te_{tmp}:=Te_{tmp}-\Delta I_{te}$
    $Te_{tmp}:=Te_{tmp}+\{(0,diff,G_R)\}$
    $S:=sup(S,Te_{tmp})$

    // Hold the point of $R$
    // Needed inverse $\beta$
    $G_\beta=\sum_{\substack{\forall te'\in\beta\\ \Delta I_{te'}<\Delta I_{te}}}G_{te'}$
    $Te_{tmp}:=\{te'|te'\in R\wedge\Delta I_{te'}<\Delta I_{te}\}$
    $Te_{tmp2}:=\{(0,diff,G_\beta)\}$
    FOR each $te_i\in Te_{tmp}$
        $Te_{tmp2}:=Te_{tmp2}\cup\{(\Delta I_{te}-\Delta I_{te_i},c_{te'_i},G_{te_{i-1}})\}$
    END FOR
    $S:=sup(S,Te_{tmp2})$
END FOR
RETURN $S$
```

We have also propose an efficient approximative model to describe stimulations of tasks in a distributed real-time system. It was shown that this model integrates many other models describing stimulation in a system and delivers due to a chooseable degree of approximation a general description of stimulation. In the next step we described how a efficient real-time analysis for the model can be done for static and dynamic priorities. In order to show the relevant impact of our model and methods we use the real-time calculus. We give an efficient way to integrate the real-time calculus in our model. Thereby we show how the abstract described functions can be implemented in a concrete manner. In future we will use this model for further applications in order to improve methods for the real-time analysis.

ACKNOWLEDGEMENT. This work was funded by the Deutsche Forschungsgemeinschaft (DFG) under the grant *SL 47/3-1*.

ΔI	interval
T	period
a	offset
l	limitation
k	number of test intervals
Θ	event stream
$\theta = (T, a)$	event element
$\hat{\Theta}$	hierachical event stream
$\hat{\theta} = (T, a, l, G, \hat{\Theta}_{\hat{\theta}})$	hierachical event stream element
s	separation point
$\Upsilon(\Delta I, \hat{\Theta})$	event bound function
$\Psi(\Delta I, \hat{\Theta})$	demand bound function
$\mathscr{I}(\Delta I, \hat{\Theta})$	interval bound function
Te	test list
$te = (\Delta I, c, G)$	test-list element
c	costs
$\mathscr{B}$	busy period

TABLE 1. List of symbols

Bibliography

[1] K. Albers, F. Bodmann, and F. Slomka. Hierachical event streams and event dependency graphs. In *Proceedings of the 18th Euromicro Conference on Real-Time Systems (ECRTS'06)*, pages 97–106, 2006.

[2] K. Albers, F. Bodmann, and F. Slomka. Run-time efficient feasibility analysis of uni-processor systems with static priorities. In *Poceedings of the International Embedded Systems Symposium (IESS 2007)*, 2007.

[3] K. Albers and F. Slomka. An event stream driven approximation for the analysis of real-time systems. In *IEEE Proceedings of the 16th Euromicro Conference on Real-Time Systems*, pages 187–195, Catania, 2004.

[4] K. Albers and F. Slomka. Efficient feasibility analysis for real-time systems with edf-scheduling. In *Proceedings of the Design Automation and Test Conference in Europa (DATE'05)*, pages 492–497, 2005.

[5] S.K. Baruah. Dynamic- and static-priority scheduling of recurring real-time tasks. *International Journal of Real-Time Systems*, 24:98–128, 2003.

[6] S.K. Baruah, A. Mok, and L.E. Rosier. Preemptive scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the Real-Time Systems Symposium*, pages 182–190, 1990.

[7] S. Chakraborty, S. Künzli, and L. Thiele. Performance evaluation of network processor architectures: Combining simulation with analytical estimations. *Computer Networks*, 41(5):641–665, 2003.

[8] R.L. Cruz. A calculus for network delay. In *IEEE Transactions on Information Theory*, volume 37, pages 114–141, 1991.

[9] K. Gresser. *Echtzeitnachweis ereignisgesteuerter Realzeitsysteme*. Dissertation, Düsseldorf, 1993.

[10] K. Gresser. An event model for deadline verification of hard real-time systems. In *Proceedings of the 5th Euromicro Workshop on Real-Time Systems*, 1993.

[11] S. Künzli. *Efficient Design Space Exploration for Embedded Systems*. PhD thesis, ETH Zürich No. 16589, 2006.

[12] A.K. Parekh and R.G.Gallager. A generalized processor sharing approach to flow control in integrated service networks. In *IEEE/ACM Transactions on Networking*, volume 1, pages 344–357, 1993.

[13] S. Perathoner, E. Wandler, L. Thiele, A. Hamann, S. Schliecker, R. Henia, R. Racu, R. Ernst, and M. González Harbour. Influence of different system abstractions on the performance analysis of distributed real-time systems. In *EMSOFT 2007*, pages 193–202. IEEE Computer Society Press, 2007.

[14] K. Richter. *Compositional Scheduling Analysis Using Standart Event Models*. Dissertation, TU Braunschweig, 2005.

[15] K. Richter and R. Ernst. Event model interfaces for heterogeneous system analysis. In *Proceedings of the Design Automation and Test Conference in Europe (DATE'02)*, 2002.

[16] L. Thiele, S. Chakraborty, M. Gries, and S. Künzli. Design space exploration for the network processor architectures. In *1st Workshop on Network Processors at the 8th International Symposium for High Performance Computer Architectures*, 2002.