Event Stream Calculus for Schedulability Analysis

Karsten Albers[1] and Frank Slomka[2]

[1] INCHRON GmbH, August-Bebel-Strasse 88, 14482 Potsdam, Germany
karsten.albers@inchron.com
[2] Department of Embedded Systems/Real-Time Systems, Ulm University, 89069 Ulm, Germany
frank.slomka@uni-ulm.de

Abstract. In the paper we will show the integration of the real-time calculus with event driven real-time analysis like the periodic or the sporadic task model. For the event-driven real-time analysis, flexible approximative analysis approaches where 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 chooseable allowing a more accurate description.

1 Motivation

The module-based design processes makes 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. Modules have only designated interfaces and connections to other modules of their set. The purpose of modularization is to split the challenging job of designing the whole system into multiple smaller jobs. Another purpose is to allow the reuse of modules in different designs or use IP components of third-party vendors.

Each module-based design concept needs a well defined interface-concept for connecting the different modules. For developing real-time systems a concept for analysing the system which can handle the real-time aspects of the different modules separately and allows to propagate the results through the system is required. One aspect of this concept 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 τ 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 a modularized system. Necessary therefore is a powerful and sufficient event model for describing the different time interfaces for the different aspects.

2 Related work

The most advanced approach for the real-time analysis of such a modular network is the real-time calculus by [1] and [2]. It is based on the network calculus approach, especially on the concept of arrival and service curves defined by [3] and [4].

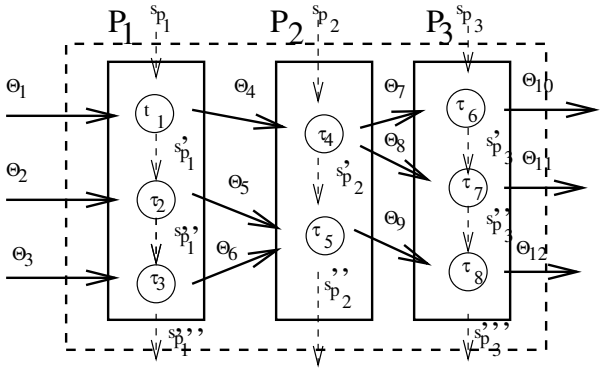

Fig. 1. Network processor example

The event pattern Θ is modeled by an arrival curve $R_f(t)$ which denotes the number of events arriving within a time interval of length I. $R_f^u(t)$ denotes the upper bound and $R_f^l(t)$ the lower bound for this curve. These functions are sub-additive and deliver for every t the maximum respective the minimum amount of events which can occure in any interval of length t. The service curves $\beta_r^u(t)$ and $\beta_r^l(t)$ model the upper and lower bound of the computational requirements which can be handled by the ressource during a time interval of length t. The real-time caclulus 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 finite description for the curves is needed. The complexity of the relationship equations depends directly on the complexity of this description. In [5] and [1] 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 an initial bursts and one the long time rate. As outlined in [6] this approach is too simplified to be suitable for complex systems. It only has a fixed degree of exactness. No suitable descriptions for the function are known so far.

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

SymTA/S [7],[8] 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 uses different event models for analysis. Unfortunately, the event models for which interfaces are provided are quite simple. In [7] an event model covering all this models was described. The problem of these models is that multiple bursts or bursts with different minimum separation times cannot be handled.

The event stream model proposed by [9] with its extension, the hierarchical event stream model proposed by [10] 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.

3 Contribution

In the paper we will give the following contributions. We will 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. Its approximation follows the same scheme like the existing approximation for event models as proposed in [11]. Therefore it is possible to transfer previously existing event models, like the periodic or the sporadic task model, the event stream model, the sporadically task model, the model of SymTA/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 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 first methodology to implement all operations needed by the real-time calculus. This is more accurate than the methodology used in the original literature.

4 Model

In the following we will give a new approximative model for the curves of the real-time calculus allowing a less pessimistic modeling of the curves. It guarantees the approximation error. In [11] 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.

We model each curve of the real-time calculus by a test list $Te = \{te\}$ consisting of a set of test-list elements $te = (I, c, G)$ each modeling one segment of the curve. I 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 $I' < I$.

For example, four events with a distance of 10 to each other and with an execution time of 2 can be modeled by the 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 useful 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 modeled 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 function has a gradient of $\frac{1}{2}$ for the first 100 time units and after them a resuling gradient of 1 for the remaining time.

4.1 Approximation

General event models generate an infinite set of events and would therefore require an infinite number of test-list elements. In the periodic task model for example each task $\tau = (T, cw, d)$ represents an infinite number of jobs sharing the same worst-case execution time cw 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 approximate 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 1 *([11]) Let Γ be any taskset bound on any resource ρ. Let ρ_l be the resource with the minimum capacity on which Γ is feasible. An approximation with approximation error ε is a test algorithm which*

1. returns "non-feasible" in those cases in which Γ on ρ is non-feasible
2. returns "feasible" in all those feasible cases in which $\mathscr{C}(\rho) \geq \frac{1}{1-\varepsilon}\mathscr{C}(\rho_l)$
3. can return either "feasible" or "non-feasible" 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 $\tau = (T, cw, d)$ and a degree of exactness of n can be transferred into a test list Te with the elements

$$Te = \{(0, cw_\tau, 0), (T_\tau, cw_\tau, 0), (2T_\tau, cw_\tau, 0), ..., (nT_\tau, cw_\tau, \frac{cw_\tau}{T_\tau})\}$$

with deadline d_τ. We can transfer this test list further in a test representing the demand bound function $\Psi(I,\tau)$ for τ by shifting it by the deadline $(Te' = \{(d_\tau, cw_\tau, 0), (T_\tau + d_\tau, cw_\tau, 0), ..., (nT_\tau + d_\tau, cw_\tau, \frac{cw_\tau}{T_\tau})\})$.

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 $Te = \{(2,0,1), (100,0,-1), (102,0,1), (200,0,-1), (202,0,1), (300,0,-1), (302,0,\frac{98}{100})\}$.

4.2 Event bound function

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

Definition 2 *An event bound function $\Upsilon(I)$ gives the amout of events which can occure at most in any interval of length I.*

The calculation can be done as follows:

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

5 Real-Time Analysis

In the following we will show how simple an efficient schedulability analysis can be realized with the introduced model.

Algorithm 1 Feasibility Analysis

```
Algorithm Superposition
Given: testList Te (sorted with rising I)
r = 0; G = 0; I_old = 0;
FOR ALL (te ∈ Te)
    r := r + (a_te − I_old)G
    IF (r < 0) THEN ⇒not feasible
    r := r + cw_te
    IF (r < 0) THEN ⇒not feasible
    I_old := I_te; G := G + G_te
END WHILE
IF (G < 0) THEN ⇒not feasible
ELSE  ⇒feasible
```

5.1 EDF

Schedulability analysis for EDF can be done using the processor demand criteria which was introduced by [12], [13].

Definition 3 *([12]) The demand bound function $\Psi(I,\Gamma)$ gives the cummulated execution requirement of those jobs having release time and deadline within I.*

Lemma 1. A task set scheduled with EDF keeps all deadlines if for every intervals $I > 0$ the demand bound function $\Psi(I,\Gamma)$ does not exceed the available capacity $\mathscr{C}(I,\rho)$ for I:$\Psi(I,\Gamma) \leq \mathscr{C}(I,\rho)$

This can be rewritten as:$\mathscr{C}(I,\rho) - \Psi(I,\Gamma) \geq 0$
Proof: See [12] and [11] ■

Both, the demand bound and the service function can be described by test lists as we have already seen. $\mathscr{C}(I,\Gamma) - \Psi(I,\Gamma)$ 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: $\Psi(I,\Gamma) = \sum_{\forall \tau \in \Gamma} \sum_{\forall te \in Te_\tau} \Psi(I,Te)$

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

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

So the resulting analysis for EDF reads:

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

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 define in section 6.

In algorithm 1 we give the short implementation to prove the condition $\Upsilon(I,Te) \geq 0$ for all $I \geq 0$ 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 $I = 0$. 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.

5.2 Analysis for static priorities

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

Definition 4 *([12]) The request bound functions $\Phi(I)$ contains the amount of execution time requested by those events having occured within I.*

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

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

Lemma 2. (similar to [3]) The worst-case response time of a task is given by:

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

Schedulability for a job of a task τ is given if $r_\tau \leq d_\tau$.
Proof: See [12]. ■

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

$\mathscr{C}(I,\tau)$ denotes the capacity available for task τ within I. For the task with the highest priority this is the capacity of the resource $\mathscr{C}(I,\rho)$. For all other tasks 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 seperately. The problem is that an amount of capacity reached for some intervals I is also available for each larger interval I' even if between I and I' a large amount of computation request occures, so that $\Phi(I',\tau) - \Phi(I,\tau) > \mathscr{C}(I',\tau) - \mathscr{C}(I,\tau)$. No part of this requested computation time can be processed within I 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 5 *([14]) Exceeding costs $\Upsilon(I,\Gamma)$ denotes those part of the costs requested within the interval I by the taskset Γ which cannot be processed within I with either scheduling due to the late request times.*

See figure 2 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 be processed within $I = 20$ even if the job fully gets the remaining processor time. The exceeding costs get 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 [14].

Lemma 3. A task set Γ is feasible if for each task $\tau \in \Gamma$ and each $I > 0$:

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

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

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

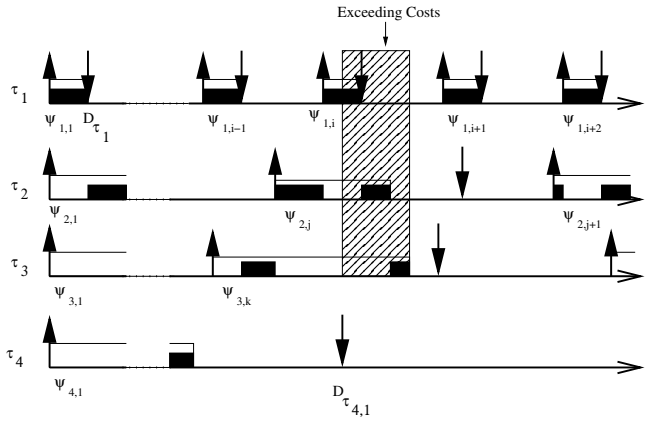

Fig. 2. Exceeding costs

The calculation of the remaining capacity can be therefore done by

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

Proof: See [14]. ■

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 hierarchical schedulability analysis.

Figure 3 visualizes its calculation. The exceeding costs function starts equally to the difference of the request bound function and the available capacity function ($\Phi(I,\tau) - \mathscr{C}(I,\tau)$). 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. Then the exceeding costs function remains zero until the difference function starts rising again, e.g. new request arrives. Then the exceeding costs function will also rise and run further in parallel to the difference function but with a higher value.

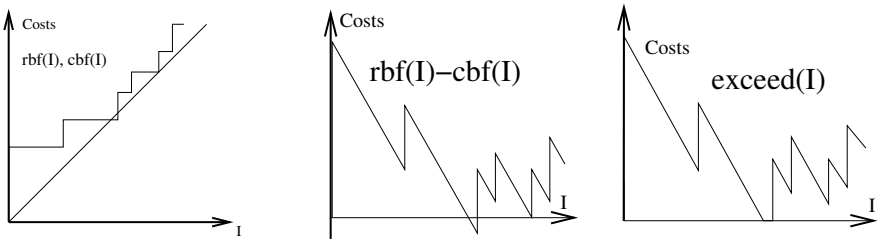

Fig. 3. Calculation of the exceeding costs functions

5.3 Practical issues

Blocking time, scheduling overhead and the priority inheritance protocol can easily be integrated in the above equations. A blocking time b can be integrated by either adding b to the equations or by integrating the test-list element $te = (0, b, 0)$.

6 Operations and Basic Functions

In the following we will introduce some operations on test-lists and their implementation.

6.1 Adding/Subtracting $(+, -)$

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 6 *(+ operation) Let Te_A, Te_B, Te_C be test lists. If Te_C is the sum of Te_A and Te_B ($Te_C = Te_A + Te_B$) then for each interval I the equation $\Upsilon(I, Te_C) = \Upsilon(I, Te_A) + \Upsilon(I, Te_B)$ is true.*

Lemma 4. *(+ operation)* The sum $Te_C = Te_A + Te_B$ can be calculated by the union of the event stream elements of Te_A, Te_B: $Te_{new} = Te_A \cup Te_B$

Proof:

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

■

The resulting test list can be simplified by eliminating test list elements with equal intervals.

Definition 7 *(− operation) Let $Te' = -Te$. The negation of Te is defined by the negation of its corresponding event bound function $\Upsilon(I, -Te) = -\Upsilon(I, Te)$.*

Lemma 5. (− 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 $I_{te'} = I_{te}$, $cw_{te'} = -cw_{te}$ and $G_{\hat{\theta}'} = -G_{\hat{\theta}}$.

Proof: It is obvious that the negation of a test list can be done by the negation of each relevant parameter. ■

We can write $Te_C = Te_A + (-Te_B)$.

6.2 Shift Operation $(\uparrow, \downarrow)$

The shift operation can be realized by adding or subtracting the shift-value from each interval of all test list elements.

Definition 8 *($\uparrow$ shift-operation) Let Te be a test list that is shifted right by the value t resulting in the test list $Te' = Te \uparrow t$. The event bound functions have the following relationship:*

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

Lemma 6. *$\Upsilon(I, Te) \uparrow t = \Upsilon(I, Te')$ if Te' contains and only contains for each element te of Te an element $te' \in Te'$ having the following relations to te: $I_{te'} = I_{te} + t$, $c_{te'} = c_{te}$, $G_{te'} = G_{te}$*

Proof: It is a simple shift operation on functions. ■

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

6.3 Scaling with a cost value

Another operation on test lists 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.

Definition 9 *Let Te' be the test list Te scaled by the cost value cw ($Te' = Te \times cw$). Then for each interval I:*

$$\Upsilon(I, Te') = \Upsilon(I, Te)cw$$

Lemma 7. $\Upsilon(I, Te') = \Upsilon(I, Te) \times cw$ 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: $I_{te'} = I_{Te}$, $c_{te'} = c_{Te} \times cw$,$G_{te'} = G_{te} \times cw$

Proof: All parts of the test list elements related to the amount of events are scaled by the variable *cw*. ■

6.4 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 form a task chain and the tasks τ_1, τ_4, τ_7 form another task chain. Each task τ is triggered by an upper and lower arrival curve $R^u_\tau(I)$ and $R^l_\tau(I)$ and the available computational effort for this task is described by an upper and lower service curve $\beta^u_\tau(I)$ and $\beta^l_\tau(I)$.

Figure 4 gives a closer look at one single task τ and their curves.

For each task we have an incoming (upper and lower) arrival curve $R^u_\tau(I)$ and $R^l_\tau(I)$ modeling 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 $\beta^u_\tau(I)$ and $\beta^l_\tau(I)$ modeling the amount of workload that can be handled by the task.

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

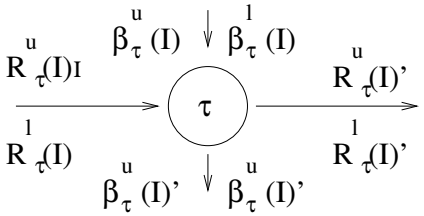

Fig. 4. Real-Time Calculus of single task

following 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 [1]. For the calculation the functions sup and inf are needed providing upper and lower bounds. Their value can be reachable, but does not need 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:

$$\beta_\tau^l(I)' = \min(\sup_{0\le I'\le I}\{\beta_\tau^l(I') - R_\tau^u(I)\}, 0)$$
$$\beta_\tau^u(I)' = \sup_{o\le I'\le I}\{\beta_\tau^u(I') - R_\tau^l(I')\}$$

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

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

We can set $\beta_\tau^x(I) = \mathscr{C}^x(I,\tau)$ and $R_\tau^x(I) = \Phi^x(I,\tau)$.
The outgoing lower arrival curve is given by:

$$R_\tau^l(I)' = \inf_{0\le I'\le I}\{R_\tau^l(I') + \beta_\tau^l(I-I')\}$$

Algorithm 2 gives a concrete implementation for this operation based on test lists. The idea is to keep either I' or $I-I'$ fixed, calculate the fixed value for either $R_\tau^l(I')$ or $\beta_\tau^l(I-I')$ and complete this value to every possible interval I 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 I' and $I-I'$. 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 lists 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 I' between two intervals. The calculation of I' is simply the calculation of the crossing point of two lines. The outgoing upper arrival curve is given by:

$$R_\tau^u(I)' = \min(\inf_{0\le I'\le I}\{\sup_{0\le v\le\infty}[R_\tau^u(I'+v) - \beta_\tau^l(v)] + \beta_\tau^u(I-I')\}, \beta_\tau^u(I))$$

Algorithm 2 inf-split

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

Algorithm 3 sup-add

Algorithm sup-add // $\sup_{0\le v<\infty}(R(I+v)-\beta(v))$
TestList R,β
testList $S=\emptyset$
for all $te\in R$ and all $te\in\beta$
  // $I_{te}=v$
  $diff=\sum_{\substack{te'\in R\\ I_{te'}\le I_{te}}}[cw_{te'}+(I_{te}-I_{te'})G_{te'}]-\sum_{\substack{te'\in\beta\\ I_{te'}<I_{te}}}[cw_{te'}+(I_{te}-I_{te'})G_{te'}]$
  //$diff=R(I_{te})-\lim_{\substack{I'\to I_{te}\\ I'<I_{te}}}\beta(I')$
  // Hold the point of β
  $G_R=\sum_{\substack{\forall te'\in R\\ I_{te'}\le I_{te}}}G_{te'}$
  $Te_{tmp}:=\{te'|te'\in R\wedge I_{te'}>I_{te}\}$
  $Te_{tmp}:=Te_{tmp}-I_{te}$
  $Te_{tmp}:=Te_{tmp}+\{(0,diff,G_R)\}$
  $S:=sup(S,Te_{tmp})$

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

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

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

Its implementation for test lists is given in algorithm 3. 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 use v here always as a fixed value. The implemetation of the sup or maximum operation is similar to the inf or minimum operation.

7 Conclusion

In this paper we 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 an 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 abstractly 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.

References

1. Chakraborty, S., Künzli, S., Thiele, L.: Performance evaluation of network processor architectures: Combining simulation with analytical estimations. Computer Networks **41**(5) (2003) 641–665
2. Thiele, L., Chakraborty, S., Gries, M., Künzli, S.: 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)
3. Cruz, R.: A calculus for network delay. In: IEEE Transactions on Information Theory. Volume 37. (1991) 114–141
4. Parekh, A., R.G.Gallager: A generalized processor sharing approach to flow control in integrated service networks. In: IEEE/ACM Transactions on Networking. Number 3 (1993) 344–357
5. Künzli, S.: Efficient Design Space Exploration for Embedded Systems. PhD thesis, ETH Zürich No. 16589 (2006)
6. Albers, K., Slomka, F.: Efficient feasibility analysis for real-time systems with edf-scheduling. In: Proceedings of the Design Automation and Test Conference in Europa (DATE'05). (2005) 492–497
7. Richter, K.: Compositional Scheduling Analysis Using Standart Event Models. Dissertation, TU Braunschweig (2005)
8. Richter, K., Ernst, R.: Event model interfaces for heterogeneous system analysis. In: Proceedings of the Design Automation and Test Conference in Europe (DATE'02). (2002)
9. Gresser, K.: An event model for deadline verification of hard real-time systems. In: Proceedings of the 5th Euromicro Workshop on Real-Time Systems. (1993)
10. Albers, K., Bodmann, F., Slomka, F.: Hierachical event streams and event dependency graphs. In: Proceedings of the 18th Euromicro Conference on Real-Time Systems (ECRTS'06). (2006) 97–106
11. Albers, K., Slomka, F.: An event stream driven approximation for the analysis of real-time systems. In: IEEE Proceedings of the 16th Euromicro Conference on Real-Time Systems, Catania (2004) 187–195
12. Baruah, S.: Dynamic- and static-priority scheduling of recurring real-time tasks. International Journal of Real-Time Systems **24** (2003) 98–128
13. Baruah, S., Mok, A., Rosier, L.: Preemptive scheduling hard-real-time sporadic tasks on one processor. In: Proceedings of the Real-Time Systems Symposium. (1990) 182–190
14. Albers, K., Bodmann, F., Slomka, F.: Run-time efficient feasibility analysis of uni-processor systems with static priorities. In: Poceedings of the International Embedded Systems Symposium (IESS 2007). (2007)