Internal Report

Dependencies Aware Event-Driven Real-Time Analysis for Distributed Fixed-Priority Systems

Steffen Kollmann, Karsten Albers, and Frank Slomka

Department for Embedded Systems/Real-Time Systems, Ulm University
{forename}.{surname}@uni-ulm.de
http://www.uni-ulm.de/in/esys

Abstract. In this paper we present an approach to calculate the maximum density of events in a distributed hard real-time system having tree-shaped dependencies. Thereby we will present how it is possible to relax the density of events in such a system by including scheduling dependencies. This relaxation has a direct impact of successive tasks and leads to more realistic real-time analysis. In this paper we distinguish between the task model and the model for the stimulation with the result that we can describe a major range of stimulation. In the end we will show how it is possible to make a real-time analysis with the presented approach.

Key words: Distributed System, Fixed Priority, Real-Time Analysis, Event Streams, Tree-Shaped Dependencies, Embedded System

1 Introduction

Many approaches have been developed to analyze hard real-time systems but most of them assume that the tasks in the system are independent. This means, for example, that no influences between tasks caused by scheduling or communication are considered. But these dependencies can lead to a more relaxed stimulation in the system which therefore leads to a higher utilization of the processor and communication elements in the system. So it is necessary to have methods which are able to analyze such complex systems in an appropriate time and are not too pessimistic during the calculation at the same time. First we introduce an architecture serving as motivation and as example for the whole paper. In figure 1 a heterogeneous distributed system is presented which is part of an overall context and consists of two processing elements ($PE1$ and $PE2$), a bus ($BUS1$) and their tasks ($\tau_1, ..., \tau_8$). Note that between τ_1 and τ_6 is a direct connection. We assume that the subsystem has to fulfill time constraints and therefore a real-time analysis is required. So it is necessary to compute the maximum density of events which can occur in the system. The arrival of data in the

system is specified by the environment and the tasks on the processing elements are scheduled by a fixed priority schedule. The edges represent the triggering between the tasks when new data are available. Event streams $(ES1, ..., ES11)$ describing the maximum density of events are assigned to these edges. Since only the stimulation for τ_1, τ_2 and τ_3 are available from the environment, the aim is to calculate the event streams of the remaining system.

Our contribution in this paper is to present an approach to determine the maximum density of events occurring in heterogeneous distributed systems like in the example. Thereby dependencies caused by the scheduling on a resource are taken into account which leads to more relaxed event streams. One special predicate of the analysis is that tasks stimulated by the same events present a special case during the analysis. Such a case is presented via τ_2 and τ_3 in figure 1. It is obvious that if the outgoing event stream of a task is more relaxed by the implication of dependencies that successive tasks have a more relaxed input event stream. In the figure 1, if τ_3 produces a more relaxed output τ_5 will get a more relaxed input. In turn the output of τ_5 is more relaxed and therefore the input of τ_7 as well. The propagation of the relaxed event streams ends in a pattern of tree-shaped dependencies.

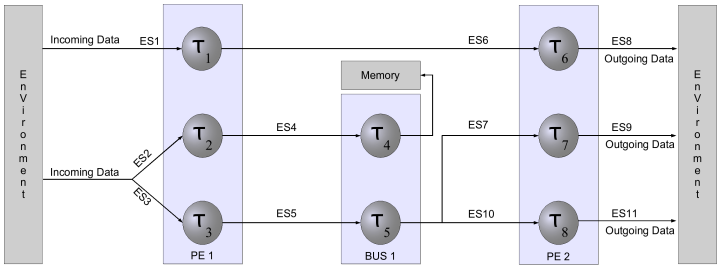

Fig. 1. Example of a distributed system

The rest of the paper is organized as follows. Chapter 2 gives an overview about related work in this area. In chapter 3 we will introduce our model based on the event stream model by K. Gresser [1]. On basis of the model we will show how to calculate the event streams in a heterogeneous distributed system. Subsequently, we give an example and how it is possible to achieve a real-time analysis with this result. The conclusion follows at the end.

2 Related Work

In order to conduct a real-time analyses it is necessary to choose an appropriate model. Most of the work in this area is based on the periodic task model with jitter as it is used in Tindell and Clark [2], for example. Many approaches have been developed based on this model but the simplicity of the model leads to a loss of accuracy. Another disadvantage is that no dependencies have been considered. Hence, in [3] the transaction model has been developed where it is possible to group dependent tasks in transaction groups and describe dependencies with fixed offsets. This approach has been integrated in [4] the Holistic Scheduling Analysis. Redell generalized this approach in [5] and allows more sophisticated

task chains. In his approach it is possible that a task can trigger more than one task what is in our opinion a more realistic case. The idea was used in [6] by Henia et al.. This paper explores time-correlation between tasks via tree-shaped-dependencies. This allows to describe additionally relative dependencies to the last common predecessor of tasks. Henia et al. have been improved this idea in [7]. But dependencies caused by the scheduling on a processor has not been considered the same also applies for all the other papers.

We use the event stream model to describe the stimulation. The advantage of this model is, that we are able to describe any kind of stimulation. Due to the simplicity we use the event streams, although the approach is extendible on hierarchical events streams [8] which are more expressive.

3 Model

In this section we introduce our models. We differentiate between the task model and the model for the stimulation. The reason for this can be observed, for example, in the periodic task model with jitter [2] where we have got on the one hand the task model with the best case and worst execution times, and on the other hand the model for the stimulation with jitter and period. But this is not enough to describe a major range of stimulations. Therefore we use the more generalized event stream model to describe the stimulation.

3.1 Task Model

Definition 1. *Γ is the set of tasks of one resource $\Gamma = \{\tau_1, ..., \tau_n\}$. A task is a 4-tuple with $\tau = (c, b, d, \rho)$. c is the worst case execution time, b is the best case execution time and d is the relative deadline. ρ defines the priority of the task for the scheduling. Let τ_{ij} be the j-th job/execution of task i.*

In our model we assume that a task can only generate an event at the end of its execution to notify other tasks. Furthermore we assume that the tasks are scheduled by fixed-priority schedules. We also consider only systems where the load is lower 100% ($U < 100\%$). The load of 100 % is a special case and is not discussed here.

3.2 Event Streams

Event streams has been first defined in [1]. The purpose was to give a generalized description for every kind of stimuli. The basic idea is to define an event function $E(I)$ which can calculate for every interval I the maximum amount of events which can occur within I. For this purpose only the length of I is relevant. In the following, when speaking of intervals we mean the length of the interval. An interval with a specific begin and end point we will call specific interval. The event function needs a properly described model behind it which makes it easy to extract the information. The idea is to notate for each number of events

the minimum interval which can include this number of events. Therefore we get an interval for one event (which is infinite small and therefore considered to be zero), two events and so on. The result is a sequence of intervals showing a non-decreasing behavior. The reason for this behavior is, that the minimum interval for n events cannot be smaller than the minimum interval for n-1 events since the first interval also includes n-1 events. This sequence of intervals shows a periodic behavior and is called event stream. Each of the single intervals is called event stream element.

Definition 2. *An event stream is a set of event stream elements* $\psi{:}ES{=}\{\psi_1,\psi_2,...,\psi_n\}$ *and each event stream element* $\psi = (p\ a)^T$ *consists of an offset-interval* a *and a period* p*.*

Each event stream element describes a set of elements of the sequence. For the event stream element ψ the element $a + k \cdot p$ is part of the sequence and all the elements with $k \in N$. An event stream models a given sequence if all the elements and only the elements of the sequence can be generated using the event stream elements. Therefore it is possible to calculate for each possible interval the maximum amount of events that can occur within this interval:

Definition 3. *Event Stream Function*

$$E(I){=}\textstyle\sum_{i=1}^{n} E_i(I) \quad ; \quad E_i(I){=}\begin{cases} 0 & I{<}a_i \\ \left\lfloor \frac{I-a_i}{p_i}+1 \right\rfloor & I{\geq}a_i \wedge p_i{<}\infty \\ 1 & I{\geq}a_i \wedge p_i{=}\infty \end{cases}$$

Furthermore the event stream model complies the characteristic $E(I_1{+}I_2){\leq} E(I_1){+}E(I_2)$. This characteristic is called sub-additivity. It means that the maximum number of events of an interval cannot exceed the cumulated maximum number of events of its subintervals.

Events can occur in a greater distance than it is described in the event stream which describes only the minimum distance between a number of events.

An event stream in which all elements have either the same or infinite period is called homogeneous and every event stream can be made homogeneous using the least common multiplier of its periods as new period of all elements and complete its set of event stream elements. With a period of infinite (∞) it is possible to model irregular behavior.

Note that the order of the elements is of no concern for the evaluation. For the purpose of evaluation it is not necessary to find the exact minimum intervals. It is sufficient to find for all intervals a lower bound. This can allow to simplify the event stream (also might mean to accept an overly pessimistic description). A detailed definition of the concept and the mathematical foundation can be found in [9].

3.3 Subset of Event Streams

For our calculations in chapter 4 we need an order relation for the event streams. We introduce the identifier ψ_i denoting the i-th element in the event stream.

$$ES=\left\{ \begin{pmatrix} \infty \\ a_1 \end{pmatrix},...,\begin{pmatrix} \infty \\ a_{m-1} \end{pmatrix},\begin{pmatrix} p_k \\ a_m \end{pmatrix},...,\begin{pmatrix} p_k \\ a_n \end{pmatrix} \right\}: 1 \leq m \leq n \wedge a_i \leq a_j \Leftrightarrow i \leq j \quad (1)$$

The order relation means that all aperiodic events are described first in the stream. All other events have the same period and follow directly after the aperiodic events. This subset is no restriction, because every event stream can be transformed into this pattern. To achieve this we take all periods being not infinit and find the least common multiplier and use it as new period. Second we take all periodic events which does not fulfill the equation above and shift the offset until the equation is fulfilled. The shift operation is performed as follows: $(p\ a)^T = (\infty\ a)^T, (p\ (a+p))^T$. Furthermore it is necessary that the aperiodic part is sorted by its offset as well as the periodic part.

It is useful to define names for certain properties of the event stream. N is the number of tuple in the event stream. N_∞ is the number of tuples with a period of infinity and N_p is the number of tuples with a period unequal infinity.

4 Competing Tasks

In this chapter we present a technique to calculate the event streams in a distributed system. Thereby the assumptions made in chapter 3 are used.

4.1 The Maximum Density of Events

In order to calculate the outgoing event stream of a task we have to determine the worst case. Or in other words, the maximum density of events that a task can produce. The next lemma gives the worst case. We assume that the task is triggered by the maximum density of events which is described by the event stream. The event stream stimulating a task is called *incoming event stream* and the event stream produced by a task is called *outgoing event stream.*

Lemma 1. *A number of outgoing events occur in the maximum density when the first event is delayed as much as possible and all further events occur as early as possible.*

Proof. We assume that two outgoing events exist having a higher density than the events fulfilling the assumption. If the first outgoing event and second outgoing event are closer together than in the assumption, this would mean either the first outgoing event arrives later than allowed by the assumption or the second event arrives earlier than allowed by the assumption. This is a contradiction, because we assume already the maximum or minimum values for both arrival

times. So there must be two other events later in the outgoing event stream having a shorter distance to each other. Assume that two events are occurring closer than in the assumption and the first event is delayed as much as possible and the second arrives as early as possible, this would mean that the corresponding incoming events also have a shorter distance to each other than the first two incoming events. But this is in contradiction to the event stream definition. The proof for another number of events is analog. □

From this lemma we can follow the required information to calculate the outgoing event stream of a task. Since the first execution of the task is maximum delayed and the stimulations of tasks triggered by the same event stream as the task itself are considered as well, the following information are necessary:

1. *The incoming event stream of the task itself and the incoming event streams of the higher priority tasks on the same processing element.*
2. *The properties of the task itself and of the higher priority tasks on the same processing element.*

For Example τ_3 on *PE1* in figure 1 needs $ES1$, $ES2$ and $ES3$ and the properties of τ_1, τ_2 and τ_3. In this case we assume that τ_1 has got a higher priority than τ_2 and that τ_2 has got a higher priority than τ_3.

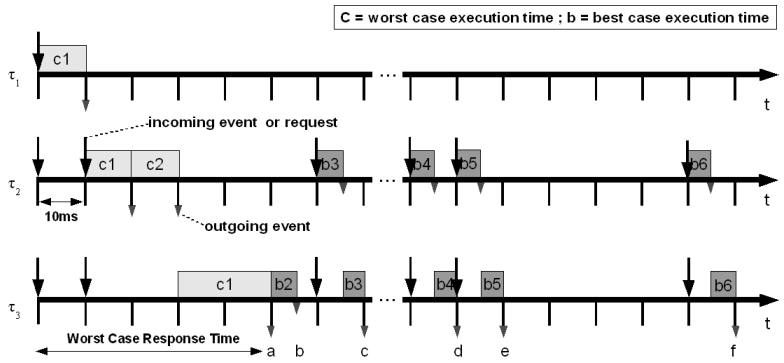

Fig. 2. Scheduling example of τ_1, τ_2 and τ_3

The figure 2 is a cut-out of a gantt-chart showing the described scenario above. The first execution of τ_3 is stretched as much as possible by its worst case response time. This means that the outgoing event is delayed as much as possible.

The remaining executions of τ_3 run only with their best case execution time to ensure that the next events come as early as possible. As mentioned in chapter 1 only one task can be processed on a resource. This means that the executions of τ_2 has to be considered for the analysis of τ_3, because this task is triggered by the same event stream. τ_1 must only be considered during the worst case

response time to ensure that the first event of τ_3 delayed as much as possible. Since the incoming event stream describes only the worst case, events can occur in a greater distance than that one specified by the event stream. In order to ensure that the remaining events of τ_3 occur as early as possible we can assume - according to the event stream model - that the executions of τ_1 have no influence after the first execution of τ_3.

4.2 Calculation of the Maximum Density

In order to calculate the outgoing event stream of a task, we will introduce the necessary equation.

$$ES_{out} = \left\{ \bigcup_{i=1}^{j} \left(\underset{RET(i,\tau,ES_{in})-WCRT_1(\tau)}{\infty} \right), \bigcup_{i=j+1}^{N^P+j} \left(\underset{RET(i,\tau,ES_{in})-WCRT_1(\tau)}{p_i} \right) \right\} \quad (2)$$

Equation 2 defines how the incoming event stream is processed in order to calculate the outgoing event stream. The equation consists of two parts. The first part gives all event stream elements having an aperiodic behavior and the second part all event stream elements having a periodic behaviour. The offset for the i-th element of both parts is the distance between that point in time when the i-th event of the incoming event event stream has been processed (**R**equest **E**nd **T**ime; RET) and that point in time when the first event of the incoming event stream has been processed (**W**orst **C**ase **R**esponse **T**ime; WCRT). The following incoming events have no influence on the offset of the i-th event, because these events can occur time-shifted according to the event stream definition.

The division of periodic and aperiodic outgoing events is determined by a gap of time. A gap occurs when the processing of the i-th event finishes before the arrival of the (i+1)-th event. We need the time of the first gap after the finishing of all aperiodic incoming events. At this point it can be assumed that the aperiodic behavior is finished, because the remaining events have a constant execution time and occur periodically. Equation 3 shows how to calculate this point in time. As we only explore systems with a load lower 100%, the gap exists.

$$j = min\{\forall i : RET(i,\tau,ES_{in}) \leq RT(i+1,ES_{in}) \wedge RT(i+1,ES_{in}) \geq min\{a_l | p_l \neq \infty\}\} \quad (3)$$

The equation $RT(i, ES)$ (**R**equest **T**ime) delivers for the i-th event of an event stream the time when it is requested. By means of the modulo calculation in the second case it is possible to ensure that aperiodic events are to be considered only once.

$$RT(i,ES) = \begin{cases} a_i & i \leq N \\ \left\lfloor \frac{i-N_\infty}{N_P} \right\rfloor \cdot p_{((i-N) mod\ N_P+N_\infty)} + a_{((i-N) mod\ N_P+N_\infty)} & i > N \end{cases} \quad (4)$$

The worst case response time has been introduced in [10]. We have adjusted the equation here so that we can use the event stream model with it. Equation

5 gives the length I with k requests of a task processed in the worst case. The calculation of I can be done by a fixpoint iteration.

$$WCRT_k(\tau)=min\{I|I=k\cdot c_\tau+\sum_{\tau'\in HP} E_{\tau'}(I)\cdot c_{\tau'}\} \tag{5}$$

Finally we introduce the equation RET which makes it possible to determine when the i-th request has been processed. The first parameter describes the i-th request, τ the task which is actually explored and ES_{in} the incoming event stream of the task.

$$RET(1,\tau,ES_{in})=WCRT_1(\tau) \tag{6}$$

According to the lemma the first event occurs at the end of the worst case response time. So $RET(1,\tau,ES_{in})$ is the worst case response time of one execution.

$$\forall i\geq 2: RET(i,\tau,ES_{in})=h(RET(i-1,\tau,ES_{in}),RT(i,ES_{in}),\sum_{\tau'\in HP\ \wedge\ ES_\tau=ES_{\tau'}} b_{\tau'},b_\tau,WCRT_1(\tau)) \tag{7}$$

$$h(t,a,b_a,b,WCRT)=\begin{cases} a+b_a+b & t\leq a, \\ t+b & t>a\wedge a<WCRT, \\ t+b_a+b & t>a\wedge a\geq WCRT, \end{cases} \tag{8}$$

Equation 7 applies for the remaining events using their best case execution times. This formula is recursively defined. This is necessary, because it has to be considered whether the previous execution is finished before the new request occurs. Equation 8 has got the time as parameter when the previous execution is finished ($RET(i-1,\tau,ES_{in})$), the time of the current request ($RT(i,ES_{in})$), the sum of the best case executions of the higher priority tasks which are triggered by the same event stream ($\sum_{\tau'\in HP\ \wedge\ ES_\tau=ES_{\tau'}} b_{\tau'}$), the best case execution time of the task (b_τ) and the worst case response time of the task ($WCRT_1(\tau)$).

Equation 8 consists of three cases which have to be considered separately. In the first case the previous execution is finished before the current execution is requested. The result is the sum of the request time and the best case executions of the higher priority tasks and the best case execution of the task itself. In figure 2 we can see this behavior of τ_3 during its third request.

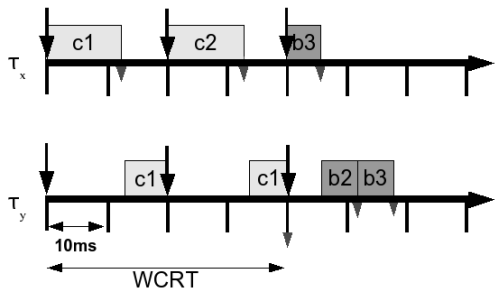

Fig. 3. Example of τ_x and τ_y

In the second case the previous execution is finished later than the current request and this request occurs during the first job of the task (τ_{i1}). In this case we do not have to consider the best case executions of the higher priority tasks, because the executions of those requests have already been taken into account

by the worst case response time calculation. Figure 2 illustrates this case for τ_3. The second execution of task τ_2 doesn't belong to the second execution of τ_3.

The third case is similar to the second case with the exception that the request does not occur within the worst case response time of the task's first job (τ_{i1}). Therefore, we need to sum the executions of the corresponding higher priority tasks. The result is the finishing time of the previous execution plus the best case executions of the higher priority tasks plus the best case execution of the task which is analyzed. This case is shown in figure 3 where the third request of τ_y occurs earlier than the second request has been processed (τ_x and τ_y are triggered by the same ES).

The new scientific contribution of this approach is that the stimulations of the higher priority tasks which are triggered by the same event stream are taken into account which allows to relax the density of the events much more.

4.3 The Communication in the System

As presented in figure 1 we can consider the communication as tasks on a processing element. Each connection over a bus can be considered as task which have a worst case execution time (longest time to send a message) and a best case execution time (shortest time to send a message) and the processing element is the bus itself. During the communication we assume a static behavior. This means either the communication is scheduled by fixed-priorities or by time slices (e.g. static part of a FlexRay Bus). The later would mean that all communication tasks have got the same priority, since the influences of the other tasks are included in the execution times. In the example (section 4.4) we will present such a schedule. The communication over direct connections is trivial and is not discussed here.

4.4 Example:

After the explanation of the equations we will give an example to illustrate the whole concept. In the example we calculate the outgoing event stream *ES9* for τ_7 in figure 1. Thereby we start with the stimulation of τ_3. The calculation consists of three steps.

Outgoing event stream of τ_3: We assume that we have given the event stream $ES1 = \{(1000, 0)^T\}$ and $ES2 = ES3 = \{(200, 0)^T, (200, 10)^T, (200, 60)^T)\}$. Furthermore we need the properties of the tasks. $\tau_1 = (10, 5, 30, 1)$, $\tau_2 = (10, 5, 40, 2)$ and $\tau_3 = (20, 5, 70, 3)$. A cut-out of the gantt-chart of τ_3 is presented in figure 2. Now the outgoing event stream for τ_3 can be computed. First of all the response time will be computed.

$$WCRT_1^0(\tau) = c_\tau + \sum_{\tau' \in HP} E_{\tau'}(c_\tau) \cdot c_{\tau'} = 20 + \sum_{\tau' \in HP} E_{\tau'}(20) \cdot c_{\tau'} = 20 + 20 = 40$$

$$WCRT_1^1(\tau) = c_\tau + \sum_{\tau' \in HP} E_{\tau'}(RT_1^0) \cdot c_{\tau'} = 20 + \sum_{\tau' \in HP} E_{\tau'}(30) \cdot c_{\tau'} = 20 + 30 = 50$$

$$WCRT_1^2(\tau)=c_\tau+\sum_{\tau' \in HP} E_{\tau'}(RT_1^1)\cdot c_{\tau'}=20+\sum_{\tau' \in HP} E_{\tau'}(40)\cdot c_{\tau'}=20+20=50$$

The response time is $WCRT_1(\tau_3) = 50$. Next, the event stream will be set up gradually. Therefore, all tuple for the union function will be passed through.

Passing through the union: $\bigcup_{i=1}^{j} \begin{pmatrix} \infty \\ RET(i,\tau,ES_{in})-WCRT_1(\tau) \end{pmatrix}$	
1.*tuple*:	$\begin{pmatrix} \infty \\ RET(1,\tau_3,ES3)-WCRT_1(\tau_3) \end{pmatrix} = \begin{pmatrix} \infty \\ WCRT_1(\tau_3)-50 \end{pmatrix} = \begin{pmatrix} \infty \\ 50-50 \end{pmatrix} = \begin{pmatrix} \infty \\ 0 \end{pmatrix}$
2.*tuple*:	$\begin{pmatrix} \infty \\ RET(2,\tau_3,ES3)-WCRT_1(\tau_3) \end{pmatrix} = \begin{pmatrix} \infty \\ h(50,10,5,5,50)-50 \end{pmatrix} = \begin{pmatrix} \infty \\ 55-50 \end{pmatrix} = \begin{pmatrix} \infty \\ 5 \end{pmatrix}$

After the second event is the first gap to a periodic event ($j = 2$).

Passing through the union: $\bigcup_{i=j+1}^{N_p+j} \begin{pmatrix} p_i \\ RET(i,\tau,ES_{in})-WCRT_1(\tau) \end{pmatrix}$	
3.*tuple*:	$\begin{pmatrix} 200 \\ RET(3,\tau_3,ES3)-WCRT_1(\tau_3) \end{pmatrix} = \begin{pmatrix} 200 \\ h(55,60,5,5,50)-50 \end{pmatrix} = \begin{pmatrix} 200 \\ 70-50 \end{pmatrix} = \begin{pmatrix} 200 \\ 20 \end{pmatrix}$
4.*tuple*:	$\begin{pmatrix} 200 \\ RET(4,\tau_3,ES3)-WCRT_1(\tau_3) \end{pmatrix} = \begin{pmatrix} 200 \\ h(70,200,5,5,50)-50 \end{pmatrix} = \begin{pmatrix} 200 \\ 210-50 \end{pmatrix} = \begin{pmatrix} 200 \\ 160 \end{pmatrix}$
5.*tuple*:	$\begin{pmatrix} 200 \\ RET(5,\tau_3,ES3)-WCRT_1(\tau_3) \end{pmatrix} = \begin{pmatrix} 200 \\ h(210,210,5,5,50)-50 \end{pmatrix} = \begin{pmatrix} 200 \\ 220-50 \end{pmatrix} = \begin{pmatrix} 200 \\ 170 \end{pmatrix}$

The resulting outgoing event stream of τ_3 is : $ES5= \left\{ \begin{pmatrix} \infty \\ 0 \end{pmatrix}, \begin{pmatrix} \infty \\ 5 \end{pmatrix}, \begin{pmatrix} 200 \\ 20 \end{pmatrix}, \begin{pmatrix} 200 \\ 160 \end{pmatrix}, \begin{pmatrix} 200 \\ 170 \end{pmatrix} \right\}$

Outgoing event stream of τ_5: In this part we have to consider the communication of the bus. As mentioned in the last chapter we assume here a time slice approach. The time slices on the bus are 5 time units. For both tasks applies that the worst case execution time to send a message is 20 time units and the best case execution time to send a message is 10 time units. The execution times are the sum of the time slices which are needed to send a message. The execution interval includes already the time slices of the other tasks. The result is that we do not have to consider other tasks during the calculation of one outgoing event stream. Since we have not enough space and the calculation includes only one event stream we give the result directly.

The outgoing event stream of τ_5 is : $ES7= \left\{ \begin{pmatrix} \infty \\ 0 \end{pmatrix}, \begin{pmatrix} \infty \\ 10 \end{pmatrix}, \begin{pmatrix} 200 \\ 20 \end{pmatrix}, \begin{pmatrix} 200 \\ 150 \end{pmatrix}, \begin{pmatrix} 200 \\ 160 \end{pmatrix} \begin{pmatrix} 200 \\ 200 \end{pmatrix} \right\}$

Outgoing event stream of τ_7: We give $ES6 = \{(1000, 0)^T, (1000, 1995)^T\}$ without calculation. The properties of the tasks are: $\tau_6 = (10, 5, 30, 1)$ and $\tau_7 = (10, 5, 40, 2)$. First of all the response time will be computed.

$$WCRT_1^0(\tau)=c_\tau+\sum_{\tau' \in HP} E_{\tau'}(c_\tau)\cdot c_{\tau'}=10+\sum_{\tau' \in HP} E_{\tau'}(10)\cdot c_{\tau'}=10+10=20$$

$$WCRT_1^1(\tau)=c_\tau+\sum_{\tau' \in HP} E_{\tau'}(RT_1^0)\cdot c_{\tau'}=10+\sum_{\tau' \in HP} E_{\tau'}(20)\cdot c_{\tau'}=10+10=20$$

The response time is $WCRT_1(\tau_7) = 20$. Next, the event stream will be set up gradually. Therefore, all tuple for the union function will be passed through.

Passing through the union: $\bigcup_{i=1}^{j} \begin{pmatrix} \infty \\ RET(i,\tau,ES_{in}) - WCRT_1(\tau) \end{pmatrix}$	
1.*tuple*:	$\begin{pmatrix} \infty \\ RET(1,\tau_7,ES7) - WCRT_1(\tau_7) \end{pmatrix} = \begin{pmatrix} \infty \\ WCRT_1(\tau_7) - 20 \end{pmatrix} = \begin{pmatrix} \infty \\ 20-20 \end{pmatrix} = \begin{pmatrix} \infty \\ 0 \end{pmatrix}$
2.*tuple*:	$\begin{pmatrix} \infty \\ RET(2,\tau_7,ES7) - WCRT_1(\tau_7) \end{pmatrix} = \begin{pmatrix} \infty \\ h(20,5,0,5,20) - 20 \end{pmatrix} = \begin{pmatrix} \infty \\ 25-20 \end{pmatrix} = \begin{pmatrix} \infty \\ 5 \end{pmatrix}$
3.*tuple*:	$\begin{pmatrix} \infty \\ RET(3,\tau_7,ES7) - WCRT_1(\tau_7) \end{pmatrix} = \begin{pmatrix} \infty \\ h(25,5,0,5,20) - 20 \end{pmatrix} = \begin{pmatrix} \infty \\ 30-20 \end{pmatrix} = \begin{pmatrix} \infty \\ 10 \end{pmatrix}$

After the third event is the first gap to a periodic event ($j = 3$).

Passing through the union: $\bigcup_{i=j+1}^{N_p+j} \begin{pmatrix} p_i \\ RET(i,\tau,ES_{in}) - WCRT_1(\tau) \end{pmatrix}$	
4.*tuple*:	$\begin{pmatrix} 200 \\ RET(4,\tau_7,ES7) - WCRT_1(\tau_7) \end{pmatrix} = \begin{pmatrix} 200 \\ h(30,150,0,5,20) - 20 \end{pmatrix} = \begin{pmatrix} 200 \\ 155-20 \end{pmatrix} = \begin{pmatrix} 200 \\ 135 \end{pmatrix}$
5.*tuple*:	$\begin{pmatrix} 200 \\ RET(5,\tau_7,ES7) - WCRT_1(\tau_7) \end{pmatrix} = \begin{pmatrix} 200 \\ h(155,160,0,5,20) - 20 \end{pmatrix} = \begin{pmatrix} 200 \\ 165-20 \end{pmatrix} = \begin{pmatrix} 200 \\ 145 \end{pmatrix}$
6.*tuple*:	$\begin{pmatrix} 200 \\ RET(6,\tau_7,ES7) - WCRT_1(\tau_7) \end{pmatrix} = \begin{pmatrix} 200 \\ h(165,200,0,5,20) - 20 \end{pmatrix} = \begin{pmatrix} 200 \\ 205-20 \end{pmatrix} = \begin{pmatrix} 200 \\ 185 \end{pmatrix}$

The outgoing event stream of τ_7 is : $_{ES9=} \left\{ \begin{pmatrix} \infty \\ 0 \end{pmatrix}, \begin{pmatrix} \infty \\ 5 \end{pmatrix}, \begin{pmatrix} \infty \\ 10 \end{pmatrix}, \begin{pmatrix} 200 \\ 135 \end{pmatrix}, \begin{pmatrix} 200 \\ 145 \end{pmatrix}, \begin{pmatrix} 200 \\ 185 \end{pmatrix} \right\}$

5 Real-Time Analyses

A real-time analysis can be accomplished by analyzing each task individually and perform a continued response time analysis. This means that all requests within the busy period need to be examined. The busy period is the time interval from the first request to the first idle-time of the processor. It can be calculated as follows:

$$BP = min\{I | I \geq \sum_{\tau in \Gamma} E_\tau(I) \cdot c_\tau\} \tag{9}$$

We have to test all $WCRT_k \leq BP$. If all executions meet their deadlines in this interval, the system will always met its deadlines. This can be concluded by the event stream model since the worst case occurs at the beginning. The result is that all executions after the busy period must meet their deadlines, too. As proposed in chapter 4 the calculation can be performed by a fixpoint iteration.

A second approach is described by S. Baruah [11]. In this paper an event-driven real-time analysis with *Recurring Real-Time Tasks* is done. It is no problem to transform that analysis to our model. Baruah demands only two function definitions. The first function is the request bound function delivering to a given time interval the maximum time what a task need for its requested executions. The function is defined as $rbf = E(I) \cdot c_\tau$. The second function is similar to the first with the exception that the deadlines of the executions have to lie in the time interval. This function is $dbf = E(I - d_\tau) \cdot c_\tau$. In order to perform a real time analysis the equation $\forall t : \exists t' \leq t : \left[\left(t' - \sum_{\tau' \in HP} T_{\tau'}.rbf(t')\right) \geq T_\tau.dbf(t)\right]$ must be fulfilled. In the referenced paper it is explained how to conduct this analysis efficiently.

6 Conclusion

The purpose of this paper was to calculate more precise event streams for the real-time analysis of fixed priority systems. This was possible by using the more general event stream model. We have given the calculation of outgoing event streams from tasks which are scheduled by fixed priorities. Thereby we have considered the dependencies caused by the scheduling on a processor. Especially the new consideration of the dependency that tasks can be triggered by the same source leads to a further relaxation.

We have also shown how a real-time analysis can be achieved with the result. This enables us to do a real time analysis for appropriate heterogeneous distributed systems. A further step is to use this approach in connection with the hierarchical event stream model [8] which is more expressive.

References

1. 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)
2. Tindell, K., Clark, J.: Holistic schedulability analysis for distributed hard real-time systems. Microprocessing and Microprogramming **40** (April 1994) 117–134
3. Tindell, K.: Adding time-offsets to schedulability analysis. Internal report, University of York, Computer Science Dept, YCS-94-221. (1994)
4. Gutirrez, J., Garca, J., Harbour, M.: The schedulability analysis for distributed hard real-time systems. In: Proceedings of the 9th Euromicro Workshop on Real-Time Systems, Toledo, Spain. (June 1997) 136–143.
5. Redell, O.: Analysis of tree-shaped transactions in distributed real-time systems. In: ECRTS '04: Proceedings of the 16th Euromicro Conference on Real-Time Systems. (2004) 239–248
6. Henia, R., Ernst, R.: Context-aware scheduling analysis of distributed systems with tree-shaped task-dependencies. In: DATE '05: Proceedings of the conference on Design, Automation and Test in Europe, Washington, DC, USA, IEEE Computer Society (2005) 480–485
7. Henia, R., Ernst, R.: Improved offset-analysis using multiple timing-references. In: DATE '06: Proceedings of the conference on Design, automation and test in Europe, 3001 Leuven, Belgium, Belgium, European Design and Automation Association (2006) 450–455
8. Albers, K., Bodmann, F., Slomka, F.: Hierarchical event streams and event dependency graphs: A new computational model for embedded real-time systems. In: ECRTS '06: Proceedings of the 18th Euromicro Conference on Real-Time Systems, Washington, DC, USA, IEEE Computer Society (2006) 97–106
9. Albers, K., Slomka, F.: An event stream driven approximation for the analysis of real- time systems. In: Proceedings of the 16th Euromicro Conference on Real-Time Systems (ECRTS 04), IEEE (July 2004) 187–195
10. Lehoczky, J., Sha, L., Ding, Y.: The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In: Proceedings of the Real-Time Systems Symposium. (1989) 166–171
11. Baruah, S.K.: Dynamic- and static-priority scheduling of recurring real-time tasks. Real-Time Systems **24**(1) (2003) 93–128