Limiting Event Streams: A General Model to Describe Dependencies in Distributed Hard Real-Time Systems

Steffen Kollmann, Karsten Albers and Frank Slomka
Ulm University
Institute of Embedded Systems/Real-Time Systems
{firstname.lastname}@uni-ulm.de

Internal Report

Abstract

This paper introduces a common approach to describe dependencies in distributed hard real-time systems. Taking dependencies into account will improve schedulability analysis. Thereby, we identify a new kind of dependency not considered in related work. The new model can be also applied to different kinds of dependencies including task offsets allowing an abstract and generalized analysis. Finally, we show how this approach leads to tighter bounds of this analysis.

1. Introduction

Requirements of embedded systems increase from generation to generation. A car, for example, gets more software functionality in each generation which results in more complex embedded systems. New cars have up to 70 ECUs connected by several busses. Most functions have real-time constraints such as the engine management system and the ABS-system.

For a wide acceptance of real-time analysis by industrial software developers tight bounds to the real worst-case response times of tasks are important. A successful approach is the consideration of dependencies of chained task-sets. Previous work considers different kinds of dependencies. Mutual exclusion of tasks, offsets between task stimulation, task chains or tasks competing for the same resource are some examples for such dependencies. But the integration of many other types of dependencies are still open. Especially, missing is a holistic model for dependencies as new abstraction layer. The idea is to separate the calculation and the concrete type of dependencies from the remaining real-time analysis. In other words the calculation of the dependencies is orthogonal to the real-time analysis.

This paper presents a new holistic model to integrate different types of dependencies in real-time analysis. We show how this general model can be integrated into the schedulabilitiy analysis of fixed-priority systems. We also outline the calculation of two totally different kinds of dependencies in order to show the flexibility.

This paper is organized as follows: In chapter 2 an overview of the related work is given. Chapter 3 describes the problem which will be explored. The model is defined in chapter 4. Chapter 5 defines the limiting event stream model and how it can be used to describe dependencies. The resulting real-time analysis is presented in chapter 6. A case study in chapter 7 shows the significance of our approach. The work closes with the conclusion.

2. Related Work

Most considered related work uses the periodic event model with jitter. The analysis for distributed systems was introduced by Tindell and Clark [15]. In this holistic schedulability approach, tasks are considered as independent and so for each task the worst-case response time is calculated separately. This idea has been improved by the transaction model [14] which allows the describtion of static offsets between tasks. Gutierrez et al. [4] extended this work to dynamic offsets so that the offset can vary from one job of a task to another. Furthermore they introduced an idea about mutual exclusion of tasks [5] which is based on offsets between tasks. Since Gutierrez et al. have considered simple task chains, Redell has enhanced the idea to tree-shaped dependencies [13] and Pellizoni et al. applied the transaction model to earliest deadline first scheduling in [12].

Henia et al. used the SymTA/S approach to extended the idea of the transaction model in order to introduce timing-correlations between tasks in parallel paths in distributed systems [6]. This idea has then been improved in [7]. Furthermore in [8] it has been shown that maximum input jitter and worst-case response time must not occur during the same job leading to a relaxation of the response times in a system.

A scalable and modular approach to analyse real-time systems is the real-time calculus (RTC) as described by Wandeler in [16]. Using the RTC it is not possible to describe dependencies like offsets, mutual exclusion of tasks or competing-based dependencies.

Because of the lack of generality or exactness the RTC and the SymTA/S approach are merged in [10]. In this paper the authors do not consider types of dependencies like mutual exclusion of tasks or competing-based dependencies.

A further dependency considering the simultaneous occurrence of events is considered by Kollmann et al. in [9]. In this paper the dependency is directly included into the real-time analysis and not calculated separately as in this paper.

3. Contribution

In figure 1 a typical distributed system is depicted. The system consists of two CPUs and one BUS. We assume fixed priority scheduling on each resource. The system has eight tasks executed on the processors and the bus. The priorities are assigned as described in the figure. The stimulation of the tasks is represented by event streams Θ.

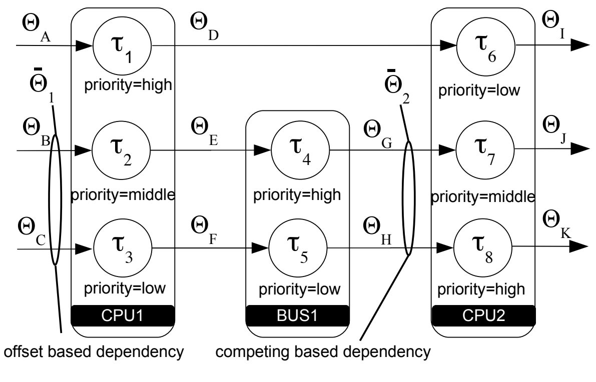

Figure 1. Example of a distributed system having several different dependencies

In order to analyse the system from figure 1 it is necessary to calculate the worst-case response time for each task. The common way to do this is to assume that all tasks and event streams describing the stimulation in a system are independent. This means that events can occur during a worst-case response time analysis in their maximal density, because the context of the system is not considered. The result of a real-time analysis is that the interference between tasks is always maximal and leads to very pessimistic results.

To get tighter response time bounds, we introduce two kinds of dependencies: The first dependency is the competing based dependency describing the situation that tasks executed by the same component compete for this component. Such a competition has the effect that certain events can not occur in the same density when the tasks are assumed not to be independent as it is the case. For example, Θ_G and Θ_H in figure 1.

The second one is an offset based dependency describing that events from different event streams must occur time-shifted to each other. Consider, for example, the event streams Θ_B and Θ_C in figure 1. It is assumed that a correlation between the event streams exists. This has a direct impact on the successive tasks and event streams shown in [13]. The purpose of introducing this dependency is to show the generality of our approach.

These two introduced dependencies lead to tighter bounds for the real-time analysis, It is desirable to include both dependencies into it. In previous work no holistic model as general approach to describe dependencies between tasks is existing.

4. Task and Event Model

In this section we introduce the model necessary for the real-time analysis discussed in section 6.

Task Model: Γ is a set of tasks on one resource $\Gamma = \{\tau_1, ..., \tau_n\}$. A task is a 4-tuple with $\tau = (c^+, c^-, \phi, \Theta)$. c^+ is the worst-case execution time, c^- is the best-case execution time, ϕ is the priority for the scheduling (the lower the number the higher the priority) and Θ defines the stimulation of the task by an event stream. Let τ_{ij} be the j-th job/execution of task τ_i.

We assume that each job of a task generates an event at the end of its execution to notify other tasks.

Event Stream Model: The event stream model gives an efficient general notation for the event bound function.

Definition 1: ([1],[2],[3]) The event bound function $\eta(\Delta t, \Theta)$ gives for every interval Δt an upper bound on the number of events occurring from the event stream Θ in any interval of the length Δt.
So we can set up the following lemma:

Lemma 1: ([3]) The event bound function is a subadditive function, that means for each interval $\Delta t, \Delta t'$:

$$\eta(\Delta t + \Delta t', \Theta) \leq \eta(\Delta t, \Theta) + \eta(\Delta t', \Theta) \tag{1}$$

Proof: $\eta(\Delta t, \Theta)$, $\eta(\Delta t', \Theta)$ return the maximum number of events possible within any Δt or $\Delta t'$. The events in Δt+$\Delta t'$ have to occur either in Δt or in $\Delta t'$. Therefore the condition holds. □

Definition 2: An event stream Θ is a set of event elements θ. Each event element is given by a period p and an offset a ($\theta = (p, a)$).

In cases where the worst-case density of events is unknown for a concrete system an upper bound of the densities can be used to describe the event stream. It is possible to model any event sequence. Only those event sequences for which the condition of subadditivity holds are valid event streams.

Corollary 1: ([3]) The event bound function for an event sequence Θ and an interval Δt is given by:

$$\eta(\Delta t, \Theta) = \sum_{\substack{\theta \in \Theta \\ \Delta t \geq a_\theta}} \left\lceil \frac{\Delta t - a_\theta}{p_\theta} \right\rceil \tag{2}$$

As the inverse function we define the following interval function which gives to a number of events and an event stream the minimum interval in which these events can occur:

Corollary 2: ([3]) The interval function for a number of events and a Θ is given by:

$$\Delta t^{+}(n, \Theta) = min\{\Delta t | \eta(\Delta t, \Theta) = n\} \tag{3}$$

Some examples of event streams can be found in [1].

5. Limiting Event Streams with Dependencies

To extend the previous discussed model of embedded real-time systems we will introduce the limiting event streams in this section.

Definition 3: The limiting event stream is an event stream which defines the maximum occurrence of events for a set of event streams. The limiting event stream is defined as $\overline{\Theta} = (\Theta, \vec{\Theta})$. Θ describes the limiting event stream and $\vec{\Theta}$ represents the set of event streams for which the limiting event stream holds. The limiting event stream fulfills the condition:

$$\eta(\Delta t, \Theta) \leq \sum_{\Theta_i \in \vec{\Theta}} \eta(\Delta t, \Theta_i)$$

Example 1: If no correlations between event streams are defined then $\overline{\Theta} = (\cup_{\Theta_i \in \vec{\Theta}} \Theta_i, \vec{\Theta})$.

Example 2: Figure 1 gives an example for a limiting event stream. Assume $\Theta_B = \Theta_C = \{(20,0)\}$ and an offset of 10 t.u. between these two event streams. The cumulated occurrence of events can be described by the limiting event stream: $\overline{\Theta} = (\{(20,0),(20,10)\},\{\Theta_B,\Theta_C\})$. If we consider the event streams as independent we get two events in an interval $\Delta t = 5$. But the limiting event stream describes how many cumulated events can occur in an interval Δt. With this dependency we get only one event in the interval $\Delta t = 5$.

Next we define how a limiting event stream can be calculated.

Definition 4: Let $\Delta\beta : \Delta t \leftarrow n$ be a limiting interval function which assigns a minimal time interval from a given number of events in dependency from a given relationship of event streams $\vec{\Theta} := \{\Theta_1, \ldots, \Theta_n\}$, then a limiting event stream $\overline{\Theta}$ can be determined by:

$$\overline{\Theta} := \nu(\vec{\Theta}, \Delta\beta(n))$$

Note that $\nu(\vec{\Theta}, \Delta\beta(n))$ and $\Delta\beta(n)$ are abstract formulations which must be concretely formulated for the different types of dependencies.

5.1. Competing Based Dependencies

In this section we introduce a new kind of dependency. We call it competing based dependency. In figure 1 this kind of dependency between tasks is exemplarily depicted. The tasks τ_4 and τ_5 are executed by the same resource. Which means that they compete for the resource. In related work during the analysis of the tasks τ_4 or τ_5 the outgoing event streams Θ_G and Θ_H are considered independently, for example in [15].

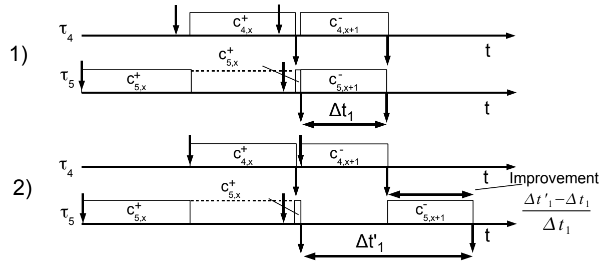

Figure 2. Example of a limiting event stream describing a competing dependency

Let us consider the gantt-diagrams in figure 2. Arrows above the time line represent incoming events. Arrows under the time line represent events generated by the task. In part one of the gantt-chart the case is considered of non competing tasks. The first jobs of the tasks are scheduled in the way that the two outgoing events can occur almost simultaneously. The next events are produced as soon as possible after the first event of the first job. In the independent case the next two events can occur also simultaneously. But this is not possible, since the jobs must be executed task after task, because τ_4 and τ_5 are executed by the same processor. This is depicted in the lower gantt-chart which describes the correct occurrence of the events. Because of the task interference it is not sufficient to consider the outgoing event streams independently from each other. This interference can be modeled by a limiting event stream.

As figure 2 illustrates two cumulated outgoing events can be generated simultaneously. This is based on the fact that the task with the higher priority interrupts the second task just before it finishes. The result is that the two events occur almost simultaneously. This can also be applied on n tasks with the result that n events can occur simultaneously.

For one task we can conclude that at least $(n-1) \cdot c^-$ execution demands must be executed in order to generate n events.

To calculate the limiting event stream we have to determine the minimal distance between n events by formulating the limiting interval function for competing based dependencies.

Lemma 2: Let Γ_R be a subset of m tasks sharing the same processor and $N = \{(n_1, \ldots, n_m) : \sum_{i=1}^{m} = n\}$ the set of distributions of n events, where each task $\tau_i \in \Gamma_R$ produces n_i events, then the limiting interval function is given by:

$$\Delta\beta(n) = \min_{(n_1,\ldots,n_m)\in N} \left(\max\left(\max_{i=1,\ldots,m} \left(\Delta t^+\left(n_i, \Theta_{\tau_i}\right)\right), \left(\sum_{i=1}^{m} (n_i - 1) \cdot c^-_{\tau_i}\right) \right) \right) \quad (4)$$

Proof: Assume that n events can occur in a smaller distance than in the assumption. This would mean that one of the combinations of the minimum results in a shorter distance. Consequently, the interval function $\Delta t^+(n_i, \Theta_{\tau_i})$ or the sum $\sum_{i=1}^{m}(n_i - 1) \cdot c^-_{\tau_i}$ delivers a shorter distance. Assume that the interval function $\Delta t^+(n_i, \Theta_{\tau_i})$ delivers a shorter distance and therefore the events occur in a shorter distance than in the event stream definition. But this is a contradiction according to the event stream definition. Therefore the sum over the best-case execution times must occur in a shorter distance. This can only occur when one of the considered execution times is smaller than the one from the assumption which is a contradiction since we already assume the best-case execution times for all tasks. □

We introduce an algorithm using lemma 2 which delivers for n events the minimum interval in which they occur. The algorithm in figure 3 can be used in conjunction with the methods introduced in [9] to implement this function. In this paper a normalisation for event streams is introduced in order to calculate the $\Delta t^+(n, \Theta)$ efficiently.

```
1  $\Delta\beta(n)$ {
2      $n_i \in \mathbb{N}$;
3      $\Delta e_{old} = \infty$;
4      for (($\forall N | \sum n_i = n$)) {
5          $\Delta t = max\{\Delta t^+(\Theta\tau_i, n_i)\}$
6          $\Delta e = 0$;
7          for ($\forall \tau_i | \tau_i \in \Gamma_R$) {
8              $\Delta e = \Delta e + (n_i - 1) \cdot c^-_{\tau_i}$; }
9          $\Delta e_{old} = min(\Delta e_{old}, max(\Delta e, \Delta t))$; }
10     return $\Delta e_{old}$;}
```

Figure 3. Calculation of the intervals of limiting event streams for competing tasks

The outer loop iterates over all combinations considered by the minimal operation of lemma 2 (line 4 to 10). Line 5 considers all intervals of each event stream as it is done by $\max(\Delta t^+(n_i, \Theta_{\tau_i}))$ of lemma 2. The inner loop (line 7 to 9) calculates the minimal distance produced by the best-case execution times like $\sum_{i=1}^{m}(n_i - 1) \cdot c^-_{\tau_i}$ of lemma 2. Finally, the minimum of all intervals is determined and the minimal interval in which n cumulated events can occur is returned (line 10).

As mentioned above to calculate the concrete event streams via $\nu(\vec{\Theta}, \Delta\beta(n))$ we refer to [9].

5.2. Offset Based Dependencies

In order to show the generality of our new approach we adapt the problem about offsets introduced in the transaction model by [12]. We will only consider static offsets between task stimulation as an example, however this approach covers also dynamic offsets.

Lemma 3: For two strict periodic tasks τ_1 and τ_2 with an offset a we only have to calculate the minimum distance a' between events of τ_1 and τ_2. This minimum distance is calculated by $a' = min(mod(a,x), mod(-a,x))$ using the greatest common divisor $x = gcd(p_{\tau_1}, p_{\tau_2})$ of the periods of the tasks. Leading to the limiting interval function:

$$\Delta\beta(n) = \min(\Delta t^+(n, \{(p_{\tau_1}, 0), (p_{\tau_2}, a')\}), \Delta t^+(n, \{(p_{\tau_1}, a'), (p_{\tau_2}, 0)\})) \quad (5)$$

Proof: For a detailed explanation see [12]. □

Now we can directly set up the event stream via $\nu(\vec{\Theta}, \Delta\beta(n))$: In the case of $mod(a,x) \leq mod(-a,x)$ the limiting event stream is $\overline{\Theta} = (\{(p_{\tau_1}, 0), (p_{\tau_2}, a')\}, \{\Theta_{\tau_1}, \Theta_{\tau_2}\})$. In the case of $mod(-a,x) < mod(a,x)$ the limiting event stream is

$\overline{\Theta} = (\{(p_{\tau_1}, a'), (p_{\tau_2}, 0)\}, \{\Theta_{\tau_1}, \Theta_{\tau_2}\})$. For more than two tasks the approach represented in [12] can be adapted to calculate the limiting interval functions.

6. Real-Time Analysis with Limiting Event Streams

In order to use the limiting event streams it is necessary to adapt the new concept to the real-time analysis, especially the worst-case response time analysis. We have to determine how great the worst-case contribution of tasks in an interval Δt is when limiting event streams are considered.

Lemma 4: The maximal contribution of tasks in an interval Δt occurs when the task with the maximum worst-case execution occurs as much as possible, then the task with the second greatest execution time as much as possible up to the task with the smallest worst-case execution time until the limiting event streams prohibits the occurrence of further events.

Proof: Assume that there is another distribution than the one given by the assumption. Therefore, it must exist at least one event which does not follow the pattern in the assumption. In order to increase the contribution of the tasks, the event must trigger a task whose worst-case execution time is greater than assumed. But this is a contradiction, since we already assume for all tasks with greater worst-case execution times the maximum number of invocations. □

The response time analysis was introduced by Lehoczky et al. [11] and is defined as follows:

Definition 5: If the condition $\forall \tau \in \Gamma : r^+(\tau) \leq d_\tau$ holds, the task set is feasible and the real-time analysis is successful. The worst-case response time of a task considering event streams can be calculated by:

$$r^+(\tau) = \max_{k \in \mathbb{N}} \{ r^+(k,\tau) - \Delta t^+(k,\Theta_\tau) \mid r^+(k-1,\tau) > \Delta t^+(k,\Theta_\tau) \}$$

$$r^+(k,\tau) = \begin{cases} c^+_\tau & k=0 \\ \min\{\Delta t \mid \Delta t = k \cdot c^+_\tau + \underbrace{\sum_{\tau' \in HP} \eta(\Delta t, \Theta_{\tau'}) \cdot c^+_{\tau'}}_{calcHPStatic} \} & k \geq 1 \end{cases} \quad (6)$$

The amount of executions produced by higher priority tasks can be calculated by the event bound function multiplied by the worst-case execution time. By means of a fixed-point iteration the worst-case response time can be calculated for every job k.

To take the limiting event streams into account lemma 4 is used. To implement $\sum_{\tau' \in HP} \eta(\Delta t, \Theta_{\tau'}) \cdot c^+_{\tau'}$ the algorithm in figure 4 was developed. The rest of equation 6 is unmodified.

```
1  // $\Delta t$ The interval which is considered
2  // $k$ Number of calls of $\tau$
3  // $\tau$ The task under analysis
4  // $\overline{\Theta}_{all}$ All necessary limiting event streams
5  // $\Gamma_{HP}$ Set of $\tau'$: $\phi_\tau > \phi_{\tau'}$
6  calcHPStatic($\Delta t, k, \tau, \overline{\Theta}_{all}, \Gamma_{HP}$) {
7      $\Delta e = 0$;
8      sort($\Gamma_{HP}$); // $\forall i \leq j : c_{\tau_i} \geq c_{\tau_j}$;
9      $(\forall \overline{\Theta} \in \overline{\Theta}_{all}) : \eta[\overline{\Theta}] = \eta(\Delta t, \Theta_{\overline{\Theta}})$;
10     $(\forall \overline{\Theta} \in \overline{\Theta}_{all} \mid \Theta_\tau \in \vec{\Theta}_{\overline{\Theta}}) : \eta[\overline{\Theta}] = \eta[\overline{\Theta}] - k$;
11     for($\tau_j \in \Gamma_{HP}$) {
12         n = $\eta(\Delta t, \Theta_{\tau_j})$;
13         m = min($\eta[\overline{\Theta}] \mid \Theta_{\tau_j} \in \vec{\Theta}_{\overline{\Theta}}$);
14         $\Delta$e = $\Delta$e + min(n,m) $\cdot$ $c^+_{\tau_j}$;
15         for(($\forall \overline{\Theta} \in \overline{\Theta}_{all} \mid \Theta_{\tau_j} \in \vec{\Theta}_{\overline{\Theta}}$) {
16             $\eta[\overline{\Theta}] = \eta[\overline{\Theta}]$ - min(n,m); } }
17     return $\Delta$e; }
```

Figure 4. Calculation of the contribution of higher priority tasks for the worst-case response time with limiting event streams

The algorithm has as parameters the interval Δt which is considered, k the job number of the task under analysis, τ the task which is explored, $\overline{\Theta}_{all}$ the set of the necessary limiting event streams and Γ_{HP} containing all tasks having a higher priority than τ. The algorithm sorts the tasks by their worst-case execution times (line 8) and stores for every limiting event stream the maximum amount of events which this stream allows within Δt (line 9). The number of invocations of the task under analysis τ must be subtracted from the corresponding limiting event streams (line 10). In a loop (line 11 to 16) all higher priority tasks are considered. The task with the greatest worst-case execution time is considered first according to lemma 4. The algorithm determines the maximum amount of invocations for the task by the event stream of the task (line 12) and the bound of the event stream if one exists (line 13). The minimum of these are used to calculate the maximum contribution of the task within Δt (line 14). The second loop (line 15 to 16) reduces the corresponding limiting event streams by the used events (line 16). Therefore the loops distribute the amount of events of the limiting event streams over the tasks according to lemma 4. This leads to the worst-case contribution of higher priority tasks within Δt.

Note, that the complexity of the response time analysis is still pseudo-polynominial. The complexity to calculate the limiting event streams depends on the kind of the dependency which is considered. To calculate the problem of competing-based dependencies can become challenging, because of its combinatorial complexity. The analysis, however, is not affected by this problem. So it is suggestive to find upper bounds for the limiting event streams to improve the runtime performance.

7. Case Study

The significance of this new approach is shown by the following case study. The system to explore is depicted in figure 1 and described in chapter 3. Table 1 gives the parameters for the system and table 2 the event streams. We have chosen this system, because it is easy to follow and it shows the new methodology in the whole.

CPU 1	τ_1	τ_2	τ_3
c^+	40	30	40
c^-	30	20	20
ϕ	1	2	3
Θ	Θ_A	Θ_B	Θ_C

BUS 1	τ_4	τ_5
c^+	90	90
c^-	50	80
ϕ	1	2
Θ	Θ_E	Θ_F

CPU 2	τ_6	τ_7	τ_8
c^+	50	35	35
c^-	30	25	25
ϕ	3	2	1
Θ	Θ_D	Θ_G	Θ_H

Table 1. Parameters of the distributed system which is depicted in figure 1

To calculate the event streams of the system, we use the approach given in [9]. The resulting event streams in the system are shown in the table 2. Thereby, we compare the event streams calculated with dependencies versus ones without dependencies. A static offset of 100 t.u. between the event streams Θ_B and Θ_C is assumed.

Θ	with dependencies	without dependencies
Θ_A	$\{(100,0)\}$	$\{(100,0)\}$
Θ_B	$\{(200,0)\}$	$\{(200,0)\}$
Θ_C	$\{(200,0)\}$	$\{(200,0)\}$
Θ_D	$\{(\infty,0),(100,90)\}$	$\{(\infty,0),(100,90)\}$
Θ_E	$\{(\infty,0),(200,150)\}$	$\{(\infty,0),(200,150)\}$
Θ_F	$\{(\infty,0),(200,140)\}$	$\{(\infty,0),(200,70)\}$
Θ_G	$\{(\infty,0),(200,100)\}$	$\{(\infty,0),(200,100)\}$
Θ_H	$\{(\infty,0),(\infty,80),(\infty,160),(200,310)\}$	$\{(\infty,0),(\infty,80),(\infty,160),(\infty,240),(200,370)\}$
Θ_I	$\{(\infty,0),(\infty,30),(\infty,60),(100,130)\}$	$\{(\infty,0),(\infty,30),(\infty,60),(\infty,90),(100,165)\}$
Θ_J	$\{(\infty,0),(200,55)\}$	$\{(\infty,0),(200,55)\}$
Θ_K	$\{(\infty,0),(\infty,70),(\infty,150),(200,300)\}$	$\{(\infty,0),(\infty,70),(\infty,150),(\infty,130),(200,360)\}$

Table 2. All event streams of the distributed system. The results are computed with as well as without the approach.

To determine the outgoing event streams with dependencies it is necessary to calculate the limiting event streams of the system. We consider only two limiting event streams $\overline{\Theta}_1$ and $\overline{\Theta}_2$. $\overline{\Theta}_1$ describes the offset between Θ_B and Θ_C. $\overline{\Theta}_2$ describes the competing based dependency between Θ_G and Θ_H.

$\overline{\Theta}$	Θ	$\vec{\Theta}$
$\overline{\Theta}_1$	$\{(200,0),(200,100)\}$	$\{\Theta_B,\Theta_C\}$
$\overline{\Theta}_2$	$\{(\infty,0),(\infty,0),(\infty,80),(\infty,130),(\infty,210),(200,300),(200,310)\}$	$\{\Theta_G,\Theta_H\}$

Table 3. Results of the calculated limiting event streams

After calculating the event streams, we have a closer look on the improvements in the analysis of the system. At first, some event streams and the improvement of the density in the system are considered. This is depicted in table 4 and figure 5.

n	Θ_F	Θ_F'	*imp.*	Θ_H	Θ_H'	*Imp.*	Θ_I	Θ_I'	*imp.*
1	0	0	0%	0	0	0%	0	0	0%
2	140	70	50%	80	80	0%	30	30	0%
3	340	270	20,58%	160	160	0%	60	60	0%
4	540	470	12,96%	310	240	22,5%	130	90	30,7%
5	740	670	9,45%	510	370	27,45%	230	165	28,26%
6	940	870	7,44%	710	570	19,71%	330	265	19,69%
7	1140	1070	6,14%	910	770	15,38%	430	365	15,11%
8	1340	1270	5,22%	1110	970	12,61%	530	465	12,26%
9	1540	1470	4,54%	1310	1170	10,68%	630	565	10,31%
10	1740	1670	4,02%	1510	1370	9,27%	730	665	8,9%

Table 4. This shows the improvement of the approach on the event streams Θ_F,Θ_H and Θ_I. Θ shows the intervals with the dependency, Θ' shows the intervals without the dependency, improvement is given in %

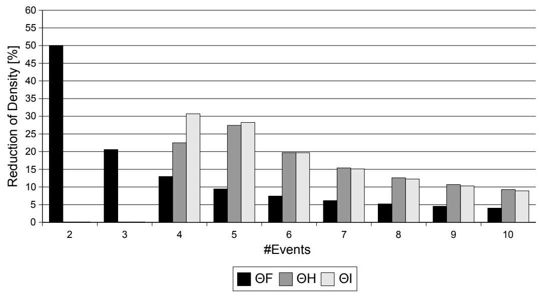

Figure 5. This figure shows the improvements of the intervals in percent of the event stream Θ_F,Θ_H and Θ_I.

The dependencies have not only an influence on the density of the event streams, but also a direct influence on the worst-case response times. The worst-case response time of the task τ_3 has been reduced from 150 t.u. to 80 t.u. This means that the result of the analysis with dependencies is in this case 46,66% tighter compared to the analysis without dependencies. The task τ_6 has a worst-case response time without dependencies of 255 t.u. and with dependencies of 205 t.u., which is a reduction of the worst-case response time of 19,6%.

This synthetical example shows that dependencies can improve the real-time analysis. Thereby we have shown how easy different dependencies can be combined in a general approach.

8. Conclusion

We have shown the possibililty to achieve a holistic model for task dependencies in distributed real-time systems. The new approach has been applied to fixed-priority systems. We have shown by two kinds of dependencies how these can be described by the new defined limiting event streams. Thereby, a new kind of dependency has been introduced. With the effect, that we have cut the complexity of the dependencies from the real-time analysis.

Finally, a case study has been conducted to show the improvements of the approach. Despite the example is synthetical, it has been shown that our concept works for different kinds of dependencies.

In the future we will show how more kinds of dependencies can be integrated by this new model and how the limiting event streams can be propagated through the system. Furthermore, the integration of the limiting event streams into approximative real-time analysis like the real-time calculus [16] or the hierarchical event streams [1] to improve the runtime performance is also an aim.

References

[1] Karsten Albers, Frank Bodmann, and Frank Slomka. 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*, pages 97–106, Washington, DC, USA, 2006. IEEE Computer Society.

[2] Karsten Albers and Frank Slomka. An event stream driven approximation for the analysis of real- time systems. In *ECRTS '04: Proceedings of the 16th Euromicro Conference on Real-Time Systems*, pages 187–195. IEEE, July 2004.

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

[4] J. C. Palencia Gutierrez and Michael Gonzalez Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *RTSS*, page 26 ff, 1998.

[5] J. C. Palencia Gutierrez and Michael Gonzalez Harbour. Exploiting precedence relations in the schedulability analysis of distributed real-time systems. In *IEEE Real-Time Systems Symposium*, pages 328–339, 1999.

[6] Rafik Henia and Rolf Ernst. 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*, pages 480–485, Washington, DC, USA, 2005. IEEE Computer Society.

[7] Rafik Henia and Rolf Ernst. Improved offset-analysis using multiple timing-references. In *DATE '06: Proceedings of the conference on Design, automation and test in Europe*, pages 450–455, 3001 Leuven, Belgium, Belgium, 2006. European Design and Automation Association.

[8] Rafik Henia, Razvan Racu, and Rolf Ernst. Improved output jitter calculation for compositional performance analysis of distributed systems. In *Proceedings Workshop on Parallel and Distributed Real-Time Systems*, March 2007.

[9] Steffen Kollmann, Karsten Albers, and Frank Slomka. Effects of simultaneous stimulation on the event stream densities of fixed-priority systems. In *Spects'08: Proceedings of the International Simulation Multi-Conference*. IEEE, June 2008.

[10] Simon Kuenzli, Arne Hamann, Rolf Ernst, and Lothar Thiele. Combined approach to system level performance analysis of embedded systems. In *CODES+ISSS '07: Proceedings of the 5th IEEE/ACM international conference on Hardware/-software codesign and system synthesis*, pages 63–68, New York, NY, USA, 2007. ACM.

[11] John P Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Proceedings of the 11th IEEE Real-Time Systems Symposium*, pages 201–209, December 1990.

[12] Rodolfo Pellizzoni and Giuseppe Lipari. Improved schedulability analysis of real-time transactions with earliest deadline scheduling. In *RTAS '05: Proceedings of the 11th IEEE Real Time on Embedded Technology and Applications Symposium*, pages 66–75, Washington, DC, USA, 2005. IEEE Computer Society.

[13] Ola Redell. Analysis of tree-shaped transactions in distributed real-time systems. In *ECRTS '04: Proceedings of the 16th Euromicro Conference on Real-Time Systems (ECRTS'04)*, pages 239–248, Washington, DC, USA, 2004. IEEE Computer Society.

[14] Ken Tindell. Adding time-offsets to schedulability analysis. Technical report, University of York, Computer Science Dept, YCS-94-221, 1994.

[15] Ken Tindell and John Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and Microprogramming*, 40:117–134, April 1994.

[16] Ernesto Wandeler. *Modular Performance Analysis and Interface-Based Design for Embedded Real-Time Systems*. PhD thesis, ETH Zurich, September 2006.