Response-Time Analysis for Transactions with Execution-Time Dependencies

Jukka Mäki-Turja
Mälardalen Real-Time research Centre
& Arcticus Systems AB
jukka.maki-turja@mdh.se

Mikael Sjödin
Mälardalen Real-Time research Centre
mikael.sjodin@mdh.se

Abstract

Mature scientific research results in the area of schedulability analysis have had a very limited impact on real industrial applications. This, we believe, is that current models are not able to accurately capture the complex temporal behavior of actual systems. In this paper we address a common assumption made in schedulability analysis methods that tasks can experience their worst case execution time independently from each other. This assumption is not very realistic for real systems since tasks collaboratively often perform certain functionality and thus depend on each other.

Our aim, in this paper, is to capture execution time dependencies between tasks and to take advantage of this information when performing response time analysis. We introduce the concept of execution modes to the task model with offsets (transactional task model). The execution modes are used as a generic way to specify temporal dependencies between tasks that execute within a transaction. We then present extensions to the Response-Time Analysis (RTA) theory to analyze transactions with execution modes.

1. Introduction

Our aim with this paper is to address one aspect leading to a discrepancy between academic results and industrial needs within the area of schedulability analysis of real-time systems.

The importance of schedulability analysis of real-time systems is quickly increasing. Today, almost all electrical products of some complexity are controlled by an embedded computing system. Often, these products need to interact with an environment in a timely manner, i.e. the computer system is a real-time system. Furthermore, a large class of embedded real-time systems are also safety critical, meaning that a system failure can have potentially catastrophic consequences. These safety-critical real-time systems are found, for instance, in vehicles, robotics, medical appliances, and production facilities.

For safety-critical computer systems, the society is increasing the pressure on system providers to provide evidence that the system if safe. This paper will not dwell into the many important issues of demonstrating safety of a computer controlled systems, e.g. as mandated by the safety standard IEC 61508 [4]. However, one important activity in order to establish the safety of a real-time system is to provide evidence that actions will be provided in a timely manner (e.g. each actions will be taken at a time that is appropriate to the environment of the system). For systems consisting of multiple concurrent/semi-concurrent operating-system tasks the number of possible execution scenarios for each actions is daunting [23], and effectively prohibits testing as a means for verifying the correct timing of actions.

To complement testing, and to provide stronger evidence for correct timing, academia has developed techniques to make *a priori* analysis to verify that each action in a system will be performed before its deadline. These, so called, schedulability analysis techniques have been continuously developed over almost four decades [20]. So, from an academic point of view, schedulability analysis can be considered as a mature technology. However, studying the industrial penetration of schedulability gives a very disappointing image. It is very difficult to find reports of successful use of schedulability analysis in real industrial systems. In fact, it is probably easier to find documents of negative results of trying to use schedulability analysis in industrial systems [13, 27]. This disappointing picture, we believe, is because traditional real-time analysis models are not applicable for large and complex real-time systems [27].

The core problem is that the scientific community uses too simplistic or research oriented timing models. The models stemming from academia do not fit well with the structure of real systems. Thus, extracting a timing model that is amenable for analysis may prove prohibitively difficult. And even if a model can be extracted, it may not capture real system scenarios well. Thus, results from analyzing these models do not reflect real system behavior, leading to unnecessary pessimistic timing predictions [17].

In this paper we address the common assumption that every task in the system experiences its worst case execution time simultaneously at the critical instant. We try to capture and express execution time dependencies between tasks and use this information when performing response time analysis in order to get more accurate schedulability analysis results. We do this by extending the task model with offsets, also known as the transactional task model, with the concept of execution modes. A mode defines a tasks execution context, i.e., a task can execute different parts of its code depending of its context. Such a context can for example be dependant on input data produced by a preceding task.

Paper Outline: We continue the paper by presenting a background to schedulability and Response Time Analysis (RTA) theory in section 2. In section 3 we introduce and discuss execution-time dependencies. In section 4 we extend the transactional task model to express execution-time dependencies and present the resulting RTA formulae. Finally, in section 5 we conclude the paper and discuss some future work.

2. RTA background

Response-Time Analysis (RTA) [1, 20] is a powerful and well established schedulability analysis technique. RTA is a method to calculate upper bounds on response-times for tasks in real-time systems. In essence RTA is used to perform a schedulability test, i.e., checking whether or not tasks in the system will satisfy their deadlines. RTA is applicable for, e.g., systems where tasks are scheduled in priority order which is the predominant scheduling technique used in industrial systems today. Furthermore, RTA is not only used as a schedulability analysis tool, but it is also used in a wider context. For example, schedulability analysis is performed in the inner loop of optimization or search techniques such as task attribute assignment and task allocation [7]. These methods require that RTA methods provide tight response times and that implementations are efficient in order to be useful in engineering tools for resource constrained real-time systems.

To be able to calculate less pessimistic response times in systems where tasks may have dependencies in their release times, Tindell introduced RTA for a task model with offsets, the transactional task model, [24]. Palencia Gutiérrez and González Harbour formalized and extended the work of Tindell in [18].

Liu and Layland [14] provided the theoretical foundation for analysis of fixed priority scheduled systems. Joseph and Pandya presented the first RTA [10] for the simple Liu and Layland task model which assumes independent periodic tasks. Since then RTA has been applied and extended in a numerous ways, e.g., [2, 3]. Extensions include lifting the independent task assumption [21, 22], analyzing communication networks [5, 26], fault tolerant systems [19], distributed systems [25], modeling OS overhead [11], etc. So from a scientific perspective RTA has become a well established and mature technology. A more detailed discussion of some of these improvements can be found in "A Practitioners Handbook for Real-Time Analysis" [6]. This book is focused on a practitioner's point of view and thus aims at applying RTA in an engineering context. A historical perspective of real-time scheduling research, where RTA is a big part, can be found in [20].

3. Execution-time dependencies

A prerequisite for schedulability analysis techniques for hard real-time systems is that the worst case execution time (WCET) is identified for every task in the system. Furthermore, it is assumed that the execution time of each task is independent of other tasks in the system, i.e. that there exist a scenario where all tasks exhibit their WCET at the same time. However, this assumption is not very realistic since many tasks collaborate to achieve a specific functionality and/or may have data dependencies among them. This will result in overestimated response times.

For example, it is common that the WCET represents an execution path that contains error recovery code. In well-engineered systems, the errors should not propagate longer than necessary; e.g. one would strive to contain the errors within a single task. In such systems, is likely that only one error-handler is executed in a sequence of tasks. If this is the case, using the WCET for all tasks in that sequence would greatly reduce schedulability of the system. Another example is when systems are engineered to have distinct functional modes. Within each mode only a subset of the system's functionality is active. However, rather than reconfiguring the set of active tasks and the signal-paths between them it is quite common to keep the task-set and signal-paths unchanged over different modes, and instead activate or deactivate different execution-paths within the tasks.

In this paper we consider the transactional task-model (a.k.a. the model for tasks with offsets) introduced by Tindell [24]. In this model tasks are grouped into transactions and assigned given a release-time within the transaction. A transaction is activated by an outside event (or clock) and the last task in the transaction typically produces some output. Throughout the paper we will use an example transaction with two tasks shown in Fig. 1. In this example the two tasks has two distinct internal functions (A and B in Task 1, and C and D in Task 2), each task will only execute one of its functions per activation (e.g. depending on the value of some global mode-variable). Furthermore, we assume in this example that when Task 1 executes A, Task 2 will execute C, and when Task 1 executes B, Task 2 will execute D. Thus, the transaction has two distinct execution behaviors: A followed by C, and B followed by D. We call each such execution behavior for a *mode*.

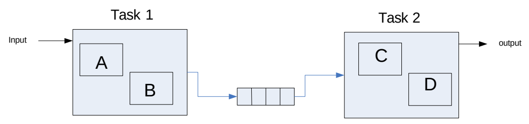

Figure 1. Example Transaction

4. RTA with execution-time dependencies

In this section we outline the existing transactional task-model and its response-time analysis; and present our extension to the model and analysis needed to capture execution-time dependencies in transaction modes.

4.1. System model

The transactional task model was introduced to model tasks which have dependencies in their release times. Tindell introduced the notions of transactions and timed offsets for tasks within the transactions [24]. Palencia Gutiérrez and González Harbour [18] showed how to use this model to also model precedence relations within a transaction. These papers both present exact (exponential time-complexity) and approximate (polynomial complexity) analysis. Later we revealed hidden pessimism in the approximate analysis [15]. In this paper we extend our previous analysis.

4.1.1 Original model

The transactional task model is defined as follows: The system, Γ, consists of a set of k transactions $\Gamma_1, \ldots, \Gamma_k$. Each transaction Γ_i is activated by a periodic sequence of events with period T_i (for non-periodic events T_i denotes the minimum inter-arrival time between two consecutive events). The activating events are mutually independent, i.e., phasing between them is arbitrary. A transaction, Γ_i, contains $|\Gamma_i|$ number of tasks, and each task may not be activated (released for execution) until a time, *offset*, elapses after the arrival of the external event.

We use τ_{ij} to denote a task. The first subscript denotes which transaction the task belongs to, and the second subscript denotes the number of the task within the transaction. A task, τ_{ij}, is defined by a worst case execution time (C_{ij}), an offset (O_{ij}), a deadline (D_{ij}), maximum jitter (J_{ij}), maximum blocking from lower priority tasks (B_{ij}), and a priority (P_{ij}). The system model is formally expressed as follows:

$$\begin{aligned}\Gamma &:= \{\Gamma_1, \ldots, \Gamma_k\} \\ \Gamma_i &:= \langle\{\tau_{i1}, \ldots, \tau_{i|\Gamma_i|}\}, T_i\rangle \\ \tau_{ij} &:= \langle C_{ij}, O_{ij}, D_{ij}, J_{ij}, B_{ij}, P_{ij}\rangle\end{aligned}$$

There are no restrictions placed on offset, deadline or jitter, i.e., they can each be either smaller or greater than the period.

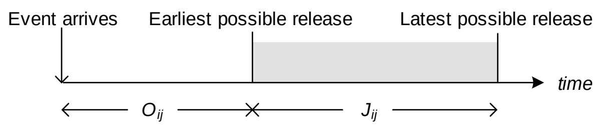

Figure 2. Relation between an event arrival, offset, jitter and task release

The relation between event arrival, offset, jitter and task release is graphically visualized in Fig. 2. After the event arrival, task τ_{ij} is not released for execution until its offset (O_{ij}) has elapsed. The task release may be further delayed by jitter (maximally until $O_{ij} + J_{ij}$) making its exact release uncertain. For a more extensive explanation of task parameters see [18].

The parameters for our example transaction from Fig. 1 is graphically presented in Fig. 3. To simplify the example we have zero jitter ($J_{ij} = 0$) and blocking ($B_{ij} = 0$) for each task. (In this section we ignore our assumed dependency between the function A-C and B-D.)

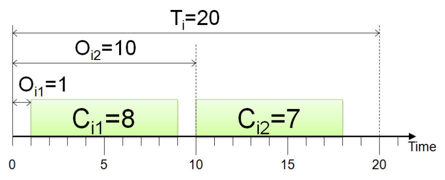

Figure 3. An example transaction Γ_i

4.1.2 Task model with modes

In order to express execution time dependencies we introduce the concept of modes. A set of modes, $\{\mu_{i1}, \mu_{i2}, \ldots, \mu_{im}\}$, are expressed for each Γ_i. Furthermore, it is no longer sufficient to express the worst case execution time with a single number, C_{ij}. Instead each task has an execution time corresponding to a mode μ, C_{ij}^{μ}. Formally, the model is expressed by:

$$\begin{aligned}\Gamma_i &:= \langle\{\tau_{i1}, \ldots, \tau_{i|\Gamma_i|}\}, T_i, \{\mu_1, \mu_2, \ldots, \mu_s\}\rangle \\ \tau_{ij} &:= \langle\{C_{ij}^1, \ldots, C_{ij}^s\}, O_{ij}, D_{ij}, J_{ij}, B_{ij}, P_{ij}\rangle\end{aligned}$$

For C_{ij}^{μ} the first subscript, i, denotes which transaction the task belongs to, the second subscript, j, denotes the number of the task within the transaction, and the superscript denotes the mode where this execution time is applicable.

An execution time must be specified for every mode. However, if no execution time information is available for a mode it is always safe to assume the traditional worst case execution time for that mode. In this model any number of dependencies between tasks within a transaction can be modeled.

In section 3 we presented an example transaction that has two modes. For the rest of this paper we will assume the timing-parameters presented in Fig. 4 for our example transaction. Thus, the transaction Γ_i is formalized as:

$$\begin{aligned}\Gamma_i &:= \langle\{\tau_{i1}, \tau_{i2}\}, 20, \{\mu_1, \mu_2\}\rangle \\ \tau_{i1} &:= \langle\{8, 5\}, O_{i1} = 1, D_{i1}, J_{i1} = 0, B_{i1} = 0, P_{i1}\rangle \\ \tau_{i2} &:= \langle\{3, 7\}, O_{i2} = 10, D_{i2}, J_{i2} = 0, B_{i2} = 0, P_{i2}\rangle\end{aligned}$$

Where C_{i1}^1 is the WCET of A, C_{i1}^2 the WCET of B, C_{i2}^1 the WCET of C, and C_{i2}^2 the WCET of D. Note that these execution-times are consistent with the WCETs presented in Fig. 3; where we ignored the dependencies A-C, and B-D.

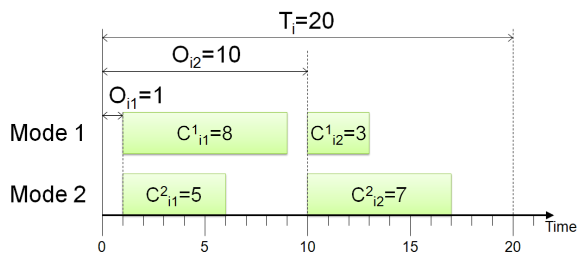

Figure 4. the transaction Γ_i with modes

4.2. RTA formulae

The goal of RTA is to facilitate a schedulability test for each task in the system by calculating an upper bound on its worst case response-time. We use τ_{ua} (task a, belonging to transaction Γ_u) to denote the *task under analysis*, i.e., the task which response time we are currently calculating.

In the classical RTA (without offsets) the *critical instant* for τ_{ua} occurs when it is released at the same time as all higher priority tasks [10, 14]. In a task model with offsets this assumption yields pessimistic response-times since some tasks cannot be released simultaneously due to offset relations. Therefore, Tindell [24] relaxed the notion of critical instant to be:

> At least one task out of every transaction is to be released at the critical instant. (Only tasks with priority higher than τ_{ua} are considered.)

Since it is not known which task coincides with (is released at) the critical instant, every task in a transaction must be treated as a *candidate* to coincide with the critical instant.

Tindell's exact RTA evaluates the response-time for every possible combination of candidates among all transactions in the system. This, however, becomes computationally intractable for anything but small task sets (the number of possible combinations of candidates is m^n for a system with n transactions and with m tasks per transaction). Therefore Tindell provided an approximate RTA that still gives good results but uses one single approximation function for each transaction.

4.2.1 Interference function

Central to RTA is to capture the worst case interference a higher priority task (τ_{ij}) causes the task under analysis (τ_{ua}) during an interval of time t. Since a task can interfere with τ_{ua} multiple times during t, we have to consider interference from possibly several *instances*. The interfering instances of τ_{ij} can be classified into two sets:

$Set1$ Activations that occur before or at the critical instant and that can be delayed by jitter so that they coincide with the critical instant.

$Set2$ Activations that occur after the critical instant

When studying the interference from an entire transaction Γ_i, we will consider each task, $\tau_{ic} \in \Gamma_i$, as a *candidate* for coinciding with the critical instant.

RTA for tasks with offsets is based on two fundamental theorems [18]:

1. The worst case interference a task τ_{ij} causes upon τ_{ua} is when $Set1$ activations are delayed by an amount of jitter such that they all occur at the critical instant and the activations in $Set2$ have zero jitter.

2. The task of Γ_i that coincides with the critical instant (denoted τ_{ic}), will do so after experiencing its worst case jitter delay.

In order to determine the amount of $Set2$ interference for a task, τ_{ij}, we need to know when the first activation of τ_{ij} occurs after the critical instant. This phasing between a task, τ_{ij}, and the critical instant, which according to theorem 2 occurs at $O_{ic} + J_{ic}$, becomes:

$$\Phi_{ijc} = (O_{ij} - (O_{ic} + J_{ic})) \bmod T_i \tag{1}$$

For our example transaction we have:

$$\Phi_{i11} = 0,\ \Phi_{i21} = 9$$
$$\Phi_{i22} = 0,\ \Phi_{i12} = 11$$

That is, if task τ_{i1} is the task coinciding with the critical instant task τ_{i2} will be released 9 time units after the critical instant. If, on the other hand, τ_{i2} is the one coinciding with the critical instant task τ_{i1} will be released 11 time units after the critical instant.

For each mode μ and each critical instant candidate c, the interference from task τ_{ij} is captured by two parts:

1. The part caused by instances in $Set1$ (which is independent of the time interval t), $I^{Set1}_{ijc}(\mu)$, and

2. the part caused by instances in $Set2$ (which is a function of the time interval t during which interference can occur), $I^{Set2}_{ijc}(t,\mu)$.

These are defined as follows:

$$\begin{aligned}
I^{Set1}_{ijc}(\mu) &= \left\lfloor \frac{J_{ij} + \Phi_{ijc}}{T_i} \right\rfloor C^{\mu}_{ij} \\
I^{Set2}_{ijc}(t,\mu) &= \left\lceil \frac{t - \Phi_{ijc}}{T_i} \right\rceil C^{\mu}_{ij} - x \\
x &= \begin{cases} 0 & t^* \le 0 \\ 0 & t^* \bmod T_i = 0 \\ 0 & t^* \bmod T_i \ge C^{\mu}_{ij} \\ C^{\mu}_{ij} - (t^* \bmod T_i) & \text{otherwise} \end{cases} \\
t^* &= t - \Phi_{ijc}
\end{aligned} \tag{2}$$

The x in the definition of $I_{ijc}^{Set2}(t,\mu)$ was introduced in [15] where it was recognized that the last task instance may not interfere with its full execution time. This has an impact on the approximate RTA which will result in tighter analysis results.

The worst case interference, in mode μ, transaction Γ_i poses on τ_{ua}, during a time interval t, when candidate τ_{ic} coincides with the critical instant, is:

$$W_{ic}(\tau_{ua},t,\mu) = \sum_{\forall j \in hp_i(\tau_{ua})} \Big(I_{ijc}^{Set1}(\mu) + I_{ijc}^{Set2}(t,\mu) \Big) \tag{3}$$

Where $hp_i(\tau_{ua})$ denotes tasks belonging to transaction Γ_i, with priority higher to that of the priority of τ_{ua}.

In our example transaction we have two tasks (thus two critical instant candidates), and two modes. This gives us a total for four interference functions, these are visualized in Fig. 5(a) to 5(d). In general, for a transaction with $|\Gamma_i|$ tasks and s modes we get $|\Gamma_i|s$ number of interference functions to consider.

4.2.2 Approximation function

In order to obtain the response time of task τ_{ua} we need to apply $W_{ic}(\tau_{ua},t,\mu)$ for all transactions in the system, including the transaction of τ_{ua}. However, in order to do this we need to determine which task in each transaction coincides with the critical instant. Since we cannot know this beforehand we need to examine all possible variations (permutations) of one task out of every transaction, and choose the variation that leads to the worst case response time for τ_{ua}. This is computationally intractable (exponential growth in relation to the number of tasks) for anything but small task sets. Furthermore, the concept of modes adds an extra dimension of complexity since we also need to examine all possible combination of modes.

Therefore we focus our attention on extending the approximate analysis that grows polynomially in relation to the number of tasks and modes.

The approximate analysis defines one single, upward approximated, function for the interference caused by transaction Γ_i:

$$W_i^*(\tau_{ua},t) = \max_{\forall \mu \in \Gamma_i} \Big[\max_{\forall c \in hp_i(\tau_{ua})} \big[W_{ic}(\tau_{ua},t,\mu) \big] \Big] \tag{4}$$

That is, $W_i^*(\tau_{ua},t)$ simply takes the maximum of each interference function (for each candidate τ_{ic} and each possible mode μ). This is, for our example transaction, graphically visualized in Fig. 5(e) and 5(f).

4.2.3 Obtaining the response time

Given the interference (W_i^*) each transaction causes, during a time interval of length t, the response time of τ_{ua} for a given mode μ, can be calculated.

Since several task instances (jobs) can be active at the same time (deadlines can be larger than periods) we need to determine how many number of jobs of τ_{ua} are activated, and thus need to be considered within the busy period [12]. The length of a busy period, for τ_{ua}, assuming τ_{uc} is the candidate critical instant, is defined as (Note that the approximation function is not used for interference from tasks within Γ_u, instead the exact interference function $W_{uc}()$ is used):

$$\begin{aligned} L_{uac}(\mu) = & B_{ua} + (p_{L,uac}(\mu) - p_{0,uac} + 1)C_{ua}^{\mu} + \\ & W_{uc}\big(\tau_{ua}, L_{uac}(\mu), \mu\big) + \\ & \sum_{\forall i \neq u} W_i^*\big(\tau_{ua}, L_{uac}(\mu)\big) \end{aligned} \tag{5}$$

where $p_{0,uac}$ denotes the first, and $p_{L,uac}(\mu)$ the last, task instance, of τ_{ua}, activated within the busy period. They are defined as:

$$p_{0,uac} = -\left\lfloor \frac{J_{ua} + \Phi_{uac}}{T_u} \right\rfloor + 1 \tag{6}$$

and

$$p_{L,uac}(\mu) = \left\lceil \frac{L_{uac}(\mu) - \Phi_{uac}}{T_u} \right\rceil \tag{7}$$

In order to get the worst case response time for τ_{ua}, we need to check the response time for every instance, $p \in p_{0,uac} \cdots p_{L,uac}(\mu)$, in the busy period. Completion time of the p'th instance is given by:

$$\begin{aligned} w_{uac}(p,\mu) = & B_{ua} + (p - p_{0,uac} + 1)C_{ua}^{\mu} \\ & + W_{uc}\big(\tau_{ua}, w_{uac}(p,\mu), \mu\big) + \\ & \sum_{\forall i \neq u} W_i^*\big(\tau_{ua}, w_{uac}(p,\mu)\big) \end{aligned} \tag{8}$$

The corresponding response time (for instance p) is then obtained by subtracting its activation time:

$$R_{uac}(p,\mu) = w_{uac}(p,\mu) - \Phi_{uac} - (p-1)T_u + O_{ua} \tag{9}$$

To obtain the worst case response time, $R_{ua,}(\mu)$, assuming mode μ, we need to consider every candidate critical instant, including τ_{ua} itself, and for each such candidate every possible task instance, p, of τ_{ua}:

$$R_{ua}(\mu) = \max_{\forall c \in hp_u(\tau_{ua}) \cup a} \Big[\max_{p = p_{0,uac},\ldots,p_{L,uac}(\mu)} \big[R_{uac}(p,\mu) \big] \Big] \tag{10}$$

As a last step we have to consider the response time for every possible mode, μ, in order to get the worst case response time, R_{ua}, for τ_{ua} in all possible circumstances:

$$R_{ua} = \max_{\forall \mu \in \Gamma_u} \big[R_{ua}(\mu) \big] \tag{11}$$

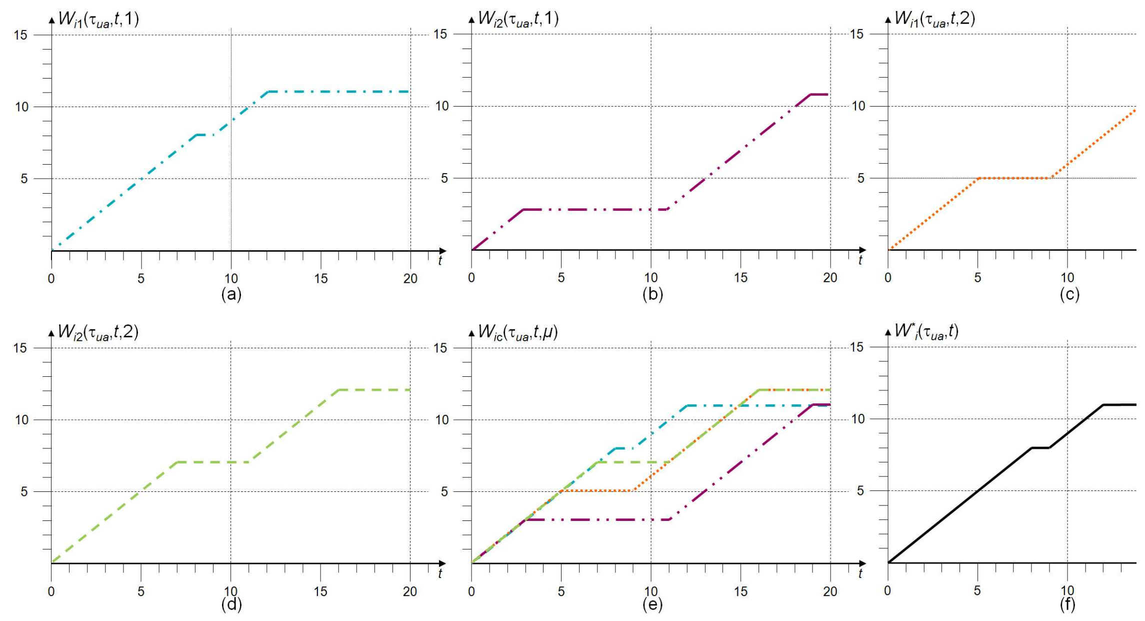

Figure 5. $W_{ic}(\tau_{ua}, t, \mu)$ **and** $W^*_i(\tau_{ua}, t)$ **functions**

4.3. Discussion

If we calculate the response-time for a lower-priority task τ_{ua} with $C_{ua} = 6$ (which is the single tasks of a transaction Γ_u which only has one mode, a very large T_u, $J_{ua} = 0$ and $B_{ua} = 0$) using equation 11 we get $R_{ua} = 18$. Whereas, if we would use the original analysis (with the WCETs from Fig. 1) we would get $R_{ua} = 36$. The main reason for the decreased response-times is the decreased system-load that results from using the modes. In our example transaction the utilization decreases from 75% (15/20) to 60% (12/20) when the modes are introduced.

The introduction of modes increases the complexity of the analysis since we have to consider $|\Gamma_i|s$, recall that s is the number of modes, approximation functions for each transaction instead only $|\Gamma_i|$ approximation functions in the original analysis. These approximations are evaluated many times during evaluation of equations 5 and 8 (which are evaluated by fix-point iteration). However, this increase in complexity can be removed using the techniques presented in [16], which allow an arbitrary number of approximation functions to be represented using a single table.

The analysis presented in this paper utilizes the strong expressiveness of the original analysis. The main changes to the original analysis are in equations 3 and 4 and the introduction of equations 11. Equations 5 to 10 remain essentially unchanged from the original analysis.

5. Conclusion and future work

We have introduced the concept of modes for the transactional task model in order to capture execution time dependencies among tasks. By being able to express execution time dependencies between tasks, previously assumed independent, one can obtain more accurate (tighter) response times for task and thus increase the schedulability of the system. We have shown how RTA for the transactional task model can be extended to utilize the notion of modes.

Being able to express more complex task interdependencies we hope that schedulability analysis techniques will be a more viable option for complex industrial applications since the overestimation of too simplified assumptions is reduced.

As a future work we plan to implement the analysis in the Rubus ICE [8] development environment. We will also implement support to identify execution-time dependencies using the build-in monitoring capabilities of the Rubus OS. Using these implementations we will investigate real systems, to quantify how much resources can be freed using our new analysis technique. Many current systems built with the Rubus tool-chain has utilization close to 100% during scheduling analysis; which to some extent depend on overestimations in the analysis [9].

Acknowledgement

This work is supported by Swedish Knowledge Foundation (KKS) within the project EEMDEF, the Swedish Research Council (VR) within project TiPCES, and the Strategic Research Foundation (SSF) with the centre PROGRESS.

References

[1] N. Audsley, A. Burns, R. Davis, K. Tindell, and A. Wellings. Fixed Priority Pre-Emptive Scheduling: An Historical Perspective. *Real-Time Systems*, 8(2/3):173–198, 1995.

[2] N. Audsley, A. Burns, K. Tindell, M. Richardson, and A. Wellings. Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling. *Software Engineering Journal*, 8(5):284–292, 1993.

[3] A. Burns, K. Tindell, and A. Wellings. Effective Analysis for Engineering Real-Time Fixed Priority Schedulers. *IEEE Transactions on Software Engineering*, 22(5):475–480, May 1995.

[4] I. E. Commission. IEC 61508 - Functional safety of electrical/electronic/programmable electronic safety-related systems.

[5] A. Ermedahl, H. Hansson, and M. Sjödin. Response-Time Guarantees in ATM Networks. In *Proc. 18th IEEE Real-Time Systems Symposium (RTSS)*, pages 274–284. IEEE Computer Society Press, December 1997.

[6] M. K. et al. A Practitioners Handbook for RMA.

[7] J. Fredriksson, K. Sandström, and M. Åkerholm. Optimizing Resource Usage in Component-Based Real-Time Systems. In *8th International Symposium on Component-based Software Engineering (CBSE8)*, May 2005.

[8] K. Hänninen, J. Mäki-Turja, S. Sandberg, J. Lundbäck, M. Lindberg, M. Nolin, and K.-L. Lundbäck. Framework for Real-Time Analysis in Rubus-ICE. In *13th IEEE International Conference on Emerging Technologies and Factory Automation*, September 2008.

[9] K. Hänninen and T. Riutta. Optimal Design. Master's thesis, Mälardalens Högskola, Dept of Computer Science and Engineering, 2003.

[10] M. Joseph and P. Pandya. Finding Response Times in a Real-Time System. *The Computer Journal*, 29(5):390–395, 1986.

[11] D. Katcher, H. Arakawa, and J. Strosnider. Engineering and analysis of fixed priority schedulers. *IEEE Transactions on Software Engineering*, 19(9):920–934, September 1993.

[12] J. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Proc. 11th IEEE Real-Time Systems Symposium (RTSS)*, pages 201–212, December 1990.

[13] R. Lencevicius and A. Ran. Can Fixed Priority Scheduling Work in Practice? In *Proc. 24th IEEE Real-Time Systems Symposium (RTSS)*, page 358, December 2003.

[14] C. Liu and J. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM*, 20(1):46–61, 1973.

[15] J. Mäki-Turja and M. Nolin. Tighter Response-Times for Tasks with Offsets. In *Proc. of the 10th International conference on Real-Time Computing Systems and Applications (RTCSA'04)*, August 2004.

[16] J. Mäki-Turja and M. Nolin. Efficient Implementation of Tight Response-Times for Tasks with Offsets. *Real-Time Systems Journal*, February 2008.

[17] M. Nolin and J. Mäki-Turja. Achieving industrial strength timing predictions of embedded system behavior. In *The 2008 International Conference on Embedded Systems and Applications*, July 2008.

[18] J. Palencia Gutiérrez and M. Gonzáles Harbour. Schedulability Analysis for Tasks with Static and Dynamic Offsets. In *Proc. 19th IEEE Real-Time Systems Symposium (RTSS)*, December 1998.

[19] S. Punnekkat. *Schedulability Analysis for Fault Tolerant Real-time Systems*. PhD thesis, University of York, June 1997.

[20] L. Sha, T. Abdelzaher, K.-E. Årzén, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A. K. Mok. Real Time Scheduling Theory: A Historical Perspective. *Real-Time Systems*, 28(2/3):101–155, 2004.

[21] L. Sha, R. Rajkumar, and J. Lehoczky. Task scheduling in distributed real-time systems. In *IEEE Industrial Electronics Conference*, 1987.

[22] L. Sha, R. Rajkumar, and J. Lehoczky. Priority Inheritance Protocols: an Approach to Real Time Synchronization . *IEEE Transactions on Computers*, 39(9):1175–1185, September 1990.

[23] H. Thane and H. Hansson. Towards systematic testing of distributed real-time systems. In *Proc. 20th IEEE Real-Time Systems Symposium (RTSS)*, pages 360–369, December 1999.

[24] K. Tindell. Using Offset Information to Analyse Static Priority Pre-emptively Scheduled Task Sets. Technical Report YCS-182, Dept. of Computer Science, University of York, England, 1992.

[25] K. Tindell and J. Clark. Holistic Schedulability Analysis For Distributed Hard Real-Time Systems. Technical Report YCS197, Real-Time Systems Research Group, Department of Computer Science, University of York, November 1994. URL ftp://ftp.cs.york.ac.uk/pub/-realtime/papers/YCS197.ps.Z.

[26] K. Tindell, H. Hansson, and A. Wellings. Analysing Real-Time Communications: Controller Area Network (CAN). In *Proc. 15th IEEE Real-Time Systems Symposium (RTSS)*, pages 259–263. IEEE, IEEE Computer Society Press, December 1994.

[27] A. Wall, J. Andersson, and C. Norström. Probabilistic Simulation-based Analysis of Complex Real-Times Systems. In *6th IEEE International Symposium on Object-oriented Real-time distributed Computing*, Hakodate, Hokkaido, Japan, May 2003.