HAL
open science

Towards EDF Schedulability Analysis of an Extended Timing Definition Language

T. Kloda, B. d'Ausbourg, L. Santinelli

To cite this version:

T. Kloda, B. d'Ausbourg, L. Santinelli. Towards EDF Schedulability Analysis of an Extended Timing Definition Language. 6th Workshop on Adaptive and Reconfigurable Embedded Systems (APRES 2014), Apr 2014, BERLIN, Germany. hal-01082722

HAL Id: hal-01082722

https://hal.science/hal-01082722v1

Submitted on 14 Nov 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Towards EDF Schedulability Analysis of an Extended Timing Definition Language

Tomasz Kloda Bruno d'Ausbourg Luca Santinelli
ONERA Toulouse, name.surname@onera.fr

***Abstract*—In a time-triggered system activities like releasing of tasks, mode switches, sensor readings are all initiated at predetermined points in real-time. This paper proposes an extension of a time-triggered compositional framework and presents, based on the widely-applied methods, a condition for its schedulability. The pessimism of this condition is then discussed and the new challenges in the compositional analysis of time-triggered systems are raised.**

I. Introduction

The development of embedded software is a highly platform dependent process. The main difficulty lies in both formulating the functional specification of the system and correctly determining its temporal behavior. While the former is facilitated by high-level programming languages which abstract from many hardware aspects, getting the expected temporal characteristic of the system involves usually much more efforts due to the implementation of scheduling policies, synchronization and inter-processes communication protocols.

To answer the problem of managing efficiently these two crucial for the correctness of the system aspects, time-triggered languages were devised for embedded programming. These languages clearly separate the functional part of applications and their timing definition. Applications are specified through two descriptions: their timing definition, expressed in a time-triggered language, and the functional code of tasks, expressed in any programming language (C for example). At the final stage, a dedicated compiler generates, based on the both descriptions, a ready-to-run executable for a selected target-platform. This allows control designers and developers to focus on the control systems aspects instead of the platform (hardware and operating system) without being interested in where, how and when tasks are actually scheduled: this may be on platforms with a single CPU or with many CPUs, on platforms with a preemptive priority scheduling or not.

The basic functional units of time-triggered languages are tasks that periodically execute some piece of code. Several concurrent tasks may be grouped into a mode. Tasks are invoked within the mode at declared frequencies and can be removed or added when switching from one mode to another. They communicate between them as well as with sensors and actuators by means of ports.

Time-triggered languages provide a programming abstraction which was firstly introduced within the *Giotto* programming language [12]. Giotto assigns to each task a *Logical Execution Time LET* [13] which defines the precise instants when the task exchanges data with the other tasks and the environment. Task is invoked and reads its input ports at the start of its LET interval, then performs a computation whose results are made available on its output ports exactly at the end of the LET. Figure 1 shows the difference between the logical and physical execution of a task: the observable temporal behavior of a task is independent from the platform related factors such as processing speed, scheduling policy and communication protocols.

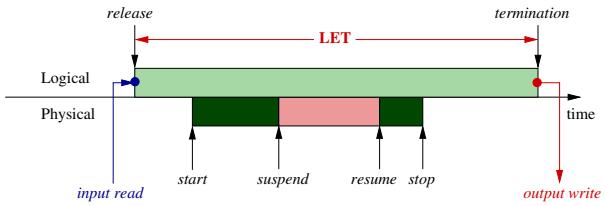

Fig. 1: Logical Execution Time and its possible physical execution.

Timing Definition Language (*TDL*) [9], [10], [23], a successor of Giotto, extends these concepts by allowing a decomposition of large real-time systems into executing concurrently modules (components). Each TDL module runs in one mode at a time and can switch the modes independently of other modules.

A. Problem Statement

While enabling timing and value predictability of programs, Giotto and TDL can be further extended towards a more flexible and more realistic application modeling. In both frameworks, the length of task LET and its period are equal. Introducing a task model with an initial offset and the LET of a task terminating before the end of its period is what we call the new framework *Extended Timing Definition Language* (E-TDL) which is presented in Section II.

With E-TDL it is necessary to guarantee safe execution during multimodal operation inside every module. In section I-C are cited some methods [24], [11] that can be adapted to our case in order to provide a schedulability condition. They address mainly event-triggered systems where the execution of some activity is a consequence of the occurrence of an event whose arrival cannot be predefined beforehand. Hence, it can be supposed that in all modules the events which produce the biggest demand may arrive at the same time.

The paper was presented at APRES 2014. Copyright retained by the authors.

Time-triggered systems observe the state of the controlled object only at specified time instants and initiate appropriate activities only at these instants. In that case, time instants of tasks activations are precisely defined and it is known whether tasks are launched simultaneously or not. Consequently it may be avoided to consider the synchronous case in analysis when it is well known that this case never occurs.

Compositional analysis of such systems should be able to exhibit the relations between the start of the intervals in different modules and compare only these that can actually start at the same time. Therefore, the analyses that are well-suited for event-triggered systems can be overestimating in the time-triggered context.

B. Contribution

A short description of E-TDL is presented and a sufficient condition for the feasibility of an E-TDL system running on a single CPU under the EDF scheduling policy [14] is proposed. The condition is based on the event-triggered paradigm what results in some degree of pessimism. An example is used to point our future works in a direction that permits to reduce this pessimism by taking into account the time-triggered aspect of the system.

C. Related Work

Many different protocols and methodologies attempting to ensure the schedulability of a system across mode switches have been proposed in the literature. Protocols known as *synchronous* [25], [4], [20] do not release new tasks until all old mode tasks are completed. On the contrary, *asynchronous* protocols, both for *Fixed Priority* [26], [18], [20] and *EDF* [3], [7], define that during transition phase the last activations of old mode tasks and new mode tasks can be executed simultaneously. In TDL and E-TDL, systems can be composed of many modules and each module can undergo mode transitions that are defined only in its local scope. For systems designed in a compositional manner, an approach based on the *real-time calculus* (*RTC*) [24] is developed in the work of Stoimenov et al. [22]. Fisher [11] proposed a schedulability test for *EDF* where the allocation of the processing resources for each subsystem is represented by an *explicit-deadline periodic* resource model [8] and a *sporadic task model* [16] is chosen. Servers can also dynamically adapt their parameters of resource reservation. The feasibility under multi-moded resource reservation was studied by Santinelli et al. in [21].

Concerning the schedulability analysis of time-triggered systems, some of the most significant contributions were made notably by Farcas [9] for TDL and Martinek for the *Giotto in Ada* [15] framework. The latter extends a Giotto's task model with the notion of deadline but, since Giotto is not a compositional framework, these results cannot be applied in this instance.

Closely related to our work is *RTComposer* framework proposed by Alur and Weiss [2]. Scheduling requirement of each component is described by a finite state automaton [1]. Compositional analysis consists in verifying that the product of the automata of all the components, intersected with constraints imposed by the platform, is not empty. In our approach, we opt for *processor demand criterion* [5], [6], a technique that is well-suited and widely applied for EDF schedulability analysis.

II. Extended TDL Model

An E-TDL periodic task $\tau_i = (\Phi_i, C_i, LET_i, T_i)$ is characterized by an offset Φ_i, a worst-case execution time C_i, a Logical Execution Time LET_i (that gives its relative deadline D_i) and a period T_i.

The offset Φ_i is restricted to be smaller than the period ($0 \leq \Phi_i < T_i$) and LET_i has to be large enough that task could be executed within it ($C_i \leq LET_i \leq T_i - \Phi_i$). Moreover, E-TDL semantics do not allow the $j-th$ instance ($j \in \mathbb{N}_+$) of task τ_i started in the time interval $[(j-1)T_i, jT_i]$ to be finished after time jT_i. Taking into account the above mentioned constraints, an E-TDL task τ_i may be implemented by a periodic real-time task whose releases and deadlines are expressed as follows:

$$r_{i,j} = \Phi_i + (j-1)T_i \leq jT_i, \tag{1}$$

$$d_{i,j} = \Phi_i + LET_i + (j-1)T_i \leq jT_i. \tag{2}$$

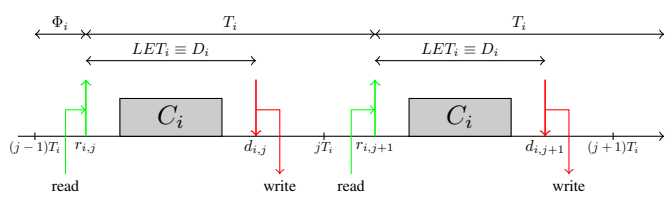

Fig. 2: E-TDL task model.

The LET semantics imposes that a task reads periodically its inputs exactly when it is released and delivers its outputs exactly at the end of its LET interval.

An E-TDL system consists of multiple *modules* running on the same node. All modules of the system, referred hereafter as $Modules$, run concurrently sharing common processing resources. Each module $M_j \in Modules$, from a scheduling point of view, is considered independent from the others.

At a time module M_j executes one of its *modes* from the set $Modes[M_j]$. One of them is the initial *start mode*. A *mode* is invoked and performs appropriate actions when the environment is in some specific state that should by handled by this particular mode [17]. Each mode m_k executes periodically, within its *mode period* $\pi[m_k]$, a set of E-TDL tasks $\tau[m_k]$. A mode period is restrained to be a common multiple of the periods of all tasks in the mode.

$$\pi[m_k] \stackrel{\text{def}}{=} n\, H(m_k) \tag{3}$$

where $H(m_k)$ is a *hyperperiod* of all tasks in the mode m_k,

$$H(m_k) \stackrel{\text{def}}{=} lcm\{T_i \mid \tau_i \in \tau[m_k]\} \tag{4}$$

$$n \in \mathbb{N}_+$$

The amount of time elapsed since the start of the last period of the mode m_k is named *mode time* δ_k and is set to 0 every time a mode period is started. As mode time δ_k evolves in the mode period $\pi[m_k]$ ($0 \leq \delta_k \leq \pi[m_k]$), appropriate actions, like releasing or terminating tasks, are triggered. When the condition change in the environment or inside the system is observed, the current operating mode stops its activities and a new mode is entered instantaneously with a mode time set to zero. These transitions are described in E-TDL by *mode switches* which define precisely at which exact mode time mode switch conditions are evaluated and transitions can take place if the mode switch conditions are satisfied. During execution of mode m_k the conditions to switch from mode m_k to mode m_{k+1} are checked periodically every *mode-switch period* $T_{sw}(m_k, m_{k+1})$. The instants δ_{ms}, at which this check occurs and which are multiple of $T_{sw}(m_k, m_{k+1})$, are named *mode switch instants*. Mode that can be activated or reactivated from the mode time δ_{ms} in the mode m_k belongs to the set $next_modes(m_k, \delta_{ms})$:

$$next_modes(m_k, \delta_{ms}) = \{m_{k+1} \mid \exists T_{sw}(m_k, m_{k+1}) : \delta_{ms} \bmod T_{sw}(m_k, m_{k+1}) = 0\} \cup \{m_k \mid \delta_{ms} = \pi[m_k]\}. \quad (5)$$

Every mode switch period $T_{sw}(m_k, m_{k+1})$ is restricted to be a common multiple of all the task periods of the mode in which it was declared:

$$T_{sw}(m_k, m_{k+1}) \bmod H(m_k) = 0 \wedge \exists n \in \mathbb{N}_+,\ n\, T_{sw}(m_k, m_{k+1}) = \pi[m_k]. \quad (6)$$

Such a choice of mode switch instants entails that a mode switch may occur only at time instants when no current mode task is running. The new mode is activated without any delay at zero mode time instant ($\delta_k = 0$). Mode switches are local within a module as a mode switch can occur in a particular module while tasks of all the other modules are running.

III. Schedulability analysis for E-TDL framework

The above presented model imposes on each module of the system a set of possible execution patterns that can be observed during its temporal evolution. In what follows, the behavior of every module is characterized and the processing resources the module demands over time are quantified. This permits to provide a sufficient condition for the schedulability of the whole system under mode transitions.

A. Execution Trace of an E-TDL Module

In [9] Farcas proposed a *mode-switch graph* which allows to exhibit all the modes of a given module and the transitions it can undergo. Modes are represented by the vertices and transitions by the edges of the graph. An edge is traced from the vertex denoting a source mode to the vertex denoting a destination mode if a mode switch between these two modes is defined. The mode switch period labels the edge.

Definition 1 (Mode-Switch Graph). *For a module $M_j \in Modules$ its mode-switch graph is defined by the set $Modes[M_j]$ and the set of transitions $\{(m_k, m_{k+1}, T_{sw}(m_k, m_{k+1})) | m_k, m_{k+1} \in Modes[M_j], \exists \delta : m_{k+1} \in next_modes(m_k, \delta)\}$.*

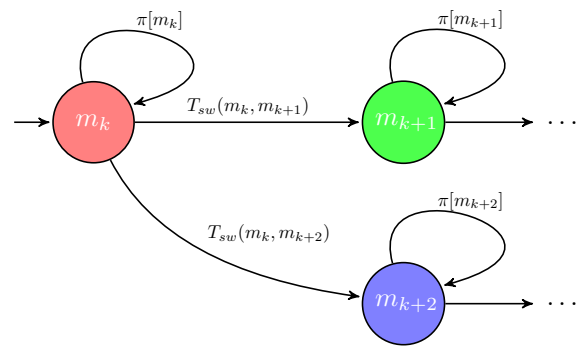

Fig. 3: Example of a mode-switch graph.

Over the course of its execution, a module can remain in the current mode or switch to one of its successive modes. The behavior of a module can be described by a walk that travels through its mode-switch graph, visiting modes for any time that is multiple of one of its mode switch periods and jumping to another modes at the time instants of the mode switch evaluation.

Definition 2 (Walk in a Mode-Switch Graph). *A walk $w = ((m_1, \mu_1), \ldots, (m_n, \mu_n))$ of an E-TDL mode-switch graph is a sequence of n pairs where $m_1, \ldots, m_n$ are the subsequent modes in the mode-switch graph and μ_k is defined as the number of:*

- *(for $k < n$: $\mu_k \in \mathbb{N}_+$) times the condition of mode switch to m_{k+1} has been checked in m_k before entering m_{k+1},*
- *(for $k = n$: $\mu_k \in \mathbb{N}_0$) full periods $\pi[m_k]$ spent in m_k.*

The first mode in the walk w is designated as $head(w)$ and the last mode as $tail(w)$. The length of a walk $w = ((m_1, \mu_1), \ldots, (m_n, \mu_n))$ can be expressed as:

$$|w| = \sum_{k=1}^{n} \mu_k T_{sw}(m_k, m_{k+1}) \quad (7)$$

To simplify notation, for $k = n$, as there is no next mode in the walk w, we use term $T_{sw}(m_k, m_{k+1})$ interchangeably with $\pi[m_k]$.

The notion of walk permits to describe the execution traces that lie between mode switch instant δ_{ms} when the module is in $head(w)$ and mode instant 0 when the module is in $tail(w)$. Every trace of a module execution can be seen as a walk w preceded and followed by two intervals. Both of them cover an execution pattern that can be observed within a mode single run (between its mode time 0 and its period end). The interval that precedes w starts at time t_s and finishes at the time when the module enters mode $head(w)$. The interval that is observed from the end of w, starts when module is in mode $tail(w)$ at

mode time 0 and finishes at time t_f. Figure 4 illustrates the proposed interval $[t_s, t_f]$ decomposition.

Definition 3 (Module's Execution Trace). *An execution trace of an E-TDL module $M_j \in Modules$ over a time interval $[t_s, t_f]$ is denoted by $\sigma_{M_j} = (m_s, \delta_s, \delta'_s, w, m_f, \delta_f)$ where*

- $m_s \in Modes[M_j]$ *is the mode executing before walk* w
- $\delta_s : 0 < \delta_s \leq \pi[m_s]$ *is the mode time related to* t_s
- $\delta'_s : \delta_s \leq \delta'_s \leq \pi[m_s]$ *is the mode time of mode* m_s
- w *is a walk in the mode-switch graph of module* M_j
- $m_f \in Modes[M_j]$ *is the mode executing after walk* w
- $\delta_f : 0 \leq \delta_f < \pi[m_f]$ *is the mode time related to* t_f,

and the following conditions are fulfilled:

- $head(w) \in next_modes(m_s, \delta'_s)$
- $if\ tail(w) \neq m_f :$
  $m_f \in next_modes(tail(w), \pi[tail(w)])$ *and* $tail(w)$ *is executed at least one time at the end of* w
- $t_f - t_s = \delta'_s - \delta_s + |w| + \delta_f$

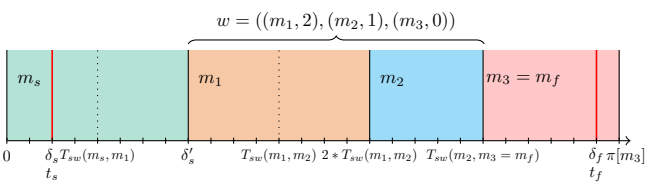

Fig. 4: Example of a module's execution trace

Short intervals, within which no more than one mode's period is executed, can be represented, in accordance with the above definition, as $\sigma_{M_j} = (m_s, \delta_s, \delta'_s, \emptyset, \emptyset, 0)$.

B. Workload Characterization of an E-TDL Module

To determine the schedulability of an E-TDL system, the workload generated by each module should be precisely quantified. An E-TDL execution trace is the combination of two types of intervals. Before and after its walk no more than one mode period is executed. The walk itself constitutes a sequence of intervals that span over some number of mode switch evaluation periods. Based on the concept of demand function [5], the next two definitions evaluate the workloads in such intervals.

Definition 4 (Demand Function for a Walk). *For a given walk w its demand function $df(w)$ can be calculated as:*

$$df(w) \stackrel{\text{def}}{=} \sum_{(m_k, \mu_k) \in w} \mu_k T_{sw}(m_k, m_{k+1}) U(m_k) \tag{8}$$

where $U(m_k)$ is the processor utilization factor of mode m_k:

$$U(m_k) \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau[m_k]} \frac{C_i}{T_i}. \tag{9}$$

Definition 5 (Demand Function within a Mode). *Let δ_k and δ'_k be the mode times of the mode m_k such that $0 \leq \delta_k < \delta'_k \leq \pi[m_k]$. The demand function $df(m_k, \delta_k, \delta'_k)$ within a mode m_k is the cumulative execution time required by all the task instances in m_k having its releases and deadlines between δ_k and δ'_k.*

$$df(m_k, \delta_k, \delta'_k) \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau[m_k]} \left(\left\lfloor \frac{\delta'_k - \Phi_i - LET_i}{T_i} \right\rfloor - \left\lceil \frac{\delta_k - \Phi_i}{T_i} \right\rceil + 1 \right)_0 C_i \tag{10}$$

The foregoing definitions can be applied to express the workload generated by an E-TDL module execution trace σ_{M_j} (see Definition 3) that has been observed in a time interval $[t_s, t_f]$.

Definition 6 (Demand Function). *If an E-TDL execution trace $\sigma_{M_j} = (m_s, \delta_s, \delta'_s, w, m_f, \delta_f)$ is observed in the time interval $[t_s, t_f]$ in module $M_j \in Modules$, the total cumulative demand $df_{M_j}(t_s, t_f)$, required by M_j over this interval is given as:*

$$df_{M_j}(t_s, t_f) \stackrel{\text{def}}{=} df(m_s, \delta_s, \delta'_s) + df(w) + df(m_f, 0, \delta_f) \tag{11}$$

Depending on the past stimuli, different execution traces can be observed in the time interval $[t_s, t_f]$. Let $\Sigma_{M_j}(t_s, t_f)$ be the set of all execution traces that can potentially appear between the time instants t_s and t_f.

Example 1. *Consider module M_1 with two modes $Modes[M_1] = \{m_1, m_2\}$ characterized by the same mode period $\pi[m_1] = \pi[m_2] = \pi$. Module M_1 can go from mode m_1 to m_2 if at the period end of the former some sensor reading is above a particular value. Given an interval $[t_s, t_f]$ where $t_s = \pi$ and $t_f = 2\pi$, then the execution traces which can be observed within it lie in the set $\Sigma_{M_j}(t_s, t_f) = \{\sigma_{M_j} = (m_1, \pi, \pi, w, m_f, 0) | w \in \{(m_1, 1), (m_2, 1)\},\ m_f = head(w)\}$.*

If $Maxdf_{M_j}(t_s, t_f)$ designates the largest value of demand functions $df_{M_j}(t_s, t_f)$ for execution traces within the set $\Sigma_{M_j}(t_s, t_f)$, then the highest demand for processing time of module M_j over any time interval is given by the following definition.

Definition 7 (Demand Bound Function). *The demand bound function $dbf_{M_j}(\Delta)$ denotes the maximal cumulative requirement of computation for module $M_j \in Modules$ during any time interval $[t_s, t_s + \Delta]$ that can be generated by some E-TDL execution trace $\sigma_{M_j} \in \Sigma_{M_j}(t_s, t_s + \Delta)$ where Δ is a non-negative integer.*

$$dbf_{M_j}(\Delta) \stackrel{\text{def}}{=} \max_{t_s} \{Maxdf_{M_j}(t_s, t_s + \Delta)\} \tag{12}$$

C. Schedulability of an E-TDL System

Under *EDF*, the schedulability analysis of an asynchronous task set where deadlines are less than periods is performed using the *processor demand criterion* [5], [6]. The schedulability of a task set can be proved if the cumulative demand of the computation made by its tasks in any interval is never larger than the length of this interval. The following theorem adapts this reasoning to the E-TDL framework. It verifies if, for any time interval, the total amount of processing time requested

by individual modules to complete their execution traces does not exceed the interval's length.

Theorem 1 (**Schedulability of an E-TDL System**). *Let be an E-TDL system defined by a set of modules $Modules$ such that:*

$$\sum_{M_j \in Modules} \max_{m_k \in Modes[M_j]} \{U(m_k)\} \quad \leq 1 \tag{13}$$

If for every time interval $\Delta > 0$:

$$\sum_{M_j \in Modules} dbf_{M_j}(\Delta) \quad \leq \Delta \tag{14}$$

then the E-TDL system is schedulable under EDF on one processor.

The above theorem gives only a sufficient condition for schedulability of an E-TDL system.

Example 2. *Given two positive-integers π and ε such that $\varepsilon < \pi$, and an E-TDL system composed of modules M_1 and M_2. Module M_1 can run in one of two modes $Modes[M_1] = \{m_1, m_1'\}$ both having the same mode period: $\pi[m_1] = \pi[m_1'] = \pi$ at the end of which each one can jump to the other mode. Module M_2 has only one mode $Modes[M_2] = \{m_2\}$ with the mode period $\pi[m_2] = 2\pi$. All the modes are single-task and their task sets are as follows: $\tau[m_1] = \{\tau_1 = (0, \varepsilon, \varepsilon, \pi)\}$, $\tau[m_1'] = \{\tau_1' = (\varepsilon, \pi - \varepsilon, \pi - \varepsilon, \pi)\}$, $\tau[m_2] = \{\tau_2 = (0, 2(\pi - \varepsilon), 2\pi, 2\pi)\}$.*

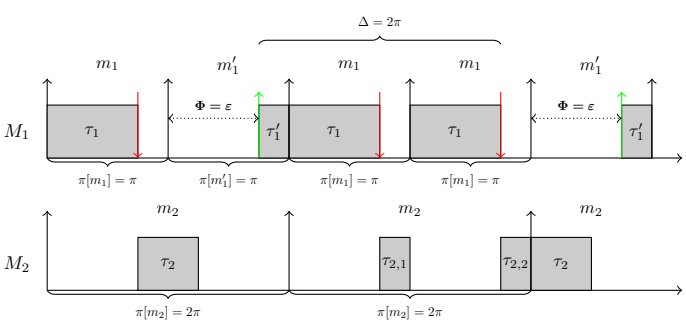

Fig. 5: A sample run of the E-TDL system from Example 2.

It can be seen from the above figure that the system is schedulable. However, for the interval $\Delta = 2\pi$ the condition of Theorem 1 (Equation 14) is not satisfied. The demand bound functions over an interval of this length for the individual modules are:

- $dbf_{M_1}(\Delta) = \varepsilon + \pi$ *for* $\sigma_{M_1} = (m_1', \varepsilon, \pi, (m_1, 1), m_1, \varepsilon)$
- $dbf_{M_2}(\Delta) = 2(\pi - \varepsilon)$ *for* $\sigma_{M_2} = (m_2, 0, 2\pi, \emptyset, \emptyset, 0)$

The cumulative demand is $dbf_{M_1}(\Delta) + dbf_{M_2}(\Delta) = 3\pi - \varepsilon > 2\pi$ as $0 < \varepsilon < \pi$. The summed up demands are associated with intervals that will never completely overlap due to the timing control of tasks in the time-triggered paradigm. Since these demands will never be simultaneous, adding them overestimates the cumulative demand.

IV. Conclusion and Future Work

In this work we proposed an extension to TDL for which we applied a scheduling analysis based on the processor demand criterion. In the future, we would like to reduce the pessimism of the solution from Theorem 1 by taking into account the time-triggered character of the system. The mode-switch graph should be examined more carefully to find out what are the allowable instants of mode starts and then, similarly to the feasibility test for asynchronous task sets proposed by Pellizzoni [19], construct the global schedules that are made up of execution traces occurring at the same time in the distinct modules. Furthermore, a maximal length of the interval on which the schedulability condition is checked should be estimated.

References

[1] R. Alur and G. Weiss. Regular Specifications of Resource Requirements for Embedded Control Software. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 159–168, 2008.

[2] R. Alur and G. Weiss. RTComposer: A framework for Real-Time Components with Scheduling Interfaces. In *Proceedings of the 8th ACM International Conference on Embedded Software*, EMSOFT '08, pages 159–168, New York, NY, USA, 2008. ACM.

[3] B. Andersson. Uniprocessor EDF Scheduling with Mode Change. In *Proceedings of the 12th International Conference on Principles of Distributed Systems*, OPODIS '08, pages 572–577, Berlin, Heidelberg, 2008. Springer-Verlag

[4] C. M. Bailey. *Hard real-time operating system kernel. Investigation of mode change. Task 14 Deliverable on ESTSEC Contract 9198/90/NL/SF* British Aerospace Systems Ltd., 1993

[5] S. K. Baruah, L. E. Rosier, and R. R. Howell. Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic, Real-Time Tasks on One Processor. *Real-Time Syst.*, 2(4):301–324, Oct. 1990.

[6] G. C. Buttazzo. *Hard Real-time Computing Systems: Predictable Scheduling Algorithms And Applications (Real-Time Systems Series)*. Springer-Verlag TELOS, Santa Clara, CA, USA, 2004.

[7] G. C. Buttazzo, L. Abeni, and S. S. S. Anna. Elastic Task Model For Adaptive Rate Control. In *IEEE Real-Time Systems Symposium*, pages 286–295, 1998

[8] A. Easwaran, M. Anand, and I. Lee. Compositional Analysis Framework Using EDP Resource Models. In *RTSS*, pages 129–138, 2007.

[9] E. Farcas. *Scheduling Multi-Mode Real-Time Distributed Components*. PhD Thesis, Department of Computer Science, University of Salzburg, July 2006.

[10] E. Farcas, C. Farcas, W. Pree, and J. Templ. Transparent Distribution of Real-Time Components Based on Logical Execution Time. In *LCTES*, pages 31–39, 2005.

[11] N. Fisher and M. Ahmed. Tractable Real-Time Schedulability Analysis for Mode Changes under Temporal Isolation. In *ESTImedia*, pages 130–139, 2011

[12] T. A. Henzinger, B. Horowitz, and C. M. Kirsch. Giotto: A Time-Triggered Language For Embedded Programming. In *EMSOFT*, pages 166–184, 2001.

[13] C. M. Kirsch and A. Sokolova. The Logical Execution Time Paradigm. In *Advances in Real-Time Systems*, pages 103–120, 2012.

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

[15] N. F. Martinek and W. Pohlmann. Mode Switching in GIA – An ADA Based Real-Time Framework. Department of Scientific Computing, University of Salzburg.

[16] A. K. Mok. Fundamental design problems of distributed systems for the hard-real-time environment. PhD Thesis, Laboratory for Computer Science, Massachusetts Institute of Technology, USA, 1983.

[17] P. Pedro. *Schedulability of Mode Changes in Flexible Real-Time Distributed Systems*. PhD Thesis, Department of Computer Science, University of York, September 1999.

[18] P. Pedro and A. Burns. Schedulability Analysis for Mode Changes in Flexible Real-Time Systems. In *ECRTS*, pages 172–179, 1998.

[19] R. Pellizzoni and G. Lipari. Feasibility Analysis of Real-Time Periodic Tasks with Offsets. *Real-Time Systems*, 30(1-2):105–128, 2005.

[20] J. Real and A. Crespo. *Mode Change Protocols for Real-Time Systems: A Survey and a New Proposal*. Real-Time Syst., 26(2) :161–197, March 2004

[21] L. Santinelli, G. C. Buttazzo, and E. Bini. Multi-Moded Resource Reservations. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 37–46, 2011.

[22] N. Stoimenov, S. Perathoner, and L. Thiele. Reliable Mode Changes in Real-Time Systems with Fixed Priority or EDF Scheduling. In *Proceedings of Design, Automation and Test in Europe, 2009* , pages 99–104, Apr 2009. IEEE.

[23] J. Templ. Timing Definition Language (TDL) Specification 1.5 Technical Report T024, Department of Computer Science, University of Salzburg, Austria, October 2008.

[24] L. Thiele, S. Chakraborty, and M. Naedele. *Real-Time Calculus for Scheduling Hard Real-Time Systems*. In *The 27th Annual International Symposium on Computer Architecture(ISCA)*, volume 4, pages 101 –104 vol.4, 2000

[25] K. Tindell and A. Alonso. *A very simple protocol for mode changes in priority preemptive systems*. Technical report, Universidad Politécnica de Madrid, 1996

[26] K. Tindell, A. Burns, and A. Wellings. *Mode changes in priority preemptively scheduled systems*. In Proceedings of the Real Time Systems Symposium, pages 100–109, 1992