3

Schedulability Analysis of Mode Changes with Arbitrary Deadlines

Paulo Martins[1,2], I. G. Hidalgo[2], M. A. Carvalho[2], A. de Angelis[2], V. Timóteo[2], R. Moraes[2], E. Ursini[2] and Udo Fritzke Jr[3]
[1]*Chaminade University*
[2]*Universidade Estadual de Campinas (UNICAMP)*
[3]*PUC Minas*
[1]*USA*
[2,3]*Brazil*

1. Introduction

Modern real-time systems are required to operate in complex applications and dynamically adapt to a wide range of changes in the environment. One strategy that allows the implementation of these systems in complex scenarios is the partitioning of their applications into modes of operation. Flexibility of operation is achieved by having the system execute in several modes of operation and undergo transitions between modes in response to external or internal events. A mode of operation can be seen as a specific configuration of the computational resources that is optimal for the operational phase that the system executes. As conditions in the environment change, resource allocations may become inadequate. Therefore, the system must reconfigure itself through a mode change, reallocating its resources in an efficient manner. A classic example of modes lies in the area of aviation where most aircraft undergo at least three basic modes of operation: take-off, level-flight, and landing modes. Having the system designed with a single mode of operation does not explore the fact that some operations, and thus resource usage, are mutually exclusive. The allocation of all these resources, as if they could be needed at the same time, leads to inefficiency. More importantly, the resulting system is not scalable. The so called "*all-modes-in-one*" system is usually feasible for simple systems with limited functionality.

Flexible modal real-time systems must guarantee by means of schedulability analysis that all tasks complete before their deadlines. Current literature in schedulability analysis for mode changes requires that all task deadlines are less than or equal to the tasks periods (Pedro & Burns, 1998; Real & Crespo, 2004). Allowing task deadlines to be less than task periods is useful for many real-time applications (Tindell et al., 1994). However, in some real-time applications an instance of a task is allowed to arrive before its previous invocation has finished. In such case the task can be delayed until its previous invocation terminates.

This chapter extends the current schedulability analysis associated with mode changes in static priority pre-emptive based scheduling. In particular, it derives analysis that includes

tasks executing across a mode change with deadlines larger than their period (arbitrary deadlines).

The rest of this chapter is organized as follows: section 2 elaborates on the concept of modes. We include a number of current views of modes and provide a working definition. In section 3 we introduce the computational model and assumptions. In section 4 we address previous work on modes in real-time systems. The schedulability analysis of mode changes with arbitrary deadlines is then derived in section 5. Section 6 shows an example of the applicability analysis. Finally, in section 7 we present our concluding remarks.

2. Definition of modes

The discussion on the concept of modes of operation appears more comprehensively in the literature of human-computer interaction and human-automation interaction (aviation and cockpit design). Modes have been commonly associated with the idea of functions, states, behavior, and schedule, depending on the research field. In real-time systems, since most work involving modes focus on the schedulability analysis of mode changes, this discussion has been limited.

The most widespread notion of modes is related to functionality. One example is the work in safety-critical systems by Papadopoulos (1996): a mode is defined by the *"set of active functions that the system is prepared to deliver while operating in that mode"*. As modes represent functions, they may be arranged hierarchically. This is due to the fact that functionality can be refined down to lower level functions, leading to a hierarchical graph of the functional space. Another example is given by Norman (1981): *"Every control system that can perform a variety of functions has modes: the more functions, the more modes."*

In human-computer interaction and interfacing, modes are seen as *special states*. Howe (1997) remarks that *"a mode is a general state, usually used with an adjective describing the state"*. They are states that are extended over time and with some specific activity being carried out. Interface modes are defined by Tesler (1981) as *"a state of the user interface that lasts for a period of time. It is not associated with any particular (display) object and has no role other than to place an interpretation of operator input "*. According to Poller & Garter (1984), modes are the application's interpretation of the user's input. Therefore, a mode change occurs whenever there is a change of the interpretation of the same inputs.

In human-automation interaction (e.g. aviation psychology and cockpit interface design), modes are described as behavior of a certain component of the system, such as the interface. In particular, Degani et al. (1999) describe a mode as *the manner of behavior of a given system*. A system may have multiple mutually exclusive ways of behaving. Each behavior is defined by the system's input, output, and states. A mode change consists of an evolution from one pattern of behavior to another as a function of time. Transitions between modes are triggered by events in the environment: the way such systems behave reflects the transformation of the environment (Degani & Kirlik, 1995).

In the literature of real-time systems modes have been regarded as a collection of tasks, or schedule (e.g. Real (2000); Tindell, Burns & Wellings (1992)). For Fohler (1994) a schedule is a set of processes or messages (literally a *collection of objects*), characterized by its inherent timing

parameters. These objects may have access to a shared resource according to a pre-defined policy. A mode change is a change in the task set, by means of replacement, addition or removal of tasks.

In computer science in general, modes have been described by the executable code or program (downloaded or memory resident) that is active under execution. Therefore, any change in the executable code reflects a change in the operational mode. A mode has also been regarded as the configuration of the system during a particular phase of operation, meaning how different system resources may be physically or logically arranged. For example, as noted by Degani et al. (1999): *"we define a mode as a machine configuration that corresponds to a unique behavior"*. A system may dynamically change its configuration in order to achieve a better performance. A classic example is load balancing in distributed systems, where reconfiguration is achieved through process migration.

From the definitions above, we can summarize the idea of a mode of operation as the system's behavior while executing a given schedule. The notion of behavior is used to address modes when dealing with the higher levels of abstraction of a real-time system, such as the application or the interface design. Function and performance are two fundamental components of the behavior of a real-time system. A real-time system delivers a function with a certain performance attribute attached to it. A change either in the functionality or in the performance of the system characterizes a change in behavior. A change in behavior is noticed by changing the performance level while keeping the functional set, or likewise by changing the functional set while keeping the systems' performance. Taking these remarks into consideration a mode can be comprehensively defined as follows (Martins & Burns, 2008):

"A mode of a real-time system is defined by the behavior of the system, described by a set of allowable functions and their performance attributes, and hence by a single schedule, containing a set of processes and their timing parameters".

Clearly, the definition of a mode depends upon the field of application and the level of abstraction considered. At the lower levels of abstraction of a real-time system, the notion of a single schedule is suitable to define a mode, since we are interested in applying schedulability analysis to guarantee the predictability of the real-time system. Nevertheless, the idea of schedule very often is a too low-level concept, and therefore devoid of meaning when addressing the end user application concerns. Therefore, as we are searching for a comprehensive definition of modes for real-time systems, we also associate our definition of modes with the idea of behavior. The behavior of the system is bounded to a restricted set of operations that it is allowed to perform when it executes in a specific mode. It constrains the actions that the system is not ready to deliver while in that mode. It also constrains the behavior that the system may exhibit. Unlike states, a mode does not limit its variable values directly. It may be regarded as the system's behavior resulting from its execution through a progression of states. Nevertheless, it is not always possible to map a group of states to a particular mode. Whereas in simple systems the modes and states can be more easily associated, in a complex real-time distributed system it may be difficult (and perhaps not necessary) to related both. Therefore it ensues, from the above definition of modes, that a change from a source mode A to a target mode B occurs in response to the need to change

the system's functionality or adjust its performance. We adopt the following definition for a mode change of a uniprocessor system (Martins & Burns, 2008):

"A mode change of a real-time system is defined as a change in the behavior exhibited by the system, accompanied by a change in the scheduling activity of the system."

Many real-time systems in fact run a fixed task set. Processes executed in mutually exclusive modes are treated as if they could run at the same time, and grouped into a single schedule. Consequently, resources have to be allocated as if they could be requested at the same time, although this is an impossible scenario. This approach may be appropriate for simple real-time systems. However, for large and complex real-time systems this may result in significant reduction in efficiency of the system and sub-utilization of system resources. From a software engineering perspective, this approach denies the principles of modularization and separation of concerns, since it allows unrelated modes (and unrelated schedules) of a system to be treated and designed as if they were instead a single mode and schedule, although they may be logically independent, mutually exclusive or both.

3. Computational model and assumptions

We shall consider a set of periodic or sporadic tasks $\tau = \{\tau_1, \tau_2, ...\tau_i, ..\tau_p\}$ per mode. Each task τ_i is characterized by the tuple $S_i = \{T_i, D_i, C_i, P_i\}$, where: 1) T_i and D_i are respectively the period of task τ_i (or, if a sporadic task, the minimum inter-arrival time between successive tasks of the stream i) and the deadline; 2) C_i is the worst-case execution time (WCET) of the task τ_i. This value is deemed to contain the overheads due to context switching. Moreover, the values of C_i, D_i and T_i are such that $C_i < D_i \leq T_i$. In subsection 5.2 we remove the restriction that $D_i \leq T_i$; 3) P_i represents the priority of task τ_i , assigned according to the Deadline Monotonic Scheduling algorithm.

Throughout this chapter, we use the notation $C_{i(O)}$,$C_{i(A)}$ and $C_{i(N)}$ when referring to the worst-case computational time of an old-mode completed task, an aborted task, and a new-mode task, respectively. τ_i denotes a task for which we are finding the worst-case response time (WCRT) and τ_j denotes a higher-priority task. We use the term *steady-state analysis* to refer to the body of schedulability analysis of single-mode systems, where the task set is fixed and there are no mode changes. We also use the notation:

- $\forall\ \tau_{j(O)}\ hp\ \tau_i$: set of old-mode tasks τ_j with priority higher than task τ_i;
- $\forall\ \tau_{j(A)}\ hp\ \tau_i$: set of aborted tasks τ_j with priority higher than task τ_i;
- $\forall\ \tau_{j(N)}\ hp\ \tau_i$: set of new-mode tasks τ_j with priority higher than task τ_i.

The mode-change model is based on the following assumptions:

- Tasks are executed in a uniprocessor system;
- Tasks are not permitted to voluntarily suspend themselves during an invocation (so, for example, tasks are not allowed to execute internal Ada-like delay statements);
- There are fixed task sets before and after the mode change;
- The worst-case response time of a generic task τ_i (WCRT), denoted R_i, is the longest time ever taken by that τ_i from the time it arrives until the time it completes its required computation;

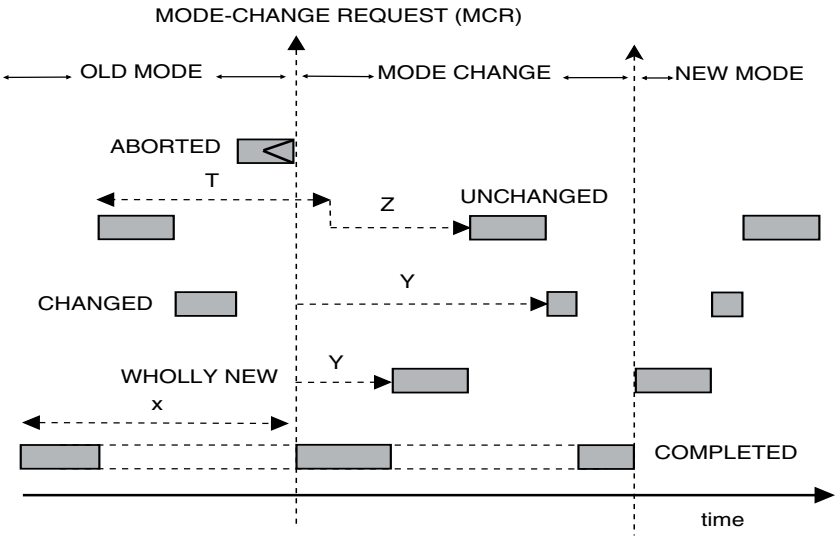

Fig. 1. Mode-Change Model (Pedro, 1999; Real & Crespo, 2004)

- Tasks are scheduled with time offsets during the mode change only. This time phasing between tasks may or may not hold after the mode change.

A *mode-change request (MCR)* is the event that triggers a transition from an old mode of operation to a new one. The window x is the phasing between the MCR and the activation of task τ_i. A MCR may not be preempted by another *MCR*. The mode-change model comprises of five types of tasks (Fig. 1):

- *Old-mode completed tasks*, $\tau_{i(O)}$: These tasks are released in the old mode, i.e. before the arrival of the *MCR*. They need to advance their execution into the transition window in order to finish execution. Old-mode completed tasks cannot be simply aborted as they would leave the system in an inconsistent state. Once they complete during the transition, there are no further releases. They are used to model the behavior of the system in the old mode that is no longer needed in the new mode.
- *Old-mode aborted tasks*, $\tau_{i(A)}$: These tasks are also released prior to the MCR. They need to be immediately discarded after the *MCR* in order to release allocated resources back to the system. The behavior they implement is no longer needed in the new mode of operation.
- *New-mode changed tasks*, $\tau_{i(C)}$: These tasks are released during the transition, with an offset Y from the *MCR*. This class models the behavior that is changed in the new mode. Changed new-mode tasks have a modified timing parameter compared to their corresponding old-mode version, such as changed worst-case execution time (C), period (T), or priority (P).
- *New-mode unchanged tasks*, $\tau_{i(U)}$: These tasks are released during the transition window, with an offset Z, from the end of the period of their corresponding old-mode version. They model the behavior of the application that is not changed across the mode change and in the new mode. Their timing parameters are the same as the preceding old-mode version.

- *Wholly new task*, $\tau_{i(W)}$: These tasks are released during the transition window with an offset Y. They are used to model the behavior that is totally new, i.e. has no equivalent in the old-mode of operation.

With respect to the way tasks are executed across a mode change, they are classified as: 1)Tasks with mode-change periodicity: these tasks are executed across the mode change and maintain their activation pace, and 2) Tasks without mode-change periodicity: these tasks do not preserve their activation pace across a mode change.

The mode-change latency is usually an important performance criterion when dealing with mode changes. We often seek to minimize the latency since during the mode change the system may deliver only partial functionality at the expenses of more critical services. The mode change latency is defined as:

"A window starting with the arrival of the mode-change request (MCR) and ending when the set of new-mode tasks have completed their first execution and the set of old-mode tasks have completed their last execution".

In the following section we review background work on schedulability analysis of mode changes using the fixed-priority preemptive scheduling approach.

4. Background

A number of mode-change protocols have been proposed and classified into synchronous or asynchronous protocols. Synchronous mode-change protocols complete the old-mode tasks before any new task start execution. Synchronous protocols do not require schedulability analysis. On the other hand, asynchronous protocols allow new-mode tasks to begin execution while old-mode tasks are still running. Asynchronous protocols may reduce the mode-change latency, but may have reduced schedulability, since the processing load is larger. These protocols do require schedulability analysis, since old-mode tasks will interfere with the execution of new-mode tasks and vice-versa.

Two types of mode-change protocols can be defined regarding the way unchanged tasks are executed: 1) Protocols with periodicity are protocols where unchanged tasks preserve their activation pace or periodicity. Under these protocols, tasks are executed independently of the mode change in progress; 2) On the other hand, the activation of unchanged tasks may be delayed by protocols without periodicity. Their rate of activation is affected by the transition. The loss of periodicity may be necessary to guarantee the feasibility of the mode change or to preserve data consistency.

The *idle time protocol* (Tindell & Alonso, 1996) delays the execution of any *MCR* until an idle time, where there is no CPU load (activity). It is a simple, synchronous protocol. A mode change task detects the idle time and performs the mode change, by suspending all old mode tasks and activating the new mode ones. The disadvantage is the delay in waiting for the idle time, especially when there may be new-mode tasks with short deadlines waiting to be executed.

The *maximum period offset protocol* (Bailey, 1993) delays all tasks for the time corresponding to the period of the least frequent task in both modes. Being a synchronous protocol, it has

the advantage of simplicity, and the fact that is does not require schedulability analysis. The disadvantage of this protocol is its poor promptness, with an even larger mode-change delay than the idle time protocol.

The *minimum single offset protocol* (without periodicity) (Real, 2000) applies an offset Y to all new mode tasks. The offset is the sum of the worst-case execution time of all old-mode tasks that have been released (but not completed) before the arrival of the *MCR*. This protocol also suffers from poor promptness, but incurs in less mode-change latency compared to the maximum period offset protocol, since all the old-mode tasks execute only once.

The *minimum single offset protocol* (with periodicity) (Real, 2000) similarly applies an offset Y to all new mode tasks. The offset is large enough to accommodate the old mode tasks and all unchanged tasks that need to preserve their periodicity. The disadvantage of this protocol is poor promptness, which is worse than the previous protocol. The protocol is also synchronous and dispenses schedulability analysis.

In the *asynchronous mode-change protocol with periodicity* presented by Tindell, Burns & Wellings (1992), old-mode tasks are allowed to complete their last activation upon the arrival of a *MCR*, but are no longer released during the mode change. The mode-change model does not include aborted tasks. Wholly new tasks are released after a sufficient offset Y after the MCR. New-mode changed tasks are released right after the end of the period of the corresponding old-mode task. Because only wholly new tasks can be introduced with an offset, the ability to make any transition schedulable is reduced.

Pedro & Burns (1998) introduced an asynchronous protocol without periodicity, which included aborted tasks in the mode-change model. Considering that in this protocol all new-mode tasks can have offsets, it is relatively easy to find a schedulable transition. The schedulability analysis is relatively simplified compared to that of Tindell, Burns & Wellings (1992), since the number of time windows to analyze is lower than in the previous protocol.

Real (2000) proposes an asynchronous protocol with periodicity that merges the advantages of the last two protocols. The mode-change model is similar to that of Pedro & Burns (1998). Nevertheless, an offset Z is introduced for unchanged tasks, relative to the end of the period of the corresponding old-mode task. An offset $Z = 0$ means that the unchanged task is introduced immediately after the end of the period of its corresponding old-mode version. The inclusion of this offset allows the desired periodicity for unchanged tasks. However, when the task set is unschedulable, it is possible to lose periodicity in order to gain schedulability by increasing the value of Z.

5. Schedulability analysis of mode changes

In the following subsection we review previous work on mode changes with deadlines less than or equal to periods, before we relax this constraint.

5.1 Mode changes with deadlines less than or equal to periods

We must find the WCRT for both old-mode tasks and new-mode tasks defined in the mode-change model. We first consider analysis for the old-mode task set. The analysis gives exact (both necessary and sufficient) bounds on the worst-case response time of each task.

5.1.1 Analysis of old-mode tasks

Old-mode tasks suffer the interference from higher-priority aborted tasks before the mode change, and from higher-priority completed tasks before and after the mode change (Fig. 1). During the mode change, old-mode tasks are also preempted by higher-priority new-mode tasks. Clearly, there is no interference from aborted tasks during the transition. We now show the interference terms for each type of task.

5.1.1.1 Interference from higher-priority old-mode completed tasks

Old-mode, higher-priority completed tasks τ_j will interfere with task τ_i mostly over the interval x. This interference extends to the mode-change window as the higher-priority tasks still run in order to complete their execution. The total interference is formulated as the ceiling function of x/T_j as follows:

$$I_{hp(O)} = \sum_{\forall\ \tau_{j(O)}\ hp\ \tau_i} \left\lceil \frac{x}{T_j} \right\rceil C_j \tag{1}$$

5.1.1.2 Interference from higher-priority aborted tasks

Aborted tasks will interfere over a lower-priority task τ_i during the interval x. Within the interval x there is an integral number of periods T_j of the higher-priority aborted task τ_j, and therefore a set of instances that complete their execution. It is only the last release that runs across the *MCR* and has to be aborted. The interference from completed releases of task τ_j before the *MCR* is given by:

$$\sum_{\forall\ \tau_{j(A)}\ hp\ \tau_i} \left\lfloor \frac{x}{T_j} \right\rfloor C_j \tag{2}$$

We also need to consider the amount of aborted execution time of a higher-priority task in the old mode: This occurs during the remaining time w_{rem}, which is a fraction of the period of the aborted task preceding the *MCR*, and is given by:

$$w_{rem} = x - \left\lfloor \frac{x}{T_j} \right\rfloor T_j \tag{3}$$

The *remaining time* w_{rem} can be large enough to: 1) allow only a partial execution of the task τ_j (equation (4)), since $w_{rem} < C_j$, or 2) accommodate an additional execution of the higher-priority aborted task (equation (5)), since $w_{rem} \geq C_j$):

$$0 < x - \left\lfloor \frac{x}{T_j} \right\rfloor T_j < C_j \tag{4}$$

$$C_j \leq x - \left\lfloor \frac{x}{T_j} \right\rfloor T_j \leq T_j \tag{5}$$

In the first case (eq. 4), $w_{rem} < C_j$ and the interference is the minimum value between C_j and w_{rem}, since there is only a partial execution of the aborted task (i.e. $I_{hp(A)rem} = \min(w_{rem}, C_j)$.

In the second case (eq. 5), $w_{rem} \geq C_j$ and there is another full execution of the task within w_{rem}: it is not really necessary to abort the task. The remaining time is greater than C_j but less than the period. Again the interference is the minimum value between C_j and w_{rem}. Once the task completes there is no further releases in the remaining time. Therefore, the amount of interference is C_j. In any case, the partial interference is always the minimum of the intervals w_{rem} and C_j. Therefore, the amount of partial (or aborted) interference is given by:

$$I_{hp(A)rem} = \min\left(x - \left\lfloor \frac{x}{T_j} \right\rfloor T_j, C_j\right) \tag{6}$$

Combining both terms (6) and (2), the total interference from higher-priority aborted tasks is:

$$I_{hp(A)} = \sum_{\forall\, \tau_{j(A)}\, hp\, \tau_i} \left(\left\lfloor \frac{x}{T_j} \right\rfloor C_j + \min\left(x - \left\lfloor \frac{x}{T_j} \right\rfloor T_j, C_j\right)\right) \tag{7}$$

5.1.1.3 Interference from higher-priority new-mode tasks

Higher-priority new-mode tasks do not preserve their mode-change periodicity. The analysis must account for the fact that these tasks are released with an offset Y_j from the MCR. Their interference interval is therefore reduced to $w_i - x - Y_j$ and expressed as:

$$I_{hp(N)} = \sum_{\forall\, \tau_{j(N)}\, hp\, \tau_i} \left\lceil \frac{w_i - x - Y_j}{T_j} \right\rceil_0 C_j \tag{8}$$

5.1.1.4 Interference from higher-priority unchanged new-mode tasks

As discussed in section 3, unchanged new-mode tasks preserve the mode-change periodicity. This term is derived by Real & Crespo (2004), and given by:

$$I_{hp(N)\_per} = \sum_{\forall\, \tau_{j(U)}\, hp\, \tau_i} \left\lceil \frac{w_i(x) - \lceil \frac{x}{T_j} \rceil T_j - Z_j}{T_j} \right\rceil_0 C_j \tag{9}$$

Combining all the interference terms, we obtain the total interference suffered from the old-mode task τ_i across the mode change:

$$I_{mc} = \sum_{\forall\, \tau_{j(O)}\, hp\, \tau_i} \left\lceil \frac{x}{T_j} \right\rceil C_j + \sum_{\forall\, \tau_{j(A)}\, hp\, \tau_i} \left(\left\lfloor \frac{x}{T_j} \right\rfloor C_j + \min\left(x - \left\lfloor \frac{x}{T_j} \right\rfloor T_j, C_j\right)\right) +$$

$$\sum_{\forall\, \tau_{j(N)}\, hp\, \tau_i} \left\lceil \frac{w_i - x - Y_j}{T_j} \right\rceil_0 C_j + \sum_{\forall\, \tau_{j(U)}\, hp\, \tau_i} \left\lceil \frac{w_i(x) - \lceil \frac{x}{T_j} \rceil T_j - Z_j}{T_j} \right\rceil_0 C_j \tag{10}$$

By adding the Worst-Case Execution Time (WCET) C_i of the old-mode task being analyzed, we obtain the $WCRT$ of task τ_i as follows:

$$w_i(x) = C_i + \sum_{\forall\, \tau_{j(O)}\ hp\ \tau_i} \left\lceil \frac{x}{T_j} \right\rceil C_j + \sum_{\forall\, \tau_{j(A)}\ hp\ \tau_i} \left\lfloor \frac{x}{T_j} \right\rfloor C_j + \min\left(x - \left\lfloor \frac{x}{T_j} \right\rfloor T_j, C_j \right) + \sum_{\forall\, \tau_{j(N)}\ hp\ \tau_i} \left\lceil \frac{w_i - x - Y_j}{T_j} \right\rceil_0 C_j + \sum_{\forall\, \tau_{j(U)}\ hp\ \tau_i} \left\lceil \frac{w_i(x) - \lceil \frac{x}{T_j} \rceil T_j - Z_j}{T_j} \right\rceil_0 C_j \quad (11)$$

The solution to equation (11) is obtained by forming a recurrence equation on w_i, to find the smallest positive integer that satisfies it. Since many values of x give the same WCRT, the significant values of x are the ones that lead to new values for the ceiling and floor functions. The set of significant values x is the set of all positive values in the domain of $w_i(x)$ such that:

$$x \,\epsilon\, (0, \epsilon, T_j + \epsilon, 2T_j + \epsilon, 3T_j + \epsilon, ..R_i^{ss}) \;\forall\, \tau_{j(O)}\ hp\ \tau_i \text{ and}$$
$$x \,\epsilon\, (0, C_j, T_j + C_j, 2T_j + C_j, ..R_i^{ss}) \;\forall\, \tau_{j(A)}\ hp\ \tau_i \quad (12)$$

where ϵ is the time quantum, which we assume to be 1 and R_i^{ss} is the steady-state WCRT of task τ_i.

5.1.2 Analysis of new-mode tasks

In the worst-case scenario, all old-mode tasks are released momentarily before the mode change, thus sharing a critical instant with the window w_i. The interference from higher-priority new-mode tasks upon a new-mode task τ_i must account for the offset of the interfering task, during which there is no interference. The interference from unchanged new-mode tasks must account for the offset Z_j, from the end of the period of the preceding old-mode version. The worst-case response time of a new-mode task across a mode change is given by: (Pedro & Burns, 1998; Real & Crespo, 2004)

$$w_i = C_{i(N)} + \sum_{\forall\, \tau_{j(O)}\ hp\ \tau_i} C_j + \sum_{\forall\, \tau_{j(N)}\ hp\ \tau_i} \left\lceil \frac{w_i - Y_j}{T_j} \right\rceil_0 C_j + \sum_{\forall\, \tau_{j(U)}\ hp\ \tau_i} \left(C_j + \left\lceil \frac{w_i - T_j - Z_j}{T_j} \right\rceil_0 C_j \right) \quad (13)$$

This equation must also be solved using recurrence relation as before. Considering that task τ_i is released with an offset Y_i after the mode change, then to obtain the value of R_i window w_i must be decreased by Y_i:

$$R_i = w_i - Y_i \quad (14)$$

If the expression $w_i - C_{i(n)} \leq Y_i$ holds, the amount of interference over new-mode task τ_i is smaller than the value of its mode-change offset Y_i. Therefore, the new-mode task τ_i is released in the steady state after all old-mode tasks have completed.

We now turn to the analysis of tasks across a mode change with arbitrary deadlines.

5.2 Mode-changes with arbitrary deadlines

Lehoczky describes qualitative analysis which can determine the worst-case response time of a given task with such arbitrary deadlines (Lehoczky, 1990). Tindell et al. (1994) derived analysis from that of M. & Pandya (1986) and using the approach of Lehoczky, for static priority pre-emptive systems that permit tasks to have arbitrary deadlines, release jitter, and behave as sporadically periodic tasks. His analysis illustrates how using a window approach to finding worst-case response times for these tasks is an appropriate way of obtaining an analysis tailored to the behavior of real-time tasks. In single mode, steady-state systems, the busy period $w_{i,q}$ for a task τ_i with arbitrary deadlines is calculated as follows (Tindell et al., 1994):

$$w_{i,q} = (q+1)C_i + I_{hp} \tag{15}$$

Where the term I_{hp} is the interference from higher-priority tasks, and q is the number of instances of task τ_i during the *busy period*. Equation 15 is the basis to extend current analysis of mode changes to allow arbitrary deadlines. Therefore, for the analysis of mode changes with arbitrary deadlines, we need to: 1) review the notion of busy period in the light of mode changes, specifically the definition of the beginning and end of the busy period; 2) find the number of instances q of a task τ_i to be analyzed, and 3) determine the amount of interference from higher-priority computation (I_{hp}), as follows:

- *Busy periods*: A *level-i* busy period is defined as *the maximum time for which a processor executes tasks of priority greater than or equal to the priority of task* τ_i. Lehoczky shows how the worst-case response time of a task τ_i can be found by examining a number of busy periods, each starting at an arrival of task τ_i (hence some multiple of T_i before the current invocation of task τ_i). The busy period ends when a lower-priority task is able to begin execution. To find the worst-case response time, all invocations of task τ_i in this busy period must be examined. In Fig. 2 task τ_i completes when the given busy period finishes, but arrives at some point in time after the start of the busy period. By knowing the width of the busy period and when the busy period starts (relative to arrival of task τ_i) then we can find the corresponding response time. The width of the busy period is equal to the higher-priority computation that is released in it. The WCRT of task τ_i is the longest of the response times corresponding to each of the examined busy periods. In the next section we discuss the beginning and end of the busy period regarding a mode change.
- *Number of instances to analyze*: If a task has a *WCRT* greater than its period, then the possibility exists for a task to re-arrive before the previous invocation has completed. In this case we assume that the new arrival is deemed to have a lower priority, and is therefore delayed from executing until after the previous invocation terminates. Fig. 3 illustrates the response time of a task τ_i, with its deadline larger than its period, executing across a mode change as an old-mode completed task. The *MCR* arrives during the execution of an arbitrary invocation of task τ_i. To facilitate the analysis we assume, without loss of generality, that the *MCR* arrives while the third invocation of task τ_i is under execution. In a busy period without a mode change, the 4^{th} and 5^{th} invocations are released and execute regularly. However, with the arrival of a *MCR*, the 4^{th} and the 5^{th} invocations do not need to be released anymore, according to the mode-change model. The cancellation of both the 4^{th} and the 5^{th} releases will shorten the length of the busy period, causing the lower-priority task (*task low*) to begin execution earlier. Therefore, we need to apply the

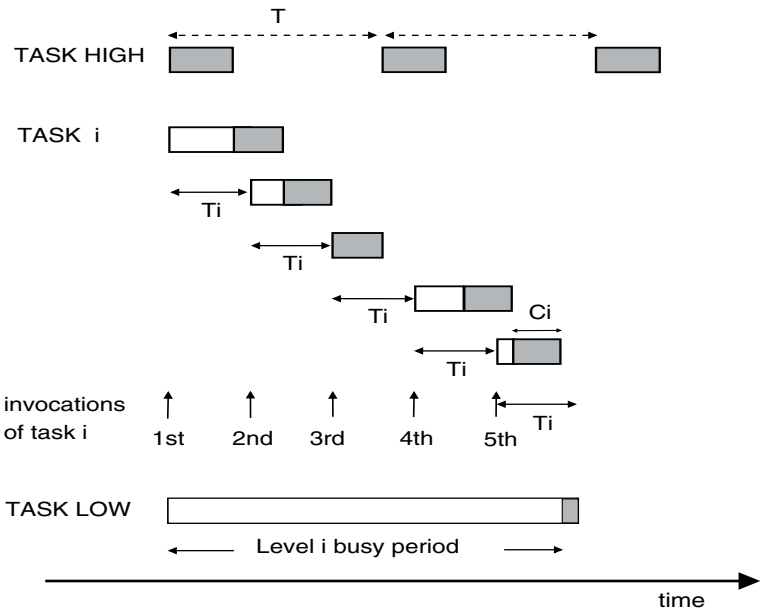

Fig. 2. A level *i* busy period (Lehoczky, 1990), (Tindell et al., 1994)

schedulability analysis only for the first three invocations. However, it is only the third invocation that needs to be analyzed using mode-change analysis. The first and second invocations can be analyzed using the steady-state equations derived by Tindell et al. (1994).

- *Higher-priority interference*: Fig. 2 also illustrates a higher-priority unchanged task (T_{high}). Clearly, other types of tasks such as wholly new and changed new-mode tasks may interfere with the execution of task τ_i. While an invocation of T_{high} may be delayed by a previous invocation, the number of instances of T_{high} interfering with task τ_i is not affected by the introduction of arbitrary deadlines. Therefore, the calculation of higher-priority computation under the assumption of arbitrary deadlines remains the same as the one from previous work (Real & Crespo, 2004).

We now look at the analysis of old-mode tasks.

5.2.1 Analysis of old-mode tasks

We begin by extending the notion of busy period for old-mode tasks: *A level "i" busy period for an old-mode task across a mode change is defined as the maximum time for which a processor executes tasks of priority greater than or equal to the priority of task. A busy period begins at a time x before the arrival of a mode-change request* (MCR). *The busy period ends during the transition, with the completion of the last invocation q of task* τ_i *before the steady-state new mode, when a lower-priority task is able to begin execution.*

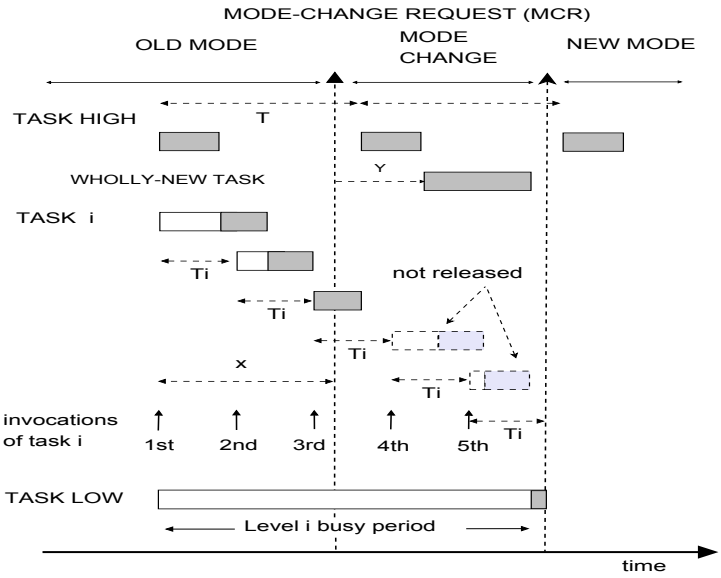

Fig. 3. Mode-Change with Arbitrary Deadlines

From the definition of latency of mode changes (section 3), it follows that the end of the busy period falls inside the mode-change window. This is because task τ_{low} is an old-mode completed and the mode change is not over until all the old-mode tasks have completed.

Consider the three instances of task τ_i before the *MCR* in Fig. 3. Instances 1 and 2 are called the steady-mode instances, since they are released and completed before the *MCR*. We are interested in finding the *WCRT* for the third instance, the *old-mode completed instance*, which runs across the *MCR* and executes through the mode change. When the old-mode completed invocation begins, the steady-mode invocations have already completed. Therefore, their schedulability is guaranteed by steady-state analysis (Audsley et al., 1993).

By defining the beginning and the end of the busy period as above we can reuse the work from Pedro & Burns (1998) and Real (2000) by including in their analysis the multiple *WCET*s introduced by the steady-mode releases (i.e. previous invocations of task τ_i) during window x. In our example, the third invocation is delayed by the first and second invocations.

There are no further releases of the old-mode completed task after the *MCR*, according to the mode-change protocol. As we increase the value of x in the analysis, more steady-state invocations of task τ_i appear before the *MCR*, potentially delaying the old-mode completed task τ_i. The impact of a mode change on the analysis of old-mode tasks is that there are less releases of task τ_i to be analyzed than in steady-state analysis. The number of instances q of task τ_i that need to be analyzed before the mode change is given by:

$$q_i = \left\lceil \frac{x}{T_i} \right\rceil \tag{16}$$

and it is only the last invocation that requires mode-change analysis. The interference suffered by the mode-changing task includes the types of tasks from the mode-change model, i.e. old-mode aborted, old-mode completed, new-mode changed, unchanged and wholly-new tasks as previously analyzed. Taking these remarks into consideration, the analysis of a mode-change for an old-mode task with arbitrary deadlines is given by:

$$w_i(x) \;=\; (q+1)C_i + \sum_{\forall\, \tau_{j(O)}\; hp\; \tau_i} \left\lceil \frac{x}{T_j} \right\rceil C_j + \sum_{\forall\, \tau_{j(A)}\; hp\; \tau_i} \left(\left\lfloor \frac{x}{T_j} \right\rfloor C_j + \min\left(x - \left\lfloor \frac{x}{T_j} \right\rfloor T_j, C_j \right) \right) +$$

$$\sum_{\forall\, \tau_{j(N)}\; hp\; \tau_i} \left\lceil \frac{w_i(x) \;-\; x - Y_j}{T_j} \right\rceil_0 C_j \;+ \sum_{\forall\, \tau_{j(U)}\; hp\; \tau_i} \left\lceil \frac{w_i(x) \;-\; \lceil \frac{x}{T_j} \rceil T_j - Z_j}{T_j} \right\rceil_0 C_j \quad (17)$$

In single-mode, steady-state systems, windows for increasing values of q need to be determined. The sequence is finite, however, because the search can stop if a level i busy period is found which finishes before task τ_i starts (i.e. the processor is released to process lower-priority tasks) - since the processor is executing lower priority tasks there can be no impact on task τ_i from previous invocations of task τ_i in busy periods starting earlier. The above iteration over increasing values of q can stop if:

$$w_i(q) \;< (q+1)T_i \quad (18)$$

The response time corresponding to the window starting qT_i before the current invocation of task τ_i is therefore given by:

$$(w_i(q) \;-\; qT_i) \quad (19)$$

Now, as mentioned above, the worst-case response time can occur at any one of these response times, and thus the WCRT is given by:

$$r_i = \max(w_i(q) - qT_i) \quad q = 1, 2, 3.. \left\lceil \frac{x}{T_i} \right\rceil \quad (20)$$

In the analysis with arbitrary deadlines and single mode systems, windows for increasing values of q need to be determined; variable x is not present in the analysis. In the WCRT analysis of mode changes with deadlines less than or equal to periods we need to apply all significant values of x, and the analysis is not a function of q. Clearly, the analysis of tasks with arbitrary deadlines across a mode change is a function of both x and q, and q is limited by x (for old-mode tasks).

5.2.2 Analysis of new-mode tasks

A level *i* busy period for a new-mode task is defined as: *The maximum time for which a processor executes tasks of priority greater than or equal to the priority of task τ_i. A busy period begins with the arrival of a mode-change request MCR. It ends when a task with lower priority than τ_i is able to begin execution during the mode-change window.*

As seen in section 5.2, the pattern of interference from higher computation over τ_i is not affected by the introduction of arbitrary deadlines. As with the analysis for old-mode tasks, the new-mode task is delayed by its previous invocations. Therefore, equation (13) is reformulated as follows:

$$w_i = (q+1)C_{i(N)} + \sum_{\forall \tau_{j(O)}\ hp\ \tau_i} C_j + \sum_{\forall \tau_{j(N)}\ hp\ \tau_i} \left\lceil \frac{w_i - Y_j}{T_j} \right\rceil_0 C_j + \sum_{\forall \tau_{j(U)}\ hp\ \tau_i} \left(C_j + \left\lceil \frac{w_i - T_j - Z_j}{T_j} \right\rceil_0 C_j \right) \quad (21)$$

Unlike the analysis of old-mode tasks, which limits the number of busy periods to be examined by $\lceil \frac{x}{T_i} \rceil$, in the analysis of new-mode tasks the number of instances q to be inspected has no direct correlation with the beginning or end of the mode change. The above iteration over increasing values of q can stop if:

$$w_i(q) < (q+1)T_i \quad (22)$$

The first qT of level i busy period falls before the current invocation of task τ_i is released. The response time corresponding to the given level i busy period starting time qt before the current invocation of task τ_i is therefore given by: $w_{i,q} - qt$. The response time can occur at any one of these response times, and thus the worst-case response time is given by:

$$R_i = w_{i,q} - qt - Y_i \quad (23)$$

Where $q = \{0, 1, 2...\}$. This equation specifies an infinite number of busy periods. Only a finite number need to be examined because the search can stop if a level i busy period is found which finishes before the invocation of task τ_i following the current one. The longest busy period that needs to be examined is bounded by the LCM of task periods.

6. Example

In this section we present an example of the mode change analysis based upon the Generic Avionics Platform (GAP) task set described by Locke et al. (1991). We consider the example of two modes of aircraft operation: *Level flight* and *Defense Mode*, described in tables 1 and 2 respectively (Pedro, 1999), and we analyze the schedulability of the transition from level-flight to defense mode. The level-flight mode contains four tasks which are not originally defined in the GAP set: auto_pilot, mission_advisor, fuelling_management and display_graphic_2. The task set used in the defense mode is the original task set defined (Locke et al., 1991), except that the Timer_Interrupt task has been removed.

Tasks printed in boldface in table 1 denote completed old-mode tasks that are replaced by the corresponding boldfaced wholly-new tasks in table 2 (e.g. Weapon_Release replaces Auto_Pilot). Tasks in the defense mode are carried out as new-mode changed tasks. Display _Hook_Update is an aborted task. In order to increase the schedulability of the task set, all new-mode tasks lose their periodicity during the transition. Table 3 illustrates the

τ	Behavior	τ	Behavior
$\tau_{1(O)}$	**Auto_pilot**	$\tau_{19(O)}$	Tracking_Target_Upd
$\tau_{3(O}$	Radar_Tracking_Filter	$\tau_{21(O)}$	**Display_Graphic_2 ***
$\tau_{5(O)}$	RWR_Contact_Mgmt	$\tau_{23(O)}$	Nav_Steering_Cmds
$\tau_{7(O)}$	Data_Bus_Poll_Device	$\tau_{25(O)}$	Display_Stores_Updates
$\tau_{9(O)}$	**Mission_advisor ***	$\tau_{27(O)}$	Display_Keyset
$\tau_{11(O)}$	**Fuelling_Mgmt ***	$\tau_{29(O)}$	Display_Stat_Update
$\tau_{13(O)}$	Nav_Update	$\tau_{31(O)}$	BET_E_Status_Update
$\tau_{15(O)}$	Display_Graphic_1	$\tau_{33(O)}$	Nav_Status
$\tau_{17(A)}$	Display_Hook_Update	—	—

Table 1. Task Set Description in Aircraft Level-Flight Mode

schedulability analysis for the mode change from the level-flight mode to the defense mode. Each column represents the period T, deadline D, $WCET$ C, priority P, the worst-case arrival time for an old-mode task x, the steady-state response time for a task in the old mode $R^{ss}_{i(O)}$, the $WCRT$ of a task during the mode change $R_{i(mc)}$ and the steady-state WCRT of a new-mode task $R^{ss}_{i(N)}$. Tasks with a lower value of P have a higher priority. All times are specified in milliseconds with a multiplication factor of 10.

The latency of the mode change is given by new-mode task $\tau_{34(c)}$, which completes its first execution at time 21400 (i.e. $R_i(n) + Y_i$ = 2140 ms) after the MCR. The worst-case response times of all tasks are less than the deadlines, and hence the task set is schedulable across the mode change.

τ	Behavior	τ	Behavior
$\tau_{2(W)}$	**Weapon_Release ***	$\tau_{20(C)}$	Tracking_Target_Upd
$\tau_{4(C)}$	Radar_Tracking_Filter	$\tau_{22(W)}$	**Weapon_Protocol ***
$\tau_{6(C)}$	RWR_Contact_Mgmt	$\tau_{24(C)}$	Nav_Steering_Cmds
$\tau_{8(C)}$	Data_Bus_Poll_Device	$\tau_{26(C)}$	Display_Stores_Updates
$\tau_{10(W)}$	**Weapon_Aiming ***	$\tau_{28(C)}$	Display_Keyset
$\tau_{12(W)}$	**Radar_Target_Update**	$\tau_{30(C)}$	Display_Stat_Update
$\tau_{14(C)}$	Nav_Update	$\tau_{32(C)}$	BET_E_Status_Update
$\tau_{16(C)}$	Display_Graphic	$\tau_{34(C)}$	Nav_Status
$\tau_{18(C)}$	Display_Hook_Update	—	—

Table 2. Task Set Description in Aircraft Defense Mode

If task WCRT's are much less than their periods (i.e. $R_i << T_i$) then the analysis will converge immediately for values of $q = 0$. There are no extra invocations (q = 1,2,3..) in the busy period. Task $\tau_{13(O)}$ is an exception: it has a period T_{13} = 1100 and deadline D_{13} = 1550 (D > T). Its WCRT is 1227 and it occurs when the task arrives at time x = 801 before the MCR. In the worst-case, $\tau_{13(O)}$ is delayed by one single invocation (i.e. q = 1) before the MCR, but it is still able to meet its deadline.

τ	T	D	C	P	Y	x	$R^{ss}_{i(O)}$	$R_{i(mc)}$	$R^{ss}_{i(N)}$	TEST
$\tau_{1(O)}$	1000	50	10	1	—	0	10	10	—	sched.
$\tau_{2(W)}$	2000	50	30	1	0	0	—	40	30	sched.
$\tau_{3(O)}$	2000	1200	200	12	—	601	742	862	—	sched.
$\tau_{4(C}$	250	60	20	2	2000	0	—	50	50	sched.
$\tau_{5(O)}$	2000	1400	5	13	—	601	747	897	—	sched.
$\tau_{6(C}$	250	120	50	3	2000	0	—	100	100	sched.
$\tau_{7(O)}$	400	400	10	4	—	1	100	130	—	sched.
$\tau_{8(C)}$	400	400	10	4	400	0	—	110	110	sched.
$\tau_{9(O)}$	600	450	20	5	—	1	120	150	—	sched.
$\tau_{10(W)}$	500	450	30	5	0	0	—	180	140	sched.
$\tau_{11(O)}$	800	500	50	6	—	1	170	230	—	sched.
$\tau_{12(W)}$	500	500	50	6	0	0	—	280	190	sched.
$\tau_{13(O)}$	1100	1550	80	15	—	801	977	1227	—	sched.
$\tau_{14(C)}$	590	590	80	7	1650	0	—	340	340	sched.
$\tau_{15(O)}$	1700	1600	40	16	—	1001	1107	1227	—	sched.
$\tau_{16(C)}$	800	600	90	8	1700	0	—	440	440	sched.
$\tau_{17(A)}$	1700	1650	100	17	—	1001	1237	1367	—	—
$\tau_{18(C)}$	800	700	20	9	1700	0	—	460	460	sched.
$\tau_{19(O)}$	2000	800	30	10	—	251	342	452	—	sched.
$\tau_{20(C)}$	1000	800	50	10	2000	0	—	740	740	sched.
$\tau_{21(O)}$	3000	900	90	11	—	401	442	552	—	sched.
$\tau_{22(W)}$	2000	900	10	11	0	0	—	482	750	sched.
$\tau_{23(O)}$	250	60	20	2	—	1	30	60	—	sched.
$\tau_{24(C)}$	2000	1200	30	12	250	0	—	542	970	sched.
$\tau_{25(O)}$	250	120	60	3	—	1	90	120	—	sched.
$\tau_{26(C)}$	2000	1400	10	13	250	0	—	567	980	sched.
$\tau_{27(O)}$	3000	1500	10	14	—	801	897	1017	—	sched.
$\tau_{28(C)}$	2000	1500	10	14	3000	0	—	990	990	sched.
$\tau_{29(O)}$	4000	590	30	7	—	1	200	310	—	sched.
$\tau_{30(C)}$	2000	1550	30	15	4000	0	—	1380	1380	sched.
$\tau_{31(O)}$	20000	600	15	8	—	1	215	325	—	sched.
$\tau_{32(C)}$	10000	1600	10	16	20000	0	—	1390	1390	sched.
$\tau_{33(O)}$	20000	700	17	9	—	1	232	342	—	sched.
$\tau_{34(C)}$	10000	1650	10	17	20000	0	—	1400	1400	sched.

Table 3. Feasibility Analysis of GAP Task Set Across a Mode Change

7. Summary and discussion

Before we discuss the schedulability analysis of tasks across a mode change, we must define what a mode of operation is in real-time systems and what a mode change from a source to a target mode represents. In the first part of this chapter we have surveyed the literature and presented a number of views on the notion of modes, before formulating our definition for real-time systems.

The second focus of this work was on guaranteeing hard real-time tasks with arbitrary deadlines that execute through a mode change. Original work on schedulability analysis for real-time tasks across mode changes assumes that all task have deadlines less than or equal to their periods. This work relaxed this constraint, by allowing tasks to have arbitrary deadlines. It also showed the generality of the analysis presented by Tindell et al. (1994) on arbitrary deadlines: when proper busy periods are considered, schedulability analysis for fixed-priority preemptive systems is amenable to extensions such as the one presented in this chapter. From another perspective, the schedulability results of mode changes by Pedro & Burns (1998) and Real & Crespo (2004) can be extended to allow for arbitrary deadlines without major modifications to their original analysis.

In order to introduce arbitrary deadlines in the schedulability analysis of mode changes of Pedro & Burns (1998) and Real & Crespo (2004) we had to consider: 1) The definition of busy periods in the light of mode changes; 2) The amount of higher-priority computation; 3) The number of instances q of the task being analyzed τ_i, and 4) The delays from earlier invocations of task τ_i. Therefore, we introduced the following modifications to the original analysis:

1. Readjusted the beginning of the busy period with regard to the arrival of the *MCR* and adopted the basic definition as given by Lehoczky (1990);
2. Maintained the calculation of the interference from higher-priority tasks: the introduction of arbitrary deadlines does not change the amount of interference from higher-priority tasks. Clearly, higher-priority tasks can be delayed by their previous invocations, but this does not change the calculation of the higher-priority computational load;
3. Changed the number of instances q to be inspected: For old-mode tasks, the arrival of the *MCR* changes the number of busy periods to be inspected. In the schedulability analysis of mode changes, the condition $w < (q+1)T_i$ can be reached much earlier than in the corresponding analysis of steady-state (single-mode) systems. It occurs long before the *LCM* of tasks and it depends on the value of x. In addition, the mode-change analysis refers only to the last invocation of task τ_i before the *MCR*: the preceding invocations do not cross the mode-change and merely delay task τ_i. For new-mode tasks we analyze a number of invocations q until the condition $w < (q+1)T_i$ is satisfied;
4. Maintained the delay of previous invocations of the task being analyzed in the analysis of both old-mode tasks and new-mode ones: Because we considered that the previous instance of task τ_i has higher priority than the new release, it will not preempt but instead delay the execution of the instance being analyzed.

This work will allow us to investigate more complex systems and applications that require mode changes using arbitrary deadlines. A good example is the schedulability analysis of the *Controller Area Network (CAN)* (Davis et al., 2007), which is based on arbitrary deadlines,

but assumes a fixed message set with one single mode of operation. Before we tackle the schedulability analysis of messages across a mode change in a CAN bus, we need to be familiar with the schedulability analysis of mode changes with arbitrary deadlines, such as the one derived in this chapter.

8. References

Audsley, N., Burns, A., Richardson, M., Tindell, K. & Wellings, A. J. (1993). Applying new scheduling theory to static priority pre-emptive scheduling, *Software Engineering Journal* 8: 284–292.

Bailey, C. M. (1993). Hard real time operating system kernel: Investigation of mode change, task 14 deliverable on estsec contract 9198/90/nl/sf, *Technical report*, British Aerospace Systems Ltd.

Davis, R. I., Burns, A., Bril, R., & Lukkien, J. (2007). Controller area network (can) schedulability analysis: Refuted, revisited and revised, *Real-Time Systems* 35: 239–272.

Degani, A. & Kirlik, A. (1995). Modes in human-automation interaction:initial observations about a modelling approach, *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 1–15.

Degani, A., Shafto, M. & Kirlik, A. (1999). Modes in human-machine systems: Review, classification, and application, *International Journal of Aviation Psychology* 9(2): 125–138.

Fohler, G. J. (1994). *Flexibility in Statically Scheduled Hard Real-Time Systems*, PhD thesis, Technische Universitat Wien, Institut fur Technische Informatik.

Howe, D. (1997). Free On-line Dictionary of Computing. http://wombat.doc.ic.ac.uk /foldoc /index.html.

Lehoczky, J. (1990). Fixed priority scheduling of periodic task set with arbitrary deadlines, *Proceedings of the 11th Real Time Systems Symposium*, pp. 201–209.

Locke, C. D., Vogel, D. & Mesler, T. (1991). Building a Predictable Avionics Platform in Ada: A Case Study, *Proceedings of the 12th Real-Time Systems Symposium (Dec.)*, pp. 181–189.

M., J. & Pandya, P. (1986). Finding response times in a real-time system, *BCS Computer Journal* 29(05): 390–395.

Martins, P. & Burns, A. (2008). On the meaning of modes in uniprocessor real-time systems, *Proceedings of the 2008 ACM symposium on Applied computing*, SAC '08, ACM, New York, NY, USA, pp. 324–325.
URL: *http://doi.acm.org/10.1145/1363686.1363770*

Norman, D. A. (1981). Categorization of action slips, *Psychological Review* 1(88): 1–15.

Papadopoulos, Y. (1996). Real-Time Safety Administration by Using Safety Cases, *Technical Report Oct*, The University of York, Computer Science,2nd Year Thesis Proposal.

Pedro, P. (1999). *Schedulability of Mode Changes in Flexible Real-Time Distributed Systems*, PhD thesis, The University of York.

Pedro, P. & Burns, A. (1998). Schedulability analysis for mode changes in flexible real-time systems, *Real-Time Systems, 1998. Proceedings. 10th Euromicro Workshop on*, pp. 172 –179.

Poller, M. F. & Garter, S. K. (1984). The Effects of Modes on Text Editing by Experienced Editor Users, *Human Factors*, Vol. 26(4), pp. 449–462.

Real, J. (2000). *Protocolos de Cambio de Modo para Sistemas de Tiempo Real*, PhD thesis, Universidad Politecnica de Valencia.

Real, J. & Crespo, A. (2004). Mode change protocols for real-time systems: A survey and a new proposal, *Real-Time Systems* 26: 161–197. 10.1023/B:TIME.0000016129.97430.c6. URL: *http://dx.doi.org/10.1023/B:TIME.0000016129.97430.c6*

Tesler, L. (1981). The SmallTalk Environment, *Byte Magazine*, Vol. 6(8), pp. 90–147.

Tindell, K. & Alonso, A. (1996). A Very Simple Protocol for Mode Changes in Priority Preemptive Systems, *Technical report*, Universidad Politécnica de Madrid.

Tindell, K., Burns, A. & Wellings, A. (1994). An Extendible Approach for Analysing Fixed Priority Hard Real-Time Tasks, *Journal of Real-Time Systems* 6(2): 133–151.

Tindell, K. W., Burns, A. & Wellings, A. (1992). Mode changes in priority pre-emptively scheduled systems, *Technical Report RTSS92-TBW*, Department of Computer Science.

Tindell, K. W., Burns, A. & Wellings, A. J. (1992). Mode changes in priority pre-emptively scheduled systems, *Proceedings of the Real Time Systems Symposium*, pp. 100–109.

Real-Time Systems, Architecture, Scheduling, and Application

Edited by Dr. Seyed Morteza Babamir

ISBN 978-953-51-0510-7
Hard cover, 334 pages
Publisher InTech
Published online 11, April, 2012
Published in print edition April, 2012

This book is a rich text for introducing diverse aspects of real-time systems including architecture, specification and verification, scheduling and real world applications. It is useful for advanced graduate students and researchers in a wide range of disciplines impacted by embedded computing and software. Since the book covers the most recent advances in real-time systems and communications networks, it serves as a vehicle for technology transition within the real-time systems community of systems architects, designers, technologists, and system analysts. Real-time applications are used in daily operations, such as engine and break mechanisms in cars, traffic light and air-traffic control and heart beat and blood pressure monitoring. This book includes 15 chapters arranged in 4 sections, Architecture (chapters 1-4), Specification and Verification (chapters 5-6), Scheduling (chapters 7-9) and Real word applications (chapters 10-15).

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Paulo Martins, I. G. Hidalgo, M. A. Carvalho, A. de Angelis, V.Timoteo, R. Moraes, E. Ursini and Udo Fritzke Jr (2012). Schedulability Analysis of Mode Changes with Arbitrary Deadlines, Real-Time Systems, Architecture, Scheduling, and Application, Dr. Seyed Morteza Babamir (Ed.), ISBN: 978-953-51-0510-7, InTech, Available from: http://www.intechopen.com/books/real-time-systems-architecture-scheduling-and-application/schedulability-analysis-of-mode-changes-with-arbitrary-deadlines

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821