Plug-In Based Aperiodic Task Handling for Diverse Real-Time Systems

Björn Lindberg and Gerhard Fohler
Department of Computer Engineering
Mälardalen University
P.O. Box 883, SE 721-23 Västerås, Sweden
{ble, gfr}@mdh.se

1 Introduction

Functionality for various services of scheduling algorithms is typically provided as extensions to a basic algorithm. Aperiodic task handling, guarantees, etc. are integrated with a specific basic scheme, such as earliest deadline first, rate monotonic, or off-line scheduling. Put in another way, various scheduling functionality comes in packages of scheduling schemes, fixed to a certain methodology, confronting designers with an "all-or-nothing" alternative.

This contrasts actual industrial demands: designers want to select various types of functionality without consideration of which "package" they come from. They are reluctant to abandon trusted methods and "to switch packages" for the sake of an additional functional module only. Instead, there is a need to seamlessly integrate new functionality with a developed system, enabling designers to choose the best of various "packages".

In this paper, we propose the use of a "plug-in" approach to add functionality to existing scheduling schemes. In particular, we present a software module for aperiodic task handling. A number of methods have been presented [9, 8, 7], but within their respective "packages" only. Instead of extending an existing scheduling "package", we concentrate the functionality into a module, define an interface and discuss its application to off-line and on-line scheduling methods as examples.

2 Plug-In Definition

A *plug-in* can be thought of as a hardware or software module that adds a specific feature or service to an existing system. The purpose of a plug-in is to add functionality without calling for redesign or extensive modifications. To accomplish this it must be clear what services the plug-in provides and an interface between the plug-in and the target system must be defined.

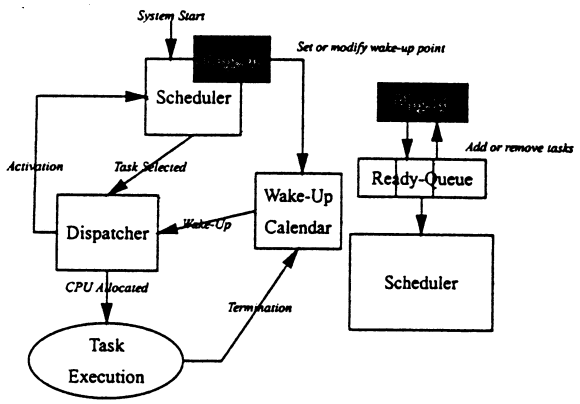


Figure 1. The target system and the plug-in.

2.1 Target System Model

Before discussing our plug-in for aperiodic task handling in a real-time system, we must define a target system model. The model, depicted in figure 1, consists of three separate modules: a scheduler, a dispatcher, and a *wake-up calendar*.

The scheduler's responsibility is to schedule real-time tasks for execution according to some algorithm. The requirement on this algorithm is that tasks are scheduled on an earliest start time and deadline basis. As depicted in figure 1, the plug-in processes aperiodic tasks and puts them into the ready-queue. The scheduler deals with the tasks in the ready-queue, not being concerned as to whether they are aperiodic or not.

The dispatcher allocates the CPU to tasks selected by the scheduler and activates the scheduler as described below.

The wake-up calendar is a module that invokes the dispatcher at so called *wake-up* points. A wake-up point is a point in time when the dispatcher must wake up and activate the scheduler. Such a point could, for example, be a system tick or a point when the dispatcher wakes up to make sure

that a task does not execute more than it is supposed to.

2.2 Plug-In Interface

The interface between the plug-in and the scheduling module is just a set of primitives for adding and removing tasks to and from the ready-queue. The order of the ready-queue is always conserved. The plug-in also needs to make use of the services offered by the wake-up calendar to set and modify wake-up points.

3 Plug-In Based Aperiodic Task Handling

Our plug-in for aperiodic task handling is meant to be "plugged into" a scheduling module that makes scheduling decisions based on earliest start times and deadlines. The plug-in works independently of the scheduling module and can be seen as a layer on top of it.

At all times the scheduling module schedules tasks that are ready to execute, that is, tasks that are present in the ready-queue. The plug-in deals with the aperiodic tasks and places them in the ready-queue. The scheduling module then processes the aperiodic tasks as it would any other tasks in the system.

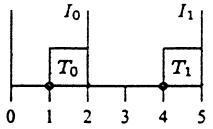
The plug-in's mechanisms for aperiodic task handling are based on the slot shifting [2] algorithm, taking advantage of resources not needed by non-aperiodic tasks and using them to schedule aperiodic tasks.

3.1 Task Model and Terminology

- All tasks T have an earliest possible start time denoted by $est(T)$.
- The maximum execution time [5] of a task T is denoted by $maxt(T)$.
- A task T's deadline is denoted by $dl(T)$.
- Aperiodic tasks Ta have *unknown* arrival times. The earliest start time of an aperiodic task is equal to its arrival time. Aperiodic tasks with known maximum execution times and deadlines are termed *firm* aperiodic. These tasks need to be completed before their deadlines. Aperiodic tasks without deadlines and possibly without known maximum execution times are termed *soft* aperiodic.

3.2 Off-Line Preparations

Before system run-time, the plug-in mechanisms are initialised in an off-line phase. Input to this phase is simply a set of n non-aperiodic tasks $S = \{T_i | i = 0, \ldots, n-1\}$ where, for each task $T_i \in S$, $est(T_i)$, $dl(T_i)$, and $maxt(T_i)$ are known. For this set we create a corresponding set of



Interval	$start(I)$	$end(I)$	$\|I\|$	$maxt(I)$	$sc(I)$	$wu(I)$
I_0	0	2	2	1	1	1
I_1	2	5	3	1	2	4

Figure 2. Two intervals and their attributes.

disjoint execution *intervals*. Each interval, see figure 2, $I_j : j = 0, \ldots, k-1$ is characterised by the following properties.

- The end of the interval: $end(I_j)$. All tasks $T_i \in S$ where $dl(T_i) = end(I_j)$ are said to *belong* to I_j. The fact that a task T_i belongs to interval I_j is denoted by $T_i \in I_j$.
- The start time of the interval: $start(I_j)$.
- The length of the interval: $|I_j| = end(I_j) - start(I_j)$.
- The maximum execution time of all tasks belonging to the interval: $maxt(I_j) = \sum_{\forall T_i \in I_j} maxt(T_i)$.
- The spare capacity of the interval: $sc(I_j) = |I_j| - maxt(I_j) + \min(sc(I_{j+1}), 0)$. The spare capacity of an interval is the amount of resources not used by non-aperiodic tasks in that or subsequent intervals.
- The wake-up point of the interval: $wu(I_j) = start(I_j) + sc(I_j)$. All wake-up points are registered in the wake-up calendar. The wake-up point $wu(I_j)$ of interval I_j is a point in time when the execution of any task $T_k \notin I_j$ must be preempted as to not violate deadlines of tasks $T \in I_j$. These initial wake-up points are identical to the *critical slot* concept of [3].

Further details can be found in [1, 2].

3.3 Guarantee Algorithm

When a firm aperiodic task arrives at the system we apply a guarantee test on it. The task is either accepted and completed before its deadline or rejected.

Assume that at time t a firm aperiodic task Ta arrives at the system. We define $I_c : start(I_c) \leq t < end(I_c)$ to be the *current* interval and $I_f : start(I_f) < dl(Ta) \leq end(I_f)$ to be the *final* interval.

If the sum of all interval spare capacities between I_c and I_f is greater than or equal to Ta's maximum execution time Ta is guaranteed; otherwise it is rejected.

If successful, we update interval spare capacities and wake-up points going from I_f towards I_c. This is because the resources that have been allocated to Ta are no longer available for other tasks.

If $dl(Ta) \neq end(I_f)$, the interval I_f needs to be split into two intervals $I_{f1} : [start(I_f), dl(T_a))$ and $I_{f2} : [dl(T_a), end(I_f))$. The spare capacities and wake-up points of all intervals $I_j : j = c, \ldots, f2$ are then updated. Finally Ta belongs to I_f or I_{f1} depending if I_f needed to be split or not. See [1] for details.

3.4 On-Line Activities

When the plug-in is activated, it updates the intervals in conformity with the last task execution and checks if there are any pending aperiodic tasks. If so, it processes them and puts one or more of them into the ready-queue.

To be able to update the intervals, the plug-in must keep track of which task executed last and when it started its latest execution. Using this information regarding how much time the last task consumed, the plug-in updates interval spare capacities and possibly also wake-up points. Details follow.

Let t be the current time and I_c be the current interval. We define T_{last} to be the last executing task and t_{last} the time when T_{last} began its latest execution. Let $S_{firm} = \{Ta_f^i | i = 0, \ldots, k-1 \wedge dl(Ta_f^i) \leq dl(Ta_f^{i+1})\}$ be the set of pending firm aperiodic tasks and $S_{soft} = \{Ta_s^i | i = 0, \ldots, m-1 \wedge est(Ta_f^i) \leq est(Ta_s^{i+1})\}$ be the set of pending soft aperiodic tasks.

First we update interval spare capacities according to one of the following three cases.

1. $T_{last} \in S_{soft} \vee T_{last} = idle$:
   $sc(I_c) = sc(I_c) - [t - t_{last} - \max(start(I_c) - t_{last}, 0)]$.
2. $T_{last} \in I_c$:
   $sc(I_c) = sc(I_c) + \max(start(I_c) - t_{last}, 0)$.
3. $T_{last} \in I_j \neq I_c \wedge end(I_j) > end(I_c)$:
   $sc(I_c) = sc(I_c) - [t - t_{last} - \max(start(I_c) - t_{last}, 0)]$.
   $sc(I_j) = sc(I_j) + (t - t_{last})$.
   Interval spare capacities and wake-up points are then updated going from I_j towards I_c.

For case 2 we also update the wake-up point of I_c.

2. $\exists T : T \in I_c \wedge T$ *not completed* $\wedge\ sc(I_c) > 0$:
   $wu(I_c) = t + sc(I_c)$.

If, after updating, $sc(I_c) = 0 \wedge T_{last} \in S_{soft}$ then T_{last} is removed[1] from the ready-queue and put back in S_{soft}. We then check if there are aperiodic tasks pending.

- $S_{firm} \neq \emptyset$: there are firm aperiodic tasks present. We apply the guarantee algorithm to the first task in S_{firm}, if successful to the second one, and so on. Each guaranteed task is inserted into the ready-queue.
- $S_{soft} \neq \emptyset$: there are soft aperiodic tasks present. If, after guaranteeing firm aperiodic tasks, $sc(I_c) > 0$ we insert the first task in S_{soft} at the head of the ready-queue, provided no soft aperiodic task is already in the queue.

When the above steps have been completed the plug-in suspends and consequently the scheduling module takes over.

[1] The task is only removed if it has not yet completed its execution. If completed, the task is no longer present in the queue.

3.5 Target System Diversity and Plug-In Applicability

If no aperiodic tasks are present in the system, the plug-in will not make any modifications to the ready-queue. Thus it will not interfere with the original scheduler.

Depending on the target system, the plug-in needs to be initialised in different ways. For an event-triggered system where the scheduling module processes a set of tasks characterised by earliest start times, maximum execution times, and deadlines, it is straightforward to create a corresponding set of intervals, as described in section 3.2.

In an event-triggered system using the EDF [4] scheduling algorithm, the plug-in is set up in the off-line phase and during run-time it will be activated by the dispatcher when tasks finish their execution, when aperiodic tasks arrive, and at interval wake-up points.

A target system using an off-line generated [6] schedule usually has more stringent task requirements, such as precedence constraints, than an on-line scheduled, event-triggered counterpart. In an off-line generated schedule, tasks have fixed starting and finishing times. Thus before setting up the plug-in, the off-line schedule needs to be transformed into a task set in which tasks are distinguished by earliest start times, deadlines, and maximum execution times. Such a transformation technique can be found in [1]. The plug-in off-line initialisation phase then continues. At run-time the plug-in will be activated in the same way as for an event-triggered system.

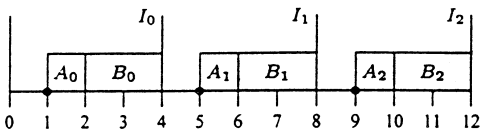
4 An Example

To illustrate how the plug-in works, we provide the following example.

Assume a set $S = \{A_0, B_0, A_1, B_1, A_2, B_2\}$ of six task instances scheduled on a system by the EDF algorithm. Each task A_i, $B_i \in S$, where $i = 0, \ldots, 2$, has the following properties.

- $maxt(A_i) = 1$.
- $maxt(B_i) = 2$.
- $est(A_i) = est(B_i) = 4i$.
- $dl(A_i) = dl(B_i) = 4(i+1)$.

For this task set, three intervals I_0, I_1, and I_2 are created. These intervals are shown in figure 3. Plug-in run-time activities and a trace of task executions are given in figure 4.



Interval	$start(I)$	$end(I)$	$\|I\|$	$maxt(I)$	$sc(I)$	$wu(I)$
I_0	0	4	4	3	1	1
I_1	4	8	4	3	1	5
I_2	8	12	4	3	1	9

Figure 3. The intervals and their initial attributes.

t	R	S_{firm}	S_{soft}	Plug-in Actions
0	$\{A_0, B_0\}$	$\emptyset$	$\emptyset$	None
1	$\{B_0\}$	$\emptyset$	$\{Ta_s\}$	Case 2 $\Rightarrow$ $sc(I_0) = 1, wu(I_0) = 2, R = \{Ta_s, B_0\}$
2	$\{Ta_s, B_0\}$	$\emptyset$	$\emptyset$	Case 1 $\Rightarrow$ $sc(I_0) = sc(I_0) - 1 = 0, R = \{B_0\}$
4	$\{A_1, B_1\}$	$\emptyset$	$\{Ta_s\}$	$R = \{Ta_s, A_1, B_1\}$
5	$\{A_1, B_1\}$	$\emptyset$	$\emptyset$	Case 1 $\Rightarrow$ $sc(I_1) = sc(I_1) - 1 = 0$
6	$\{B_1\}$	$\emptyset$	$\emptyset$	Case 2 $\Rightarrow$ $sc(I_1) = 0$
8	$\{A_2, B_2\}$	$\{Ta_f\}$	$\emptyset$	$guarantee(Ta_f), R = \{Ta_f, A_2, B_2\}$

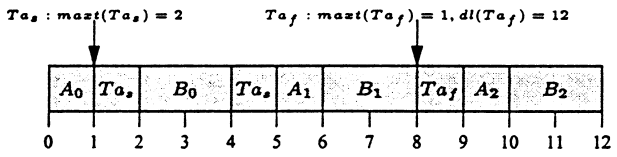


Figure 4. Execution trace and plug-in actions.

At time $t = 0$ the ready-queue $R = \{A_0, B_0\}$. No aperiodics are present and the plug-in suspends itself. As shown in figure 4, the EDF scheduler selects task A_0 for execution.

At time $t = 1$, a soft aperiodic task $Ta_s : maxt(Ta_s) = 2$ arrives at the system. The plug-in updates the spare capacity and wake-up point of I_0 according to case 2. Since $sc(I_0) > 0 \wedge S_{soft} = \{Ta_s\}, Ta_s$ is inserted at the head of the ready-queue and is executed. When the plug-in wakes up at time $t = 2$, Ta_s is preempted in favour of task B_0 which will miss its deadline unless it begins executing at this time.

At time $t = 8$, a firm aperiodic task $Ta_f : maxt(Ta_f) = 1, dl(Ta_f) = 12$ arrives at the system. Since at this time $sc(I_2) = 1$, the task is guaranteed and inserted into the ready-queue. From this point on until the start of the next schedule instance the plug-in makes no modifications to the ready-queue.

5 Conclusions and Further Work

In this paper we addressed the need for adding functionality to systems, in particular scheduling algorithms, without need for abandoning trusted methods or major revisions. We proposed a "plug-in" approach for aperiodic task handling. Our method concentrates the aperiodic task functionality into a software module with a defined interface. As the functionality of the plug-in is independent of the basic scheduling scheme and the interface is very small, we can insert and apply the "aperiodic plug-in" to both off-line and online scheduling methods. We are currently implementing the plug-in for use in various systems with diverse scheduling algorithms. Further research will go into extending the applicability to a wider range of systems and algorithms.

6 Acknowledgements

The authors wish to thank Sasi Punnekkat, Damir Isović, and Anders Ingelsson for their useful comments.

References

[1] G. Fohler. *Flexibility in Statically Scheduled Hard Real-Time Systems*. PhD thesis, Technische Universität Wien, Apr. 1994.

[2] G. Fohler. Joint scheduling of distributed complex periodic and hard aperiodic tasks in statically scheduled systems. In *Proceedings of the 16th Real-Time Systems Symposium*, Pisa, Italy, Dec. 1995.

[3] D. Isović and G. Fohler. Handling sporadic tasks in off-line scheduled distributed real-time systems. In *Proceedings of the 11th Euromicro workshop on real-time systems*, York, England, Jun. 1999.

[4] C. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM, 20, 1*, pages 46–61, Jan. 1973.

[5] P. Puschner and C. Koza. Calculating the maximum execution time of real-time programs. *RT Systems, 1(2)*, pages 159–176, Sep. 1989.

[6] K. Ramamritham. Allocation and scheduling of complex periodic tasks. In *Proceedings of the 10th Int. Conf. on Distributed Computing Systems*, pages 108–115, 1990.

[7] S. Ramos-Thuel and J. P. Lehoczky. On-line scheduling of hard deadline aperiodic tasks in fixed priority systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, Raleigh-Durham, North Carolina, Dec. 1993.

[8] M. Spuri and G. Buttazzo. Efficient aperiodic service under earliest deadline scheduling. In *Proceedings of the 15th IEEE Real-Time Systems Symposium*, Portorico, Dec. 1994.

[9] M. Spuri and G. Buttazzo. Scheduling aperiodic tasks in dynamic priority systems. *The Journal of Real-Time Systems*, 10(2):179–210, Mar. 1996.