Universite de Bretagne
Occidentale

Implementation of aperiodic server scheduler in the cheddar tool.

Scheduling methods for aperiodic tasks

Author :
Cyril Muller

Tutor :
M. Frank Singhoff

February 28, 2014

Contents

Introduction

The scheduling of aperiodic tasks is very different than with periodic tasks in the way that by definition aperiodic tasks' behavior is unpredictable however such tasks still needs to be treated. There exist many way to deal with aperiodic tasks with different level of efficiency. The method we will explore in this paper is called the aperiodic server.

An aperiodic server is a task specially created to service aperiodic tasks. This server can be activated periodically or when aperiodic requests occur depending on the server's parameters. If there are several aperiodic tasks pending, they will be serviced according to their starting time (FIFO). The server can be scheduled with the same algorithm as the other periodic tasks.

We have found four main implementations of aperiodic servers, the polling server, the deferrable server, the sporadic server and the priority exchange server. In this paper we will see the implementation of the polling server, the deferrable server and the sporadic server in the Cheddar[4] software.

The rest of the paper will be organized as follow: a presentation of the possible aperiodic servers with examples, then the implementation in Cheddar and the different tests and finally a conclusion of the work done.

1 Aperiodic servers

In this section we will see four implementations for aperiodic servers with some examples and diagrams. First, we will focus on the polling server, then on deferrable, then on the sporadic server and finally on the priority exchange server.

1.1 Polling server

The polling server is a periodic task with a period T_s, a capacity C_s and the highest priority. Every server's activation, it checks if there are any pending aperiodic tasks, if there are, the server uses its capacity to service them until either the task is finished or the server's capacity is depleted. However, if there is no pending aperiodic task, the server remains idle until its next activation which means that even if an aperiodic request occurs in the middle of the server's servicing time, the request will not be treated until the next period as the server will already be inactive.
This is the simplest implementation of an aperiodic server but the aperiodic tasks tend to have a relatively long response time (depending on the server's period), also when there is no task to service, the aperiodic servicing time is lost.

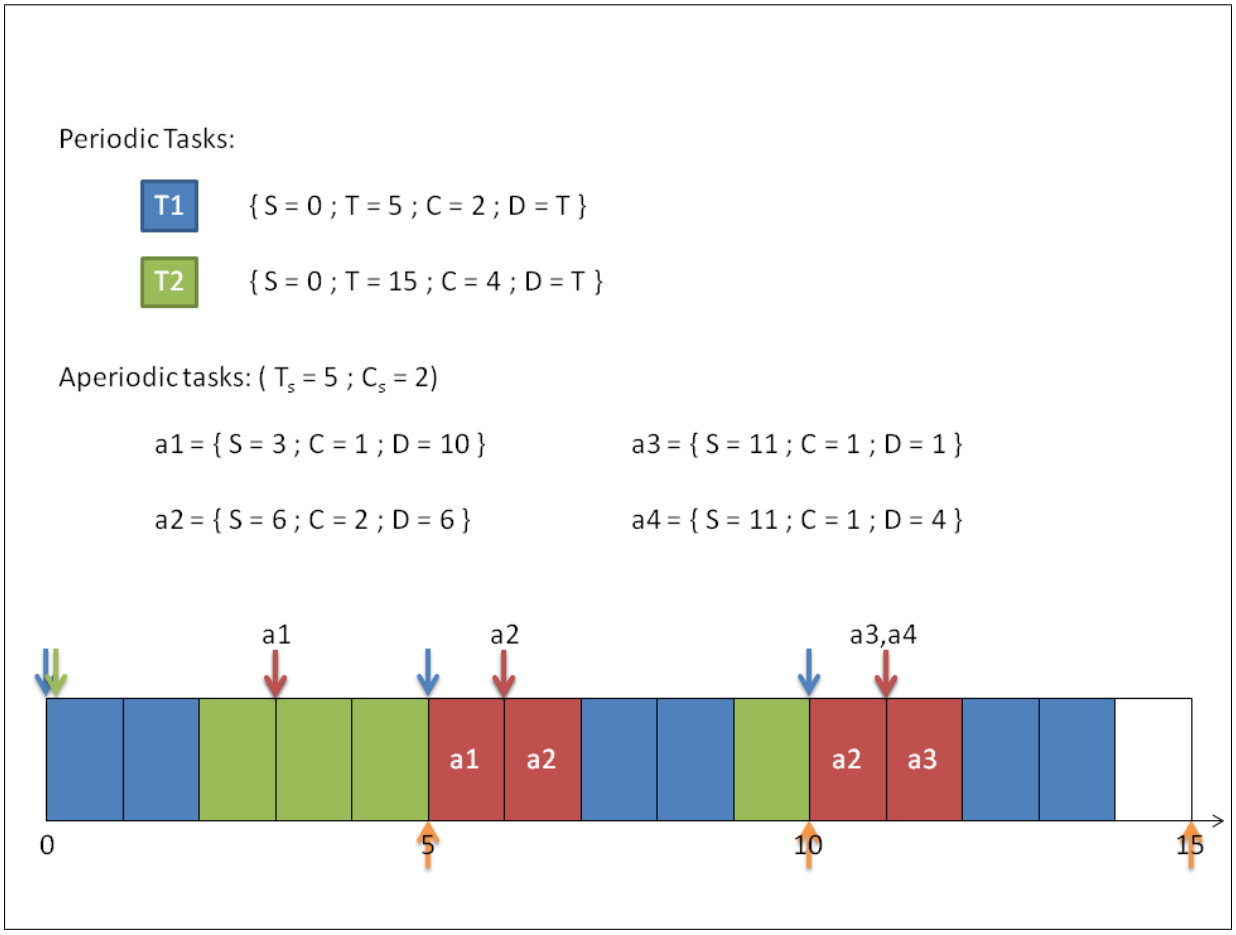

Figure 1: Polling server example with two periodic tasks and four aperiodic requests

In the figure 1, there are two periodic tasks *T1* and *T2* and four aperiodic requests *a1*, *a2*, *a3* and *a4*. We can see that the aperiodic request *a2* is serviced for one time unit during the first polling period and is finished at the tenth time unit when the server awakes. The task *a4* was not serviced during the hyper period because the polling server did not have enough capacity. It will be serviced at the fifteenth time unit though.

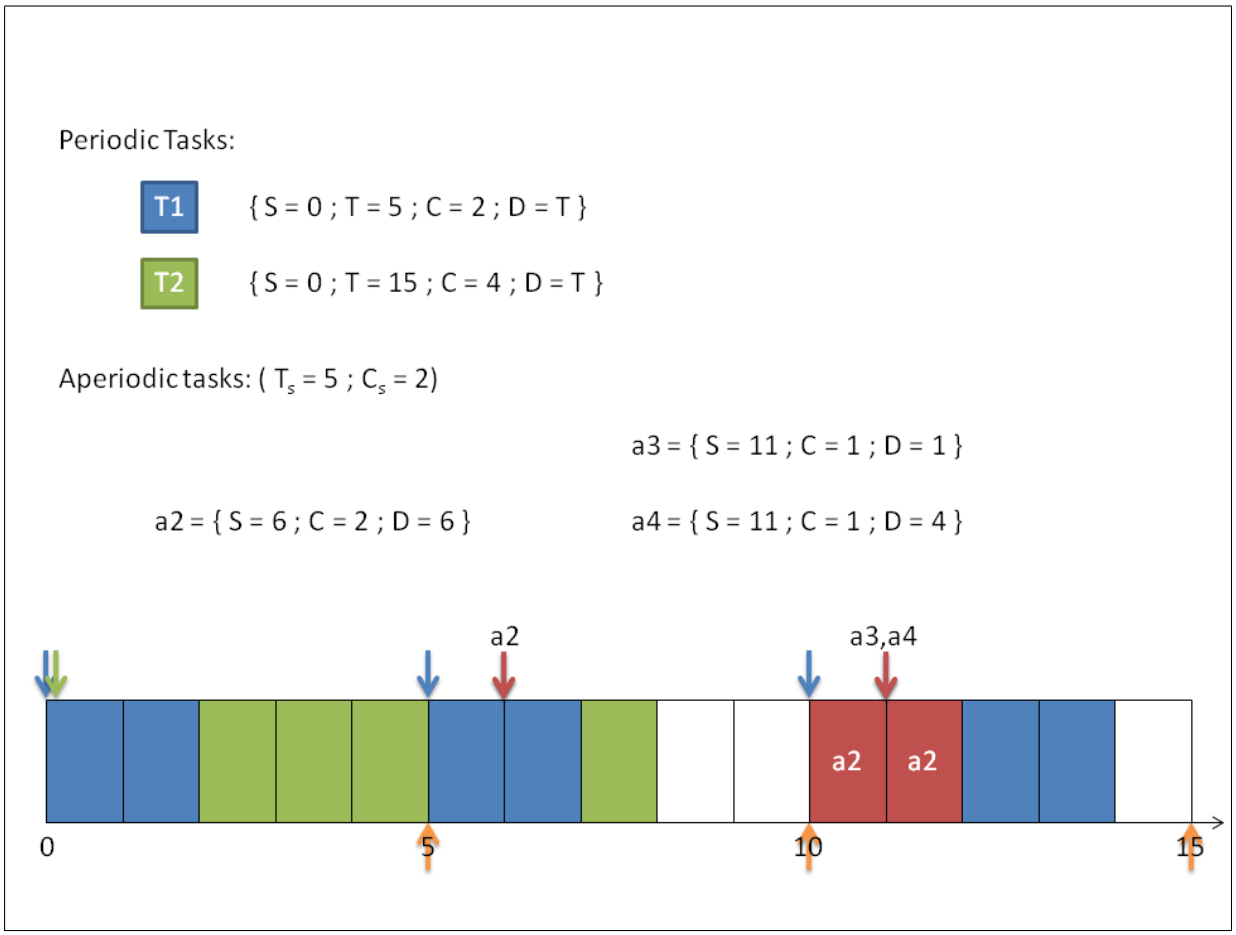

Figure 2: Polling server example when an aperiodic request occurs in the middle of the server's servicing time and is not serviced

In the figure 2 we change the previous tasks set by removing the aperiodic request *a1*. Now we can see that the task *a2* occurs in the middle of the time assigned to the server for servicing aperiodic requests but was serviced at the tenth time unit. This is because at the fifth time unit, the polling server checked for pending aperiodic request and as there was none fell back to sleep until the tenth time unit.

1.2 Deferrable server

As the polling server, the deferrable server has a capacity C_s and a period T_s and the highest priority. However, the deferrable server is not periodically activated to service aperiodic tasks but when an aperiodic task is pending and the server has capacity superior to 0. The server then services the task(s) until depletion of its capacity or the aperiodic task is finished. The server's capacity is restored periodically every T_s. If an aperiodic task occurs when the server has depleted its capacity, this task will be serviced when the server will have replenished its capacity.
Unlike the polling server, the deferrable server saves its capacity for a longer period when not used and also provides a better response time than the polling server as it is potentially able to immediately service aperiodic request. However in some cases, the server might take up to $2*C_s$ time units servicing aperiodic tasks which may impose a delay too long on periodic task as the server has the highest priority.

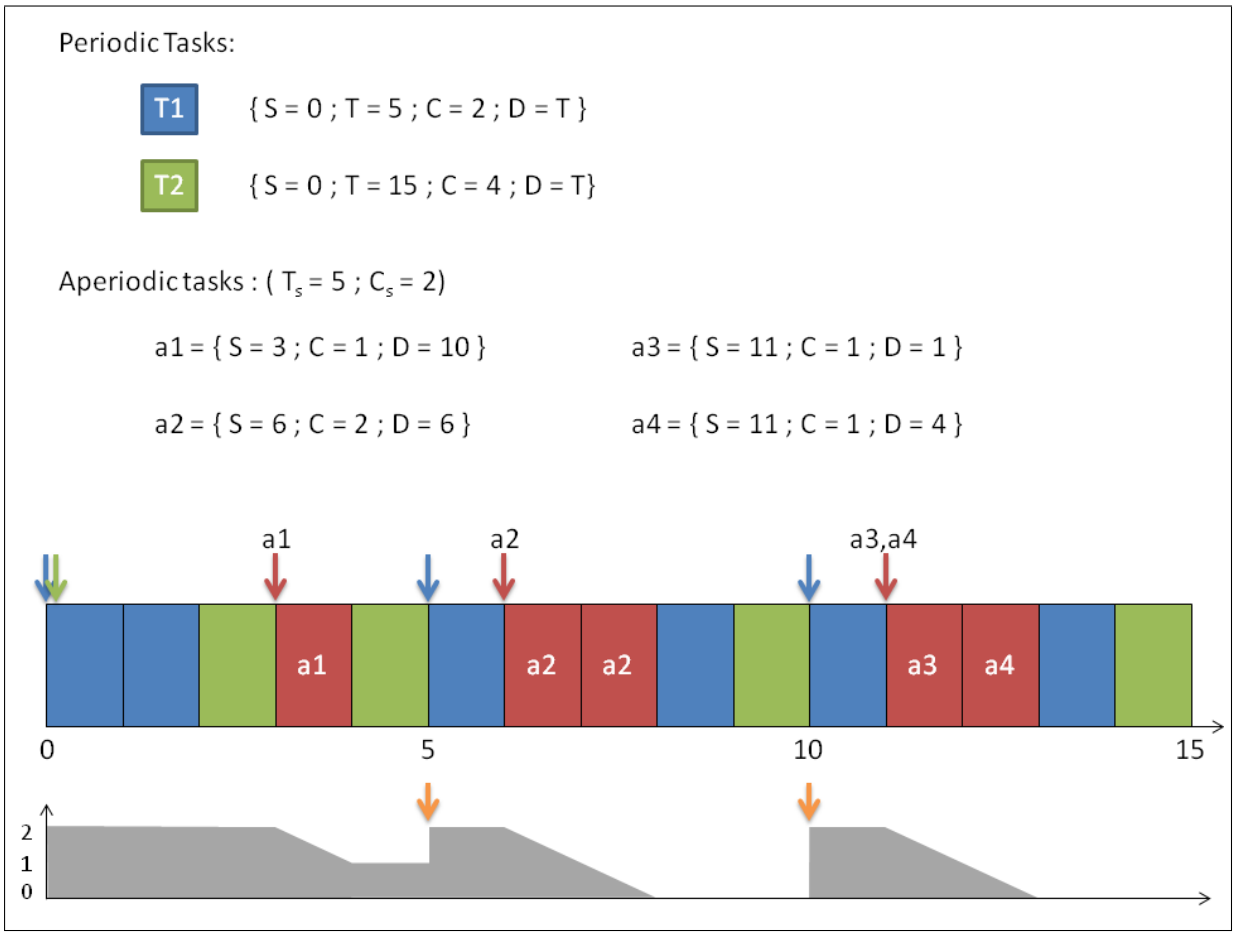

Figure 3: Deferrable server example with two periodic tasks and four aperiodic requests

In figure 3, we have once again two periodic tasks *T1* and *T2* and four aperiodic requests *a1*, *a2*, *a3* and *a4*. When *a1* occurs, it is immediately serviced by the deferrable server which then has its capacity drop from two to one. As the server has a period of five, at the fifth time unit, its capacity is refilled and it is then ready to service *a2* at the sixth time unit. After servicing *a2*, the server capacity is empty because *a2* also had a capacity of two. At the eleventh time unit, *a3* and *a4* occur, we consider that *a3* is serviced first, while the server had replenished its capacity at the tenth time unit, allowing it to service the request in a row.

5

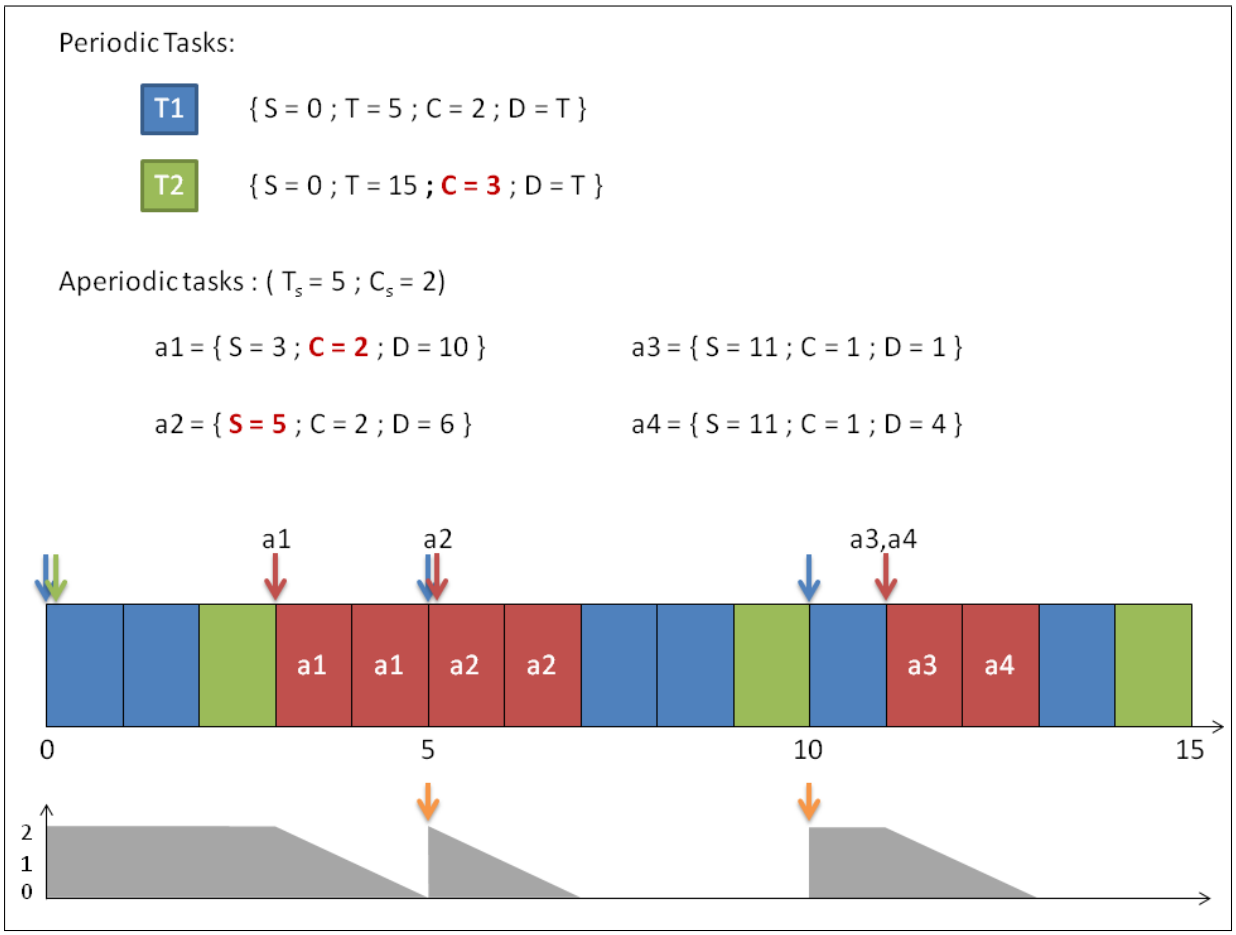

Figure 4: Deferrable server example when the server uses 2*C_s time units

In the example of the figure 4, we have modified the previous tasks set to highlight the fact that the deferrable server can use up to two times its capacity. We can see that after *a1* has been serviced, *a2* occurs right after the server had refilled its capacity making *a2* being serviced immediately and the deferrable server using four time units in a row with a capacity of two.

1.3 Sporadic server

The sporadic server works similarly to the deferrable server at the exception of when the server's capacity is refilled. In the sporadic server's case, the capacity is refilled after a specified delay (T_s) after its activation. The server is activated when an aperiodic request occurs and the server has a full capacity.
This implementation has the advantages that the server will never use 2*C_s and that the periodic tasks' deadlines are guaranteed accordingly to the rate monotonic algorithm. On the other hand this is also a more complicated design.

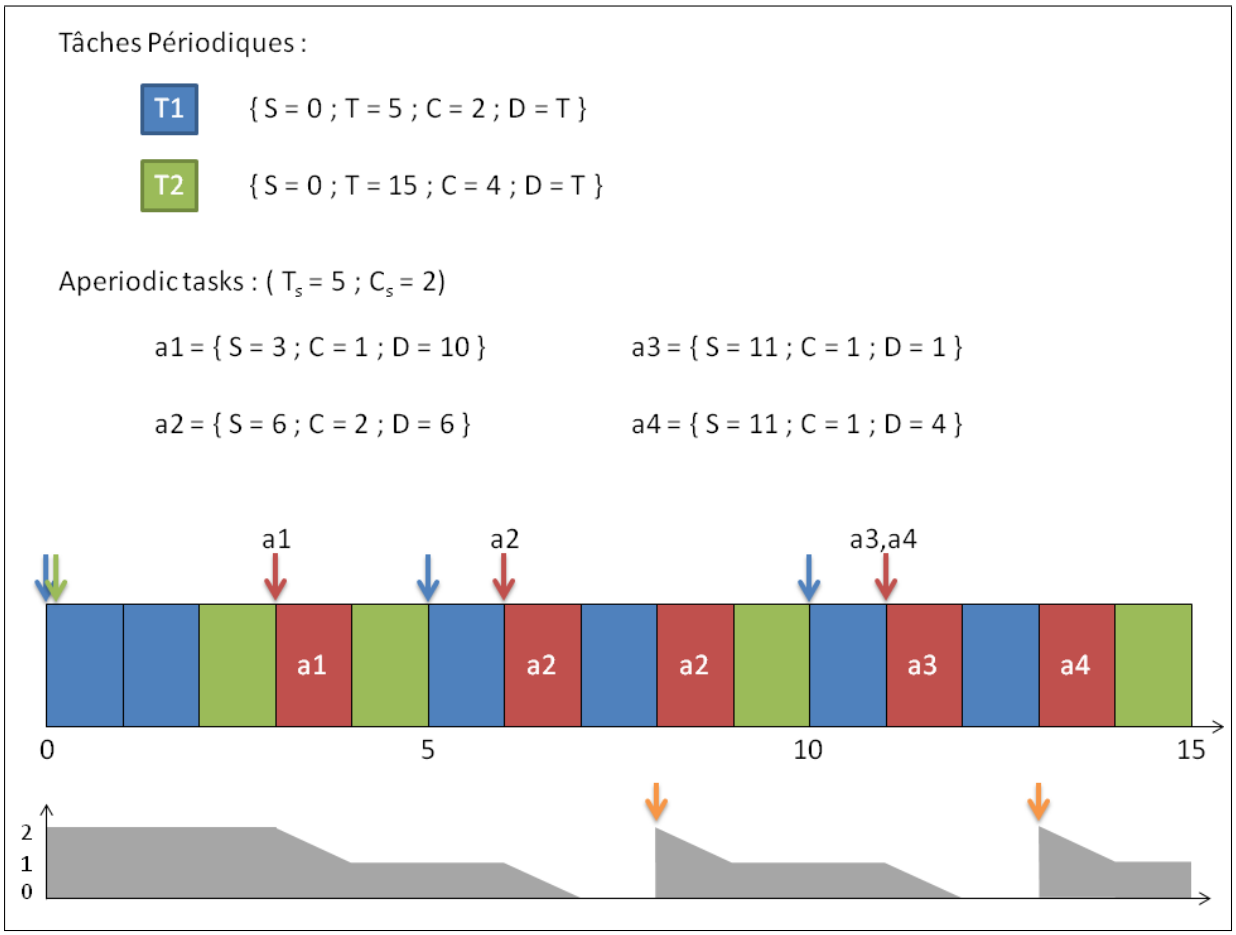

Figure 5: Sporadic server example with two periodic tasks and four aperiodic requests

In the figure 5, when *a1* occurs, the sporadic server's capacity is full, the server is then activated and compute the next time for refilling its capacity, in this case the eighth time unit. *a1* is immediately serviced. However, when *a2* occurs, the server's capacity is only at one, so *a2* is only executed for one time unit and must wait until the sporadic server has a full capacity again, which happen at the eighth time unit. The same happened with *a4*, as when it occurred the sporadic server had exhausted its capacity and so the aperiodic request had to wait.

7

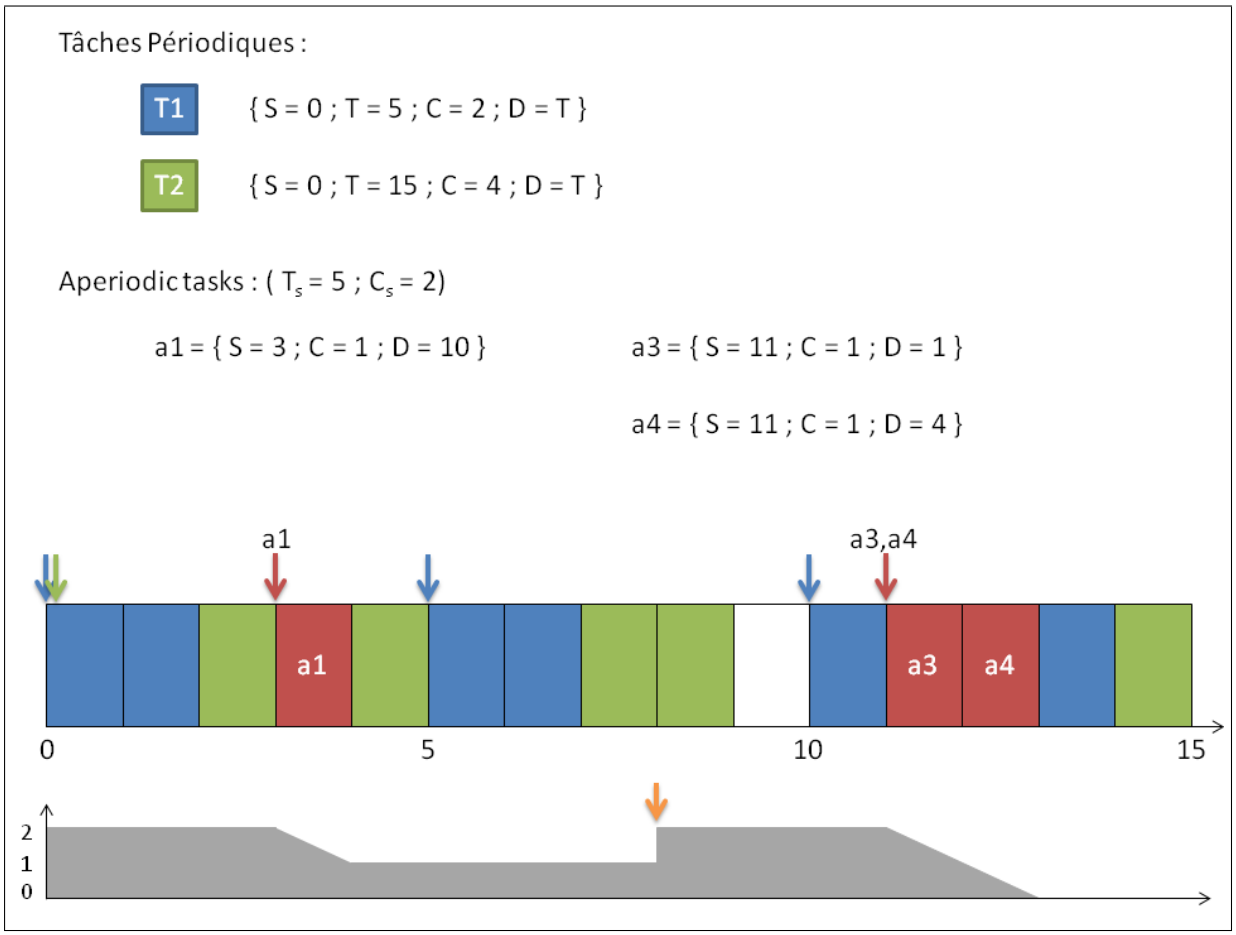

Figure 6: Sporadic server example with two periodic tasks and three aperiodic requests

In the example of the figure 6, we modified the tasks set from the previous example by removing *a2*. This example shows that the sporadic server does not have a period but a minimal delay until its capacity is refilled. We can see at the eighth time unit, the server refill its capacity, as *a1* occurred at the third time unit and will remain idle with a full capacity until *a3* and *a4* occurs. As *a3* and *a4* occurred at the eleventh time unit, the sporadic the server will refill its capacity at the sixteenth time unit.

1.4 Priority exchange server

The priority exchange server saves its capacity by exchanging it to a lower priority execution time every time a switch to a periodic task occurs. So the server will have as many capacity level as there are periodic priority level plus one more for the server higher priority. The server will then service aperiodic requests if its capacity is not exhausted and its priority is eligible by using the capacity at the highest priority level available. In case there is a priority tie between an aperiodic and a periodic task, the aperiodic task will be prioritized. As the server is executed, there could be an accumulation of unused capacity at the lowest priority level. The low level capacity will be emptied when there will be no task to serviced, periodic and aperiodic.
However this server has the highest complexity.

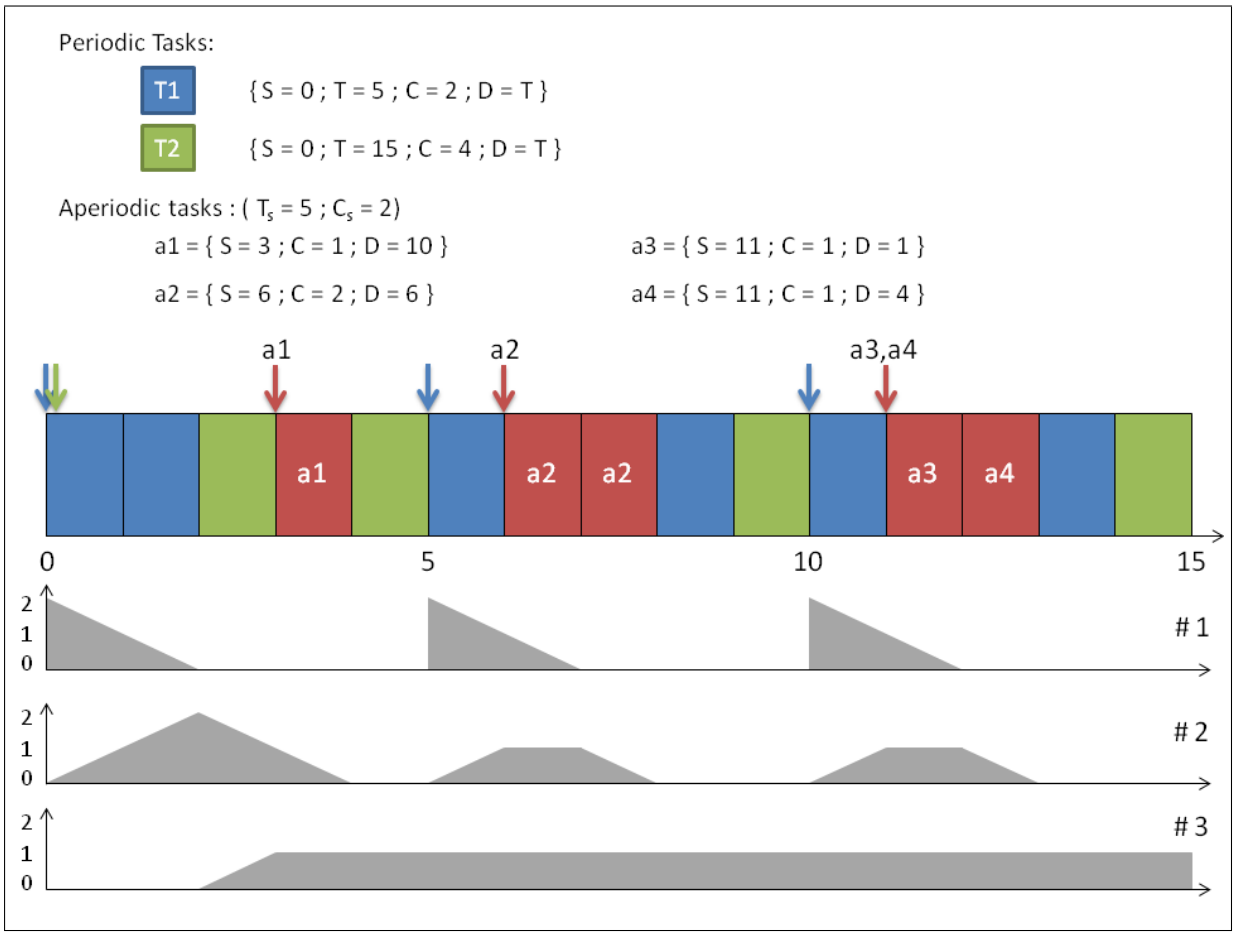

Figure 7: Priority exchange server example with two periodic tasks and four aperiodic requests

For the example of the figure 7, we can see that the server has three capacity level. The first one is the high priority capacity for the server and second an third correspond to the priority level of the tasks *T1* and *T2*. At the starting time, the high priority capacity of the server is refilled, but as there is no pending aperiodic request, the task *T1* is serviced and the server then exchange the capacity from the level one with the level two. At the third time unit another switch between tasks occurs and so the server's capacity of the level two is exchanged down with the level three that correspond to *T2*'s priority.

The aperiodic task *a1* is serviced with the level three capacity and as aperiodic request are serviced before periodic task, *a1* is serviced before *T2*. The task *a2* is partially serviced with the high priority server's capacity and finished with the level two level.

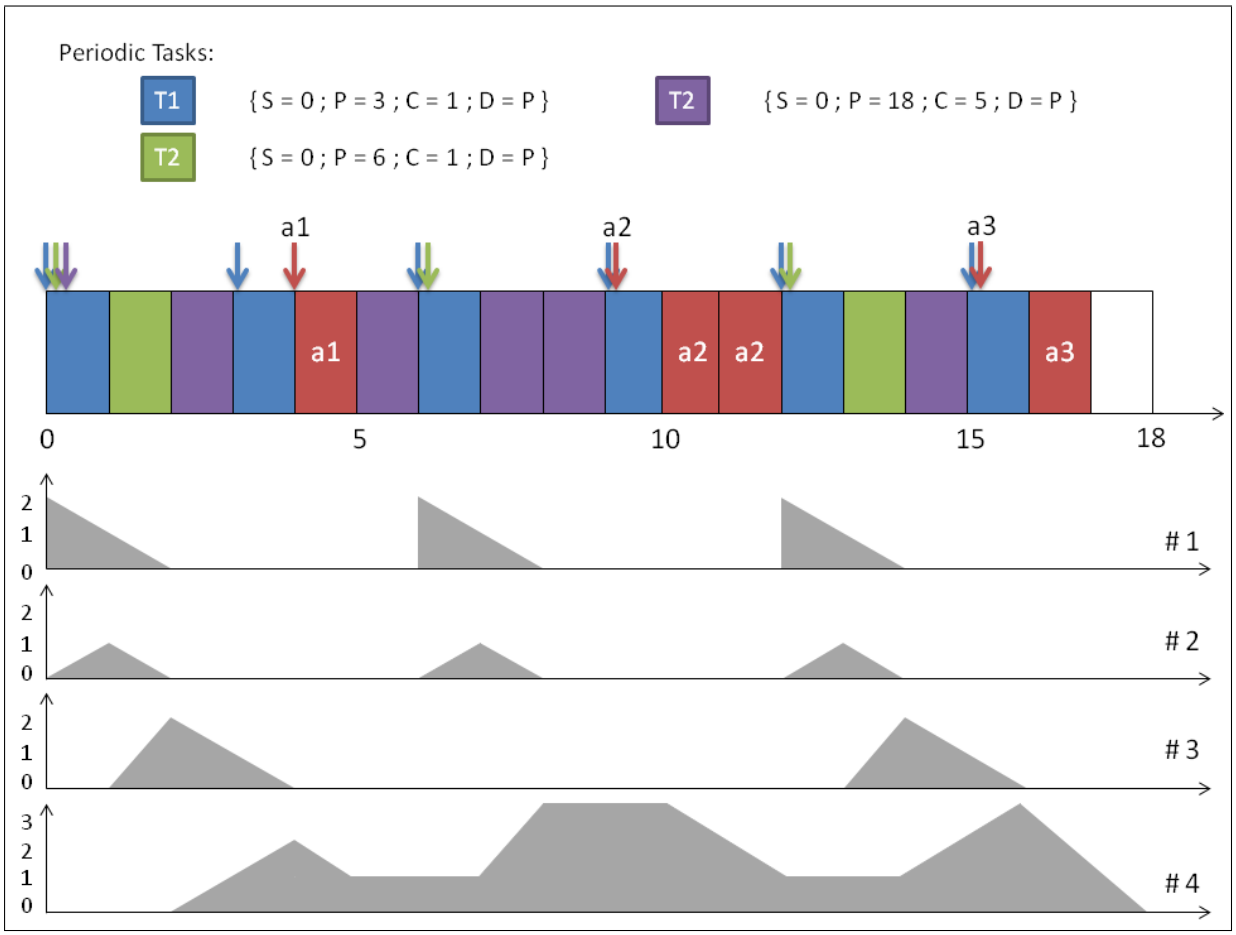

Figure 8: Another priority exchange server

In the figure 8, we have three periodic tasks and also three aperiodic requests:

- *a1*: S = 4 ; C = 1 ; D = 10
- *a2*: S = 9 ; C = 2 ; D = 6
- *a3*: S = 15 ; C = 1; D = 2

First, we can note that *a2* and *a3* are serviced after *T1* because, at the moment when they occurred, the available capacity was at a lower level than *T1*'s capacity. Also, at the starting time, as there was no aperiodic tasks to serve, the high priority execution time is exchanged down with *T1* and at the first time unit, another exchange happens with *T2*, and again with *T3* at the third time unit. So, we end up with all the capacity at the lowest capacity level at the fourth time unit right when *a1* occurs. The which was serviced immediately as the only other pending periodic task was *T3* and in case of priority tie the aperiodic task wins.

2 Implementation

In this section, we will first see the Cheddar software and how the scheduler framework works, then the different test cases for the servers implementations, then the algorithm implemented in the framework which are the polling, the deferrable and the sporadic. The priority exchange server is not yet implemented. Finally the tests of the different servers implemented.

2.1 Cheddar

Cheddar is real time scheduling analysis tool offering many solutions for scheduling. The program is developed in Ada language.

To implement the aperiodic servers, we will use the package *scheduler.fixed_priority.aperiodic_server*. This package will contain variables common to all the servers : the current capacity (*current_server_capacity*) and a wake time that represents the period (*current_server_wake_time*), it also chose the periodic and aperiodic tasks with the highest priority every tick. This package also receive the scheduler parameters, such as the default capacity (*server_capacity*) used to refill *current_server_capacity* and the server's period (*server_period*).

Then for each server, we will implement another package based on the *scheduler.fixed_priority.aperiodic_server*, where the server will decide for each tick whether the previously selected periodic or aperiodic tasks should be executed. This package also updates the variables *current_server_capacity* and *current_server_wake_time* as each server has its own way to deal with them.
The sporadic_server package has one additional variable *server_is_idle* used to compute the next server's wake time as the sporadic server is not periodically activated.

2.2 Test sets

In order to test if the future implementation of the aperiodic servers, we need to create the test cases corresponding to the tasks sets used in the section 1 Aperiodic servers. The test sets will be written in the Architecture Description Language (ADL)[5] in the form of XML files in which we describe the hardware architecture where the tasks will be ran and of course the different tasks themselves. Following is the example for a polling server using the tasks set from the figure 1. We will only detailed the relevant parts to our implementation which are the scheduler and the tasks.

Processor

```
<core_units>
 <core_unit id="id_1">
  <object_type>CORE_OBJECT_TYPE</object_type>
  <name>core1</name>
  <scheduling>
   <scheduling_parameters>
    <scheduler_type>HIERARCHICAL_POLLING_APERIODIC_SERVER_PROTOCOL</scheduler_type>
    <quantum>0</quantum>
    <preemptive_type>PREEMPTIVE</preemptive_type>
    <capacity>2</capacity>
    <period>5</period>
    <priority>10</priority>
```

```
    <start_time>0</start_time>
   </scheduling_parameters>
  </scheduling>
  <speed>0.00000</speed>
 </core_unit>
</core_units>
```

The type of scheduler is bound to a processor core, in our case we will only use one mono core processor. The scheduler parameters such as the period or the capacity are defined amongst the core parameters.

Tasks

```
<tasks>
 <periodic_task id="id_4">
  <object_type>TASK_OBJECT_TYPE</object_type>
  <name>t1</name>
  <task_type>PERIODIC_TYPE</task_type>
  <cpu_name>p1</cpu_name>
  <address_space_name>as_p1</address_space_name>
  <capacity>2</capacity>
  <deadline>5</deadline>
  <start_time>0</start_time>
  <priority>2</priority>
  <blocking_time>0</blocking_time>
  <policy>SCHED_FIFO</policy>
  <text_memory_size>0</text_memory_size>
  <stack_memory_size>0</stack_memory_size>
  <criticality>0</criticality>
  <context_switch_overhead>0</context_switch_overhead>
  <period>5</period>
  <jitter>0</jitter>
 </periodic_task>

 <aperiodic_task id="id_6">
  <object_type>TASK_OBJECT_TYPE</object_type>
  <name>a1</name>
  <task_type>APERIODIC_TYPE</task_type>
  <cpu_name>p1</cpu_name>
  <address_space_name>as_p1</address_space_name>
  <capacity>1</capacity>
  <deadline>10</deadline>
  <start_time>3</start_time>
  <priority>1</priority>
  <blocking_time>0</blocking_time>
  <policy>SCHED_FIFO</policy>
  <text_memory_size>0</text_memory_size>
  <stack_memory_size>0</stack_memory_size>
  <criticality>0</criticality>
  <context_switch_overhead>0</context_switch_overhead>
 </aperiodic_task>
```

Periodic and Aperiodic tasks are two separate entities with the same parameters at the exception of the period which is not needed for an aperiodic task.

2.3 Algorithms

Here is the list of the data used for the different algorithms

- Current_Time: current execution time
- current_server_wake_time: time when the server will refill its capacity
- current_server_capacity: the current server's capacity
- server_capacity: server's capacity parameter
- server_period: server's period parameter
- server_is_idle: sporadic server's variable to know when the server is activated and so when to compute the next capacity recharge time

Polling server

Here is the algorithm implemented in Cheddar for the polling server.

Algorithm 1: Polling server algorithm

```
Data: Current_Time, current_server_wake_time, server_capacity, server_period
Result: elected: a task
begin
    if Current_Time >= current_server_wake_time and
    Current_Time < current_server_wake_time + server_capacity then
        if There is a pending aperiodic request then
            elected := aperiodic_task;
        else
            current_server_wake_time := current_server_wake_time + server_period;
            if There is a pending periodic task then
                elected := periodic_task;
            end
        end
    else
        if There is a pending periodic task then
            elected := periodic_task;
        end
    end
end
```

Deferrable server

Here is the algorithm implemented in Cheddar for the deferrable server.

Algorithm 2: Deferrable server algorithm

```
Data: Current_Time, current_server_capacity, current_server_wake_time,
      server_capacity, server_period
Result: elected: a task
begin
    /* Refill the server's capacity and compute the next wake time        */
    if Current_Time = current_server_wake_time then
        current_server_capacity := server_capacity;
        current_server_wake_time := current_server_wake_time + server_period;
    end
    /* Election                                                           */
    if current_server_capacity > 0 and there is a pending aperiodic task then
        elected := aperiodic_task;
        current_server_capacity := current_server_capacity - 1;
    else
        if There is a pending periodic task then
            elected := periodic_task;
        end
    end
end
```

Sporadic server

Here is the algorithm implemented in Cheddar for the sporadic server.

Algorithm 3: Sporadic server algorithm

```
Data: Current_Time, current_server_capacity, current_server_wake_time,
      server_is_idle, server_capacity, server_period
Result: elected: a task
begin
    /* Refill the server's capacity and put it back into idle mode        */
    if Current_Time = current_server_wake_time then
        current_server_capacity := server_capacity;
        server_is_idle := True;
    end
    /* Election                                                           */
    if current_server_capacity > 0 and there is a pending aperiodic task then
        elected := aperiodic_task;
        current_server_capacity := current_server_capacity - 1;
        if server_is_idle = True then
            server_is_idle := False;
            current_server_wake_time := Current_Time + server_period;
        end
    else
        if There is a pending periodic task then
            elected := periodic_task;
        end
    end
end
```

2.4 Testing

In this section we will focus on the test results of the previous algorithms in Cheddar. Firstly, the result of the polling server, then the deferrable server and finally the sporadic server.

Polling server

Here are the test result for the polling server (screen captures from Cheddar).

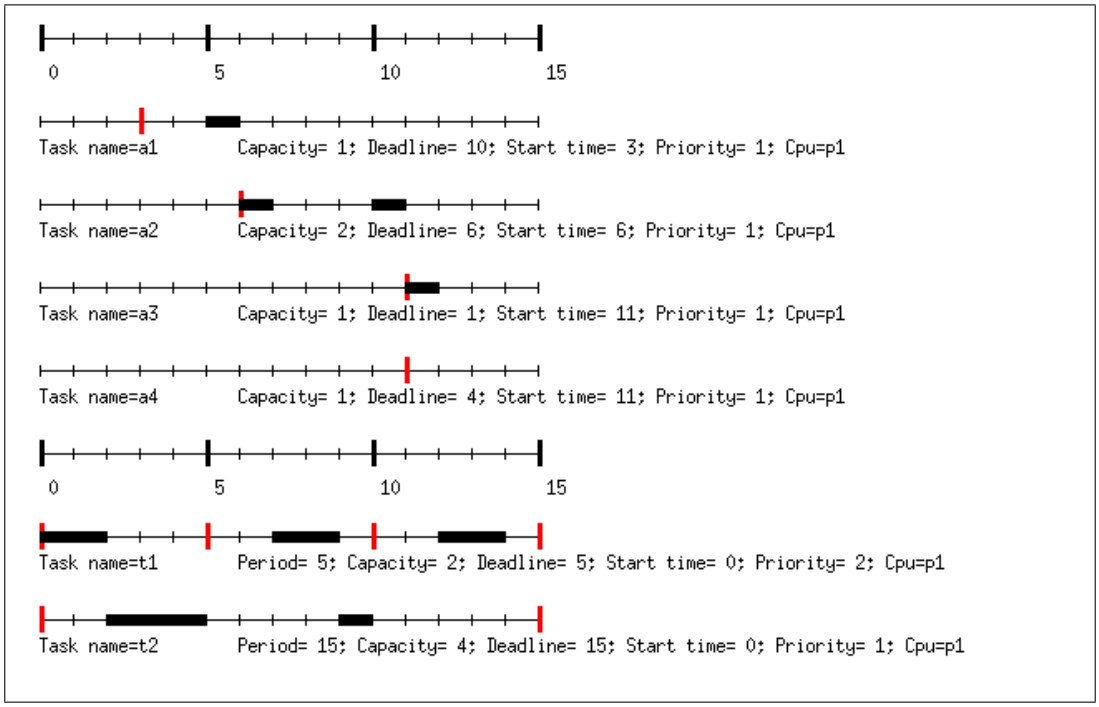

Figure 9: First polling server test

The figure 9 is the test result for the polling server with tasks set from the figure 1.

The figure 10 is the test result for the polling server with tasks set from the figure 2.

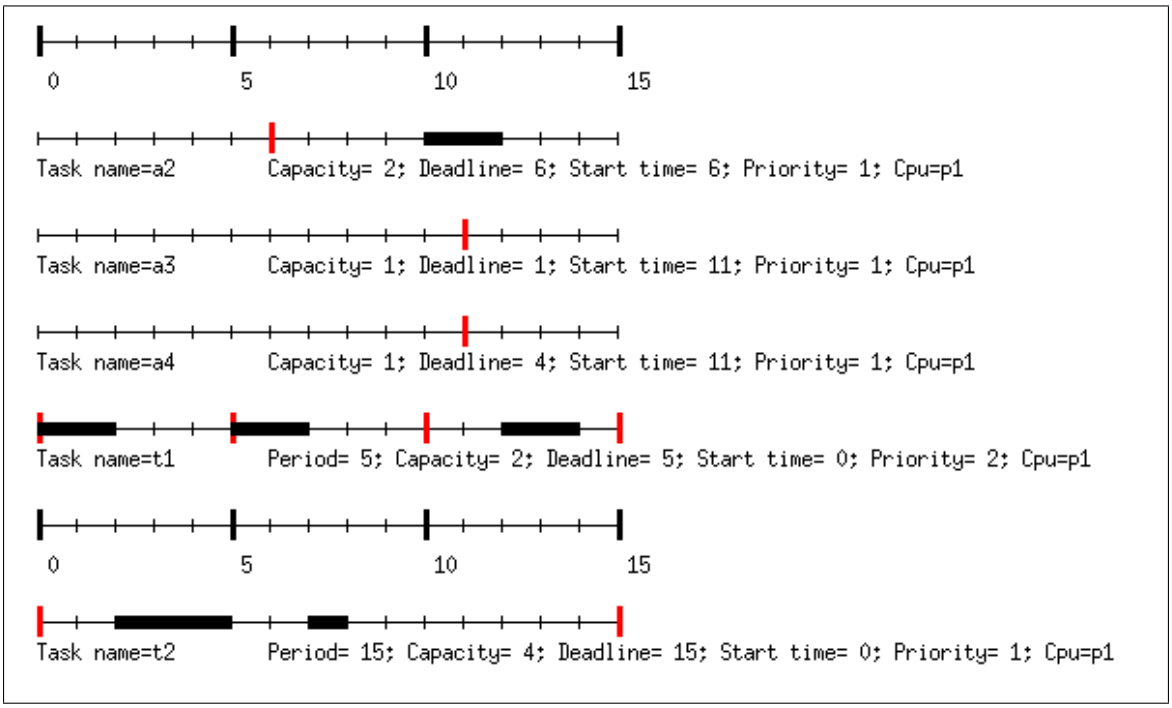

Figure 10: Second polling server test

16

Deferrable server

Here are the test result for the deferrable server (screen captures from Cheddar).

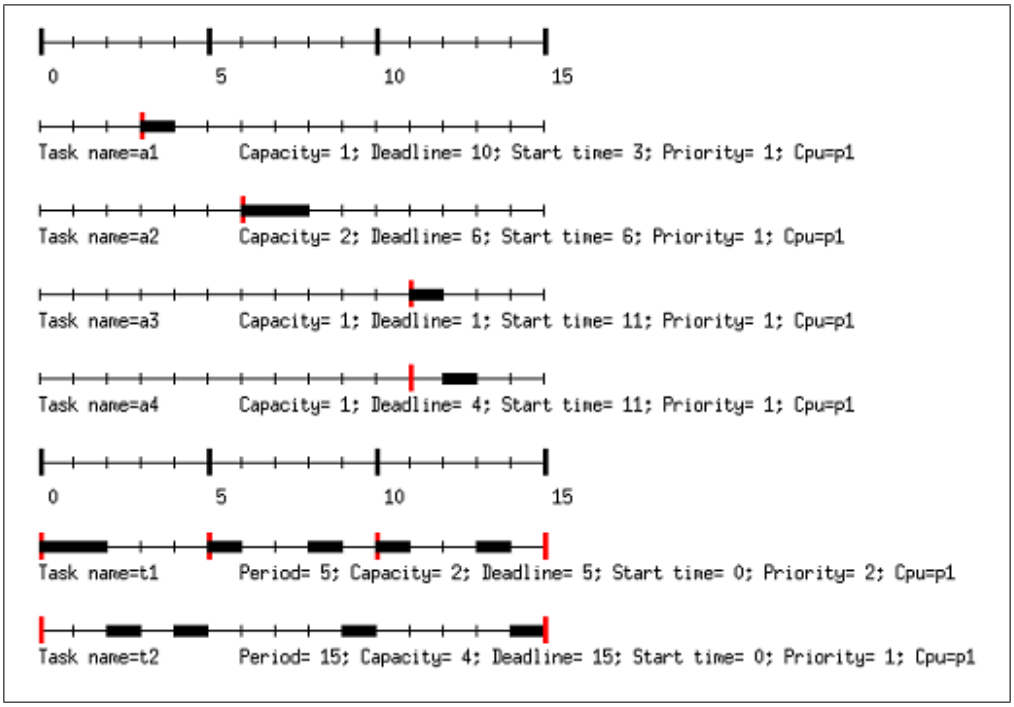

Figure 11: First deferrable server test

The figure 11 is the test result for the polling server with tasks set from the figure 3.

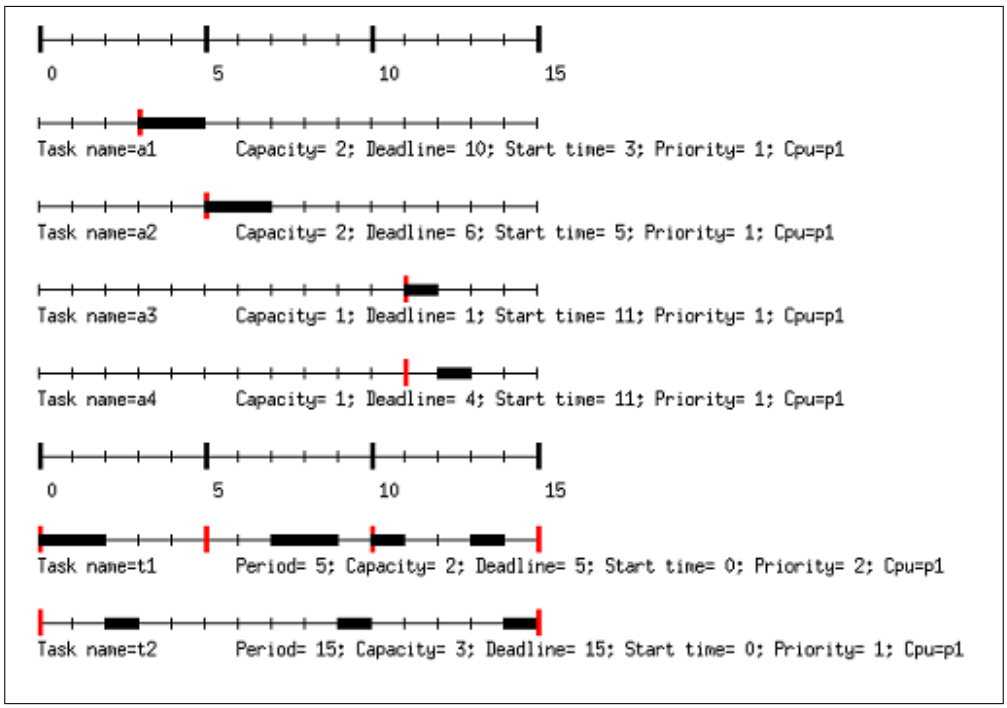

Figure 12: Second deferrable server test

The figure 12 is the test result for the polling server with tasks set from the figure 4.

17

Sporadic server

Here are the test result for the sporadic server (screen captures from Cheddar).

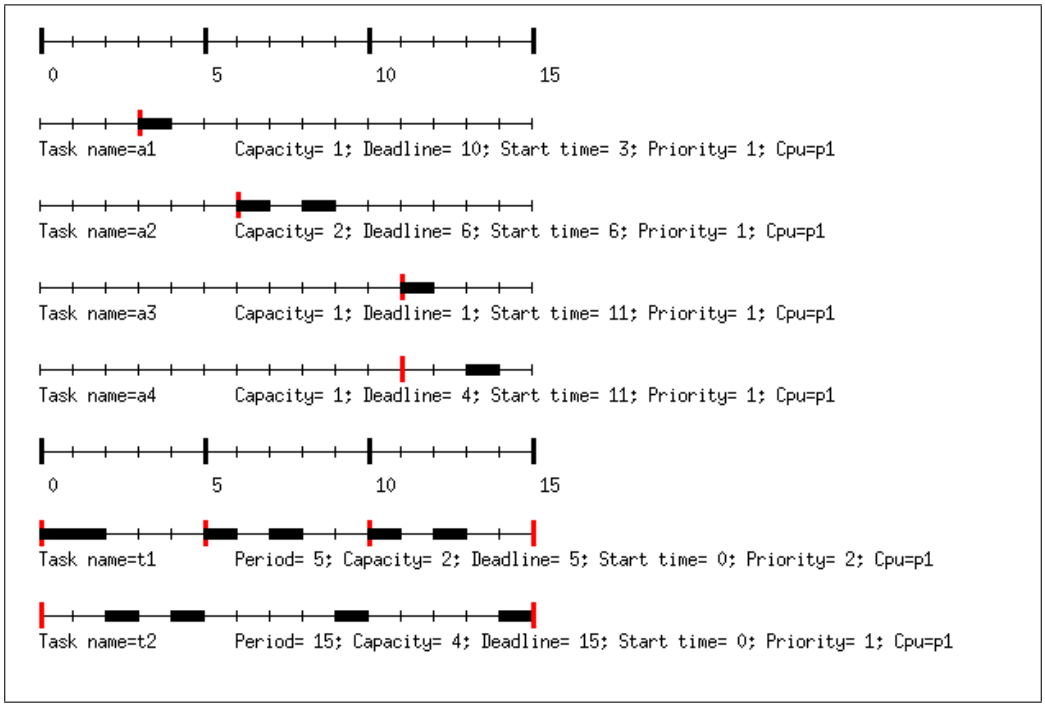

Figure 13: First sporadic server test

The figure 13 is the test result for the polling server with tasks set from the figure 5.

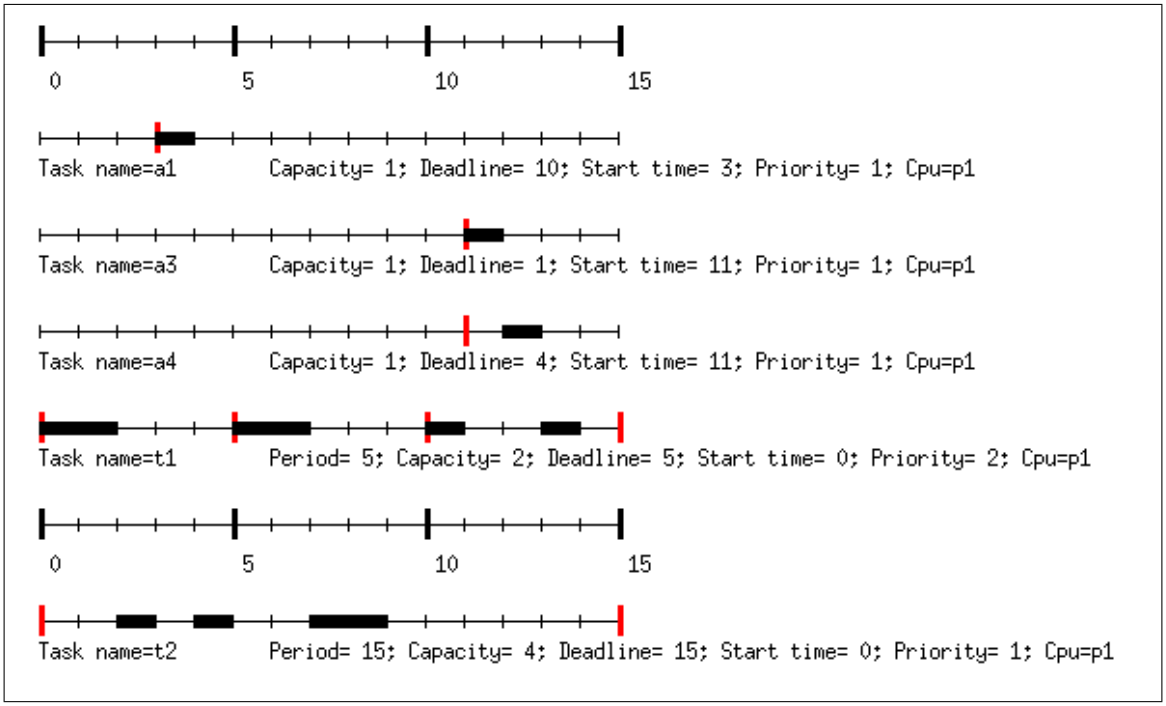

Figure 14: Second sporadic server test

The figure 14 is the test result for the polling server with tasks set from the figure 6.

18

We can see that all of the above tests were conclusive and behaved as expected (see figures in section 1 Aperiodic servers). However, as the implementation for the priority exchange server is still incomplete, there is no test available.

Conclusion

This paper's objective was to do a bibliographic research of the different aperiodic servers and then to implement different them in the Cheddar software. We have found four different aperiodic servers, the polling server, the deferrable server, the sporadic server and the priority exchange server. All but the priority exchange server were actually implemented and tested in Cheddar.

Even though the priority exchange server is not implemented in Cheddar, the server's behavior has been defined as well as the test set (XML file).

References

[1] F. Cottet, J. Delacroix, C. Kaiser, Z. Mammeri.
Ordonnancement temps reel.
Hermes Science publication.

[2] B. Sprunt, L. Sha, J. Lehoczky
Aperiodic Taks Scheduling for Hard-Real-Time Systems. Journal of Real time system.June 1989, Volume 1, Issue 1, pp 27-60.

[3] A. Linard.
Partie 5: Algorithmes pour les taches aperiodiques
Support de cours sur l'ordonnancement Temps Reel

[4] F. Singhoff, J. Legrand, L. Nana, L. Marcé.
Cheddar : a Flexible Real Time Scheduling Framework.
ACM SIGAda Ada Letters, volume 24, number 4, pages 1-8. Edited by ACM Press, New York, USA. December 2004, ISSN:1094-3641.

[5] C. Fotsing, F. Singhoff, A. Plantec, V. Gaudel, S. Rubini, S. Li, H. Nam Tran, L. Lemarchand, P. Dissaux, J Legrand.
Cheddar Architecture Description Language.
January 23, 2014. Lab-STICC technical report.