http://beru.univ-brest.fr/~singhoff/cheddar/contribs/educational/ubo/ETR15/bleuvert.gif

**Tutorial about Cheddar : an example of real-time scheduling analysis with Cheddar**

**Frank Singhoff**

http://beru.univ-brest.fr/~singhoff/cheddar/contribs/educational/ubo/ETR15/bleuvert.gif

* [**I. First handouts with Cheddar**](http://beru.univ-brest.fr/~singhoff/cheddar/contribs/educational/ubo/ETR15/tp-gb.html#Ref1)
* [**II. Comparing scheduling algorithms**](http://beru.univ-brest.fr/~singhoff/cheddar/contribs/educational/ubo/ETR15/tp-gb.html#Ref2)
* [**III. Sharing resources**](http://beru.univ-brest.fr/~singhoff/cheddar/contribs/educational/ubo/ETR15/tp-gb.html#Ref3)
* [**IV. Exercises on both modeling and verification**](http://beru.univ-brest.fr/~singhoff/cheddar/contribs/educational/ubo/ETR15/tp-gb.html#Ref4)
* [**V. References**](http://beru.univ-brest.fr/~singhoff/cheddar/contribs/educational/ubo/ETR15/tp-gb.html#Ref5)
* [**VI. Solutions**](http://beru.univ-brest.fr/~singhoff/cheddar/contribs/educational/ubo/ETR15/corrections)

http://beru.univ-brest.fr/~singhoff/cheddar/contribs/educational/ubo/ETR15/bleuvert.gif  
  
  
The objective of this lab is to experiment several classical algorithms and analysis methods of the real-time scheduling theory. For such a purpose, we will use Cheddar. The first exercise of this tutorial should allow you to understand how to basically use Cheddar. In this exercise, we see how to model a software architecture and how to analyse it. In the sequel :

1. Exercises 2 and 3 present and compare several classical real-time scheduling algorithms.
2. Exercises 4 and 5 address shared resources.
3. Finally, exercises 6 and 7 propose full software architectures to be modeled and verified.

Ideas of those exercises were taken from [COT 00, BUR 97, DEM 99, KRI 97]. We can look for those books for other extra exercises.

**I. First handouts with Cheddar**

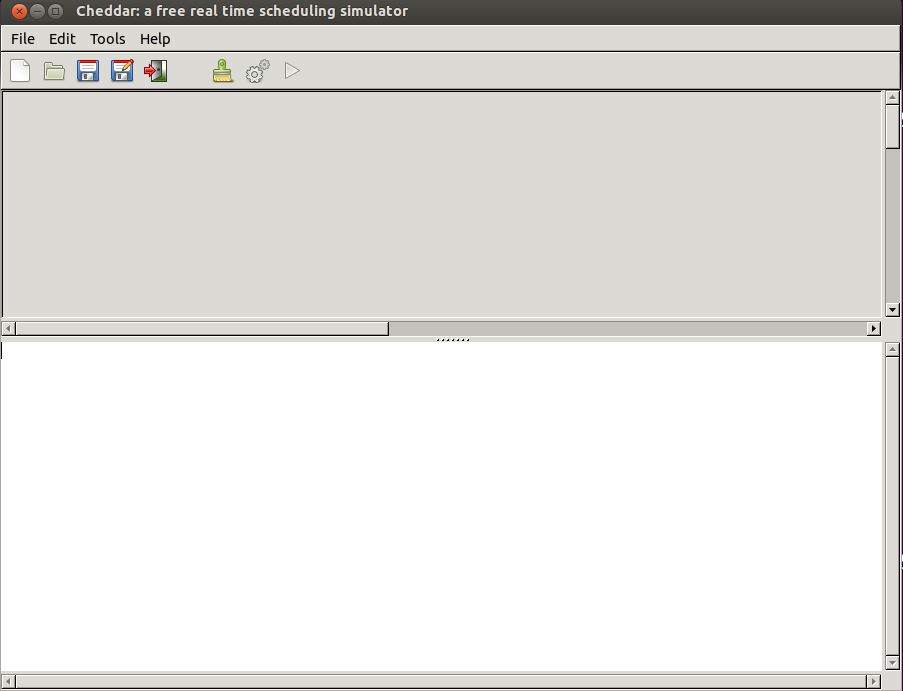
**I.1 How to set your workspace**

To perform those exercises, the simplest way to use Cheddar is to download from <http://beru.univ-brest.fr/~singhoff/cheddar/vbox>, the vdi VirtualBox file cheddar\_tutorial.vdi. This vdi file contains all binaries. This lab assume basic background on real-time scheduling analysis.   
  
Notice also that source code and binaries (for both linux and windows) of this tool can be downloaded from the Cheddar SVN repository there : <http://beru.univ-brest.fr/svn/CHEDDAR/trunk/releases>. Furthermore, AADLInspector, a commercial version of Cheddar [can be found here.](http://www.ellidiss.fr/public/wiki/wiki/inspector) AADLInspector is maintened by Ellidiss Technologies and assumes you are using AADL, which is not required to perform this tutorial.   
  
Once the virtual machine is started on your PC/Mac, you must connect yourself with the user cheddar (its password is also cheddar. Then, from a terminal, run the following sequence of commands to launch Cheddar :

>source cheddar.bash

>cheddar&

The window of the figure 1 should be displayed :



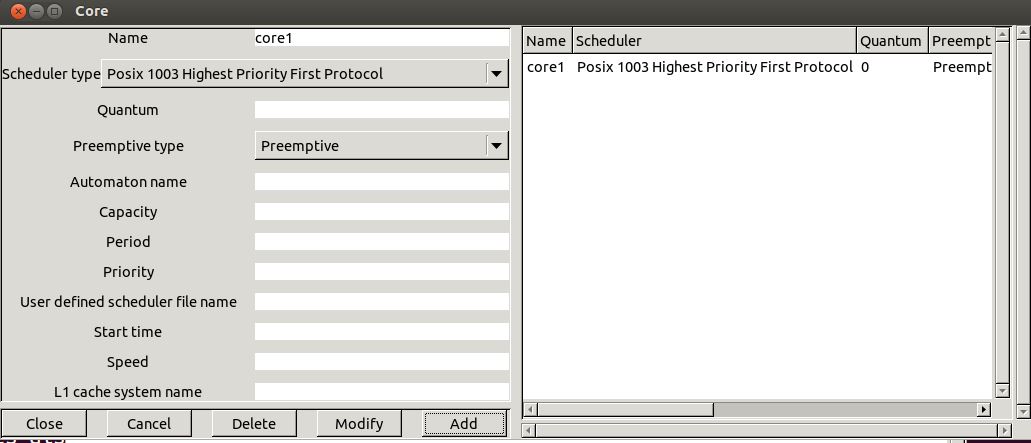
**Fig. 1 : main window of Cheddar**

The top part of this window (in gray in the figure) displays the time lines computed during scheduling simulation. The bottom part of this window displays numeric analysis results.

**I.2 Editing the model to analyse**

In this section, we will design our first Cheddar model. This model is one of the task set presented by Giuseppe Lipari during ETR 2015. You will find several other XML Cheddar files [here](http://beru.univ-brest.fr/~singhoff/cheddar/contribs/educational/ubo/ETR15/lipari_examples) . These Cheddar XML files are modeling most of the task sets used by G. Lipari for his ETR 2015 tutorial.   
  
Let describe the architecture of the real-time system we want to verify. For such a purpose, we will use the Cheddar built-in ADL (ADL stands for Architecture Description Language), called Cheddar ADL. Cheddar ADL is an architecture description language dedicated to real-time scheduling analysis. A Cheddar ADL model is composed of a description of both the hardware and of the software part of the system to analyse. Those models can be written with any editors, but in the sequel, we will use the graphical editor of Cheddar as follow :

1. First, the hardware components composing the execution environment running the software must be descrided. With Cheddar, the execution platform can be composed of one or several processor components embedding each one or several core components with optionnal cache components. A core component models a unit of computation, i.e. an entity providing the support to run one or several tasks.   
     
   To describe a core component, use the menu Edit/Hardware/Core which opens the windows of the figure 2 :



**Fig. 2 : editing a core**

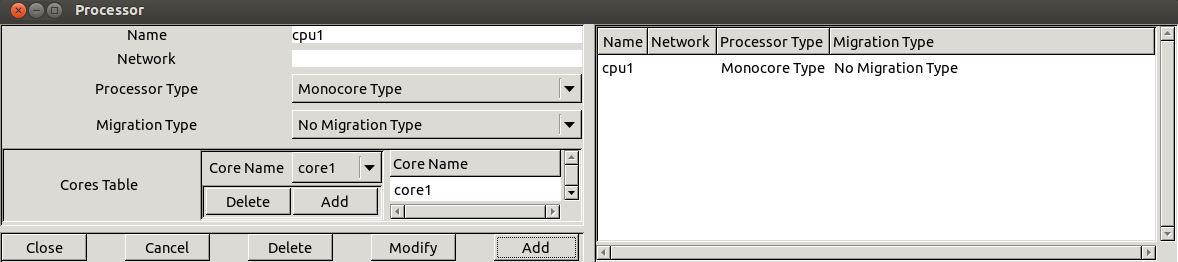
A core component is defined by the following attributes :

* + Name : the unique name of the component, here, core1.
  + Scheduler type : the way tasks will be allowed to share the core, i.e. the scheduling policy assigned to the core. In the window bellow, the core component owns a preemptive fixed priority scheduler compliant with POSIX 1003 (value POSIX\_1003\_Highest\_Priority\_First\_Protocol for the Scheduler type attribute). In Cheddar, with such scheduler, allowed priority values ranges from 0 to 255 (255 is the highiest priority level). As with Linux, priority 1 to 255 are devoted to SCHED\_RR and SCHED\_FIFO tasks while priority level 0 is devoted to the SCHED\_OTHER policy. Such policies allow Cheddar to compute a deterministic scheduling when the software architecture model have several tasks with equal priorities.
  + Preemptive type : specicy if the scheduler is preemptive or not.
  + The other attributes can be ignored for such tutorial.

All the window you may use to edit Cheddar ADL components have the same interface :

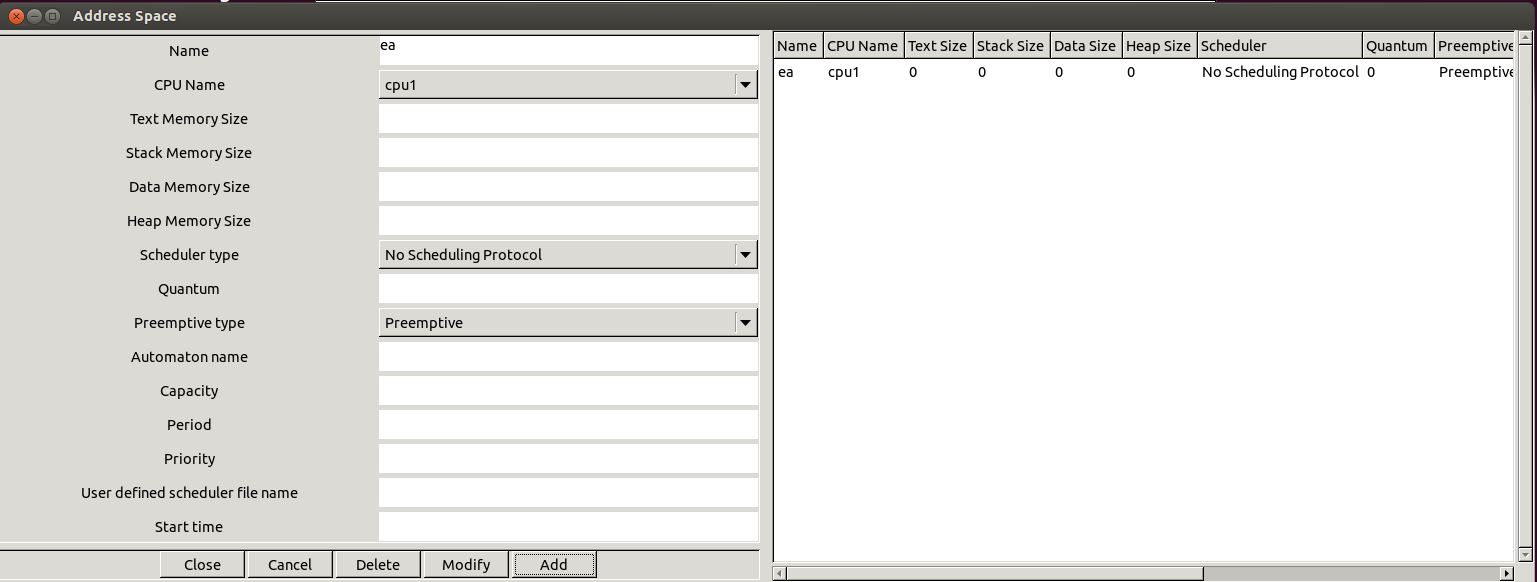
* + Button Add : allows you to add a component to the architecture model.
  + Button Delete : allows you to delete a component from the architecture model. Before pushing this button, you must select the targetted component in the component list in the right side of the window.
  + Button Modify : this button has a behavior similar to the Delete button, excepts that the component is updated with the values of the attributes in the left part of the window.
  + Button Close : closes the window and validates all the updates made since the windows was opened.
  + Button Cancel : closes the window and cancels all the updates made since the windows was opened.

Once core1 is defined, we can define processor components that use core1 with the Edit/Hardware/Processor menu, which displays the following windows:



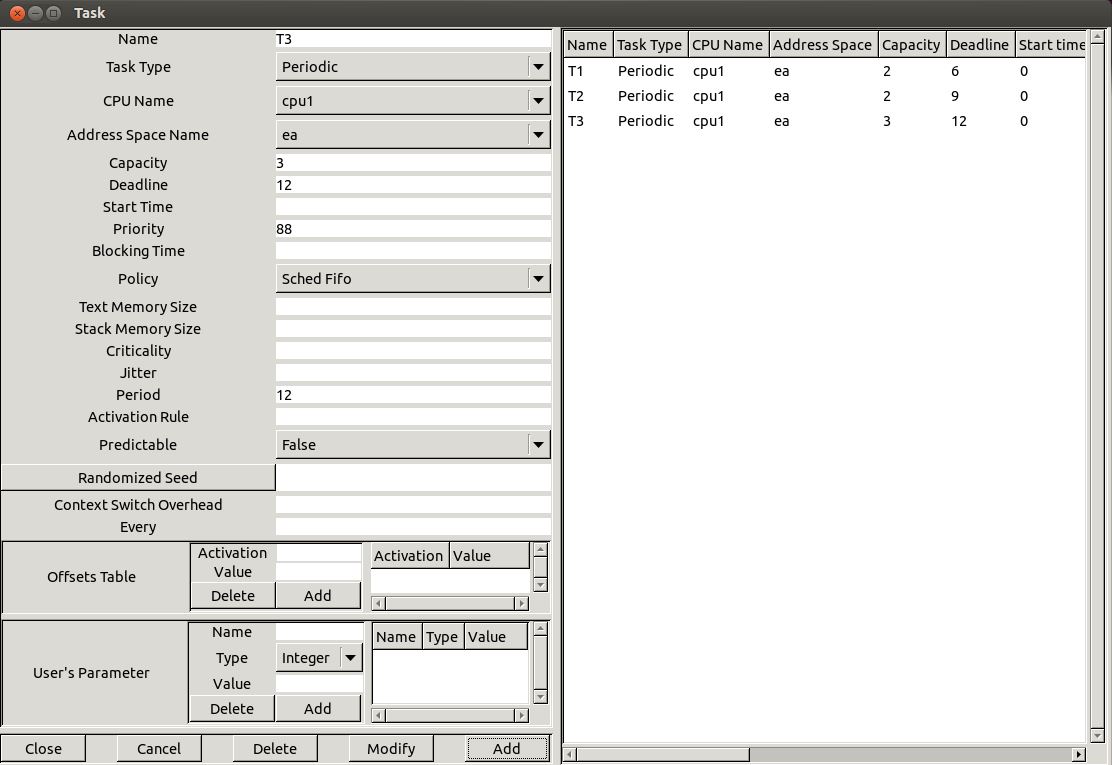
**Fig. 3 : editing a processor component**

In the figure 3, we define a processor named cpu1 which is composed of the core1 component. Adding or deleting a core inside a processor can be achieved with the buttons Add and Delete from the subwindows called Cores Table. In this lab, we restrict ourself with uniprocessor architecture models. Then, processor only contains one core component and then, we need to set Processor Type and Migration Type attributes respectively to Monocore Type and No Migration Type values.   
  
In the sequel, we express the software part of the system to analyse. For this lab, this software part is mainly composed of the tasks and the shared resource components. (task Cheddar ADL component type and resource component type). Other software components exist in Cheddar ADL, but they are not required there (See the Cheddar ADL reference guide for further details). task and resource components have to be defined inside Address Space components An Address Space component can be seen as a entity modelling the concept of an application, or a Unix process address space, or even an ARINC 653 partition.   
  
The window of the figure 4 (menu Edit/Software/Address space) shows you how to define a Address Space for the labs. The only attributes we have to define are Name and Cpu Name. For the next exercises, we have to define one Address Space component for each processor component. However, for specific analysis as ARINC 653 or more generally, hierarchical architecture models, more attributes have to be specified for such kind of component.



**Fig. 4 : editing an address space component**

The last step to edit a Cheddar ADL model is to specify task parameters, which can be achieved with the menu Edit/Software/Task :



**Fig. 5 : editing a task component**

Again, such component type have mumerous attributes. The most importants are :

* + Name : the unique name of the component
  + Task type : the type of the task, which specifies how the task will be released. A task may be periodic, sporadic, released according to a poisson process (e.g. poisson task type), GMF, aperiodic, ...
  + Capacity : the WCET of the task
  + Period : the delay between two successive release times of the task.
  + Deadline : the deadline the task has to meet. It is a delay relative to the period of the task (i.e. not an absolute deadline).
  + Address space name and CPU name where the task is supposed to be located/run.
  + Priority : : priority level assigned to the task. 255 is the higher level of priority ; 0 is the lower.

In the example of the figure 5, we have defined 3 tasks, which were periodic, synchronous, with deadlines on requests, and then, which were defined with the following attributes :

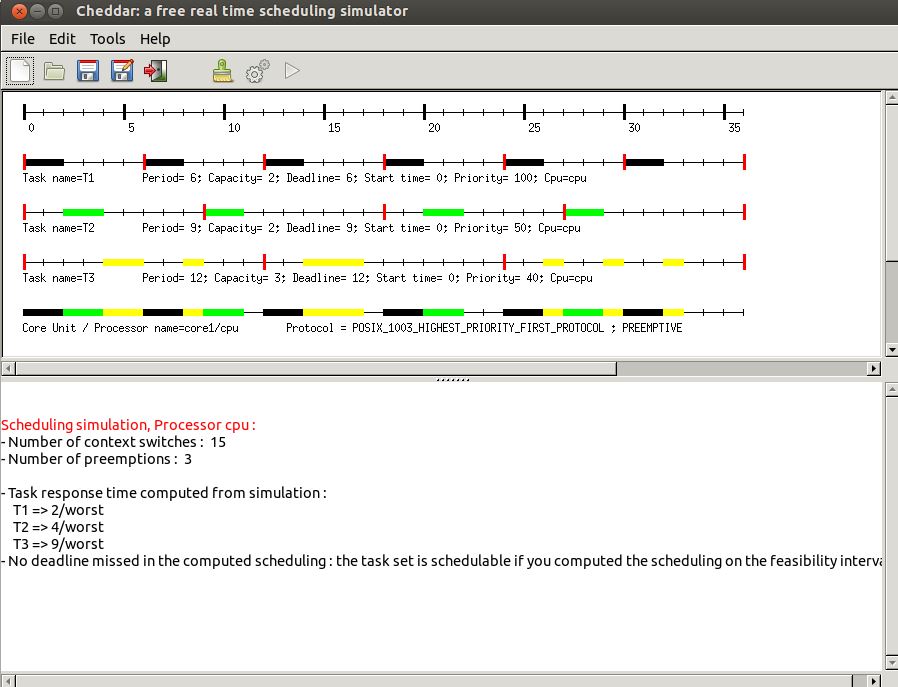
* + Task T1 : Capacity = 2, Deadline = 6, Period = 6
  + Task T2 : Capacity = 2, Deadline = 9, Period = 9
  + Task T3 : Capacity = 3, Deadline = 12, Period = 12
  + Priority attributes were assigned according to Rate Monotonic within the allowed priority range (from 1 to 255).
  + You should notice that the deadlines are equal to the periods (see attribute Deadline).
  + Finally, the attribute Start time is set to 0 in order to model a synchronous task set.

**I.3 Analysis of a model**

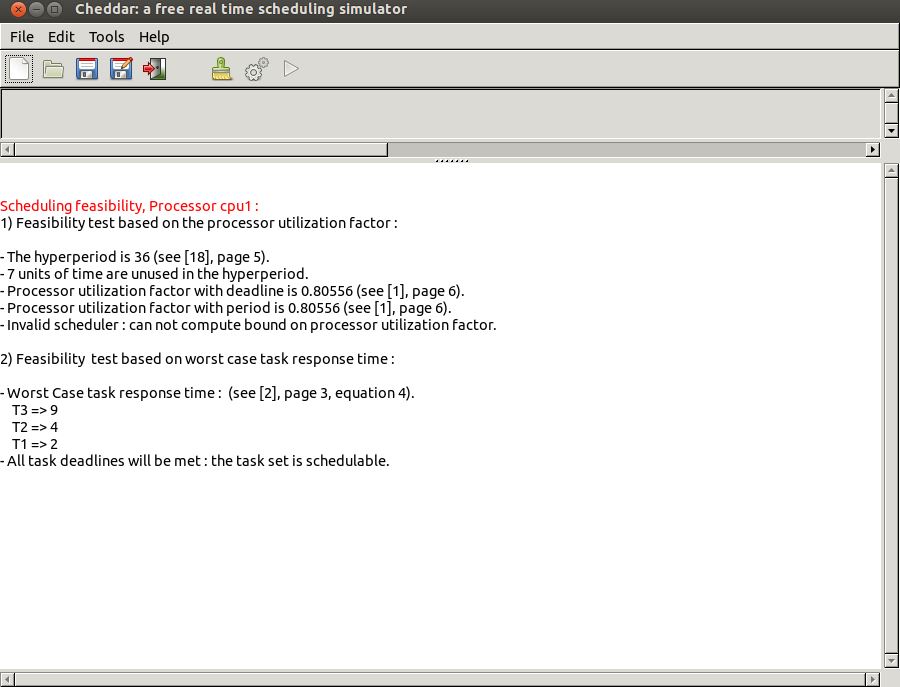
In this last section, we show how to perform an analysis of the architecture model previously designed.   
  
From the menu Tools and for a given architecture model, we can perform 3 types of analysis with Cheddar:

* + We can use simulation tools to compute the scheduling of a task set, and then, from such scheduling, to compute various performance criteria (worst/average/best response times, missed deadlines, blocking time on shared resources, deadlocks, ...).
  + We can also call various feasibility tests that allow the designers to check deadlines without computing the scheduling of the task set.
  + If the previous tools have shown that some deadlines cannot be met, we can also improve the architecture model by various means : priority assignments, task partitionning, task clustering, ...

Under the menu, some buttons allow you to quickly access to a summary of those Cheddar analysis features (all analysis features can be accessed by the Tools menu). Indeed, the http://beru.univ-brest.fr/~singhoff/cheddar/contribs/educational/ubo/ETR15/simu.jpgbutton starts a scheduling simulation and the http://beru.univ-brest.fr/~singhoff/cheddar/contribs/educational/ubo/ETR15/feasibility.jpgbutton computes a sample of feasibility tests for a basic periodic task set. Basically, the http://beru.univ-brest.fr/~singhoff/cheddar/contribs/educational/ubo/ETR15/feasibility.jpgbutton computes worst case task response time following the Joseph et Pandia [JOS 86] method extended for any Deadline and classical CPU bound tests. Again, for specific feasibility tests (e.g. transactions), see the Tools menu.   
  
Figures 6 et 7 show a screenshot of Cheddar for the model edited in section I.2, and for results produced by buttons http://beru.univ-brest.fr/~singhoff/cheddar/contribs/educational/ubo/ETR15/simu.jpgand http://beru.univ-brest.fr/~singhoff/cheddar/contribs/educational/ubo/ETR15/feasibility.jpg. We can see there worst case response times computed by the Joseph and Pandia [JOS 86] method and also by the scheduling simulation on the hyper-period. Feasibility tests on cpu utilization factor cannot be applied there as the tool cannot be sure that priorities have been assigned according to Rate Monotonic. To apply this test, one must change the scheduling policy of the core1 component with the Rate\_Monotonic\_Protocol value.   
  
We can also notice in the time lines that red rectangles show task release times, black rectangles show when tasks are running. Many other display options exist there and can be modified by the Tools/Scheduling/Scheduling Options menu (for example : to display a different color for each task).



**Fig. 6 : a scheduling simulation**



**Fig. 7 : compute schedulability feasibility tests**

Let see now our first true exercise.   
  
  
  
  
**Exercise 1 :**

* + Edit a Cheddar model for the system analyzed in the figures 6 and 7, and then, with buttons http://beru.univ-brest.fr/~singhoff/cheddar/contribs/educational/ubo/ETR15/simu.jpgand http://beru.univ-brest.fr/~singhoff/cheddar/contribs/educational/ubo/ETR15/feasibility.jpg, check that you get the results of the figures 6 and 7.
  + From the scheduling simulation, read the different response times for each task (remind that response time = completion time - release time). For a given task, can you see the response time varying during the scheduling simulation ? Why ? When should we see the shortest response time ? Why ?
  + Update the parameters of the tasks as follows:
    - Task T1 : Capacity = 2, Deadline = Period = 6, Start time = 0
    - Task T2 : Capacity = 8, Deadline = Period = 24, Start time = 0
    - Task T3 : Capacity = 4, Deadline = Period = 12, Start time = 0
    - Priority : apply Rate Monotonic.

This task set is said to be harmonic as each period is multiple with the other periods of the task set. For such a kind of task set, we know that we can meet task deadlines even if the processor is busy upto 100%, provided a preemptive fixed priority scheduling policy and a Rate Monotonic priority assignment are applied.

* + With Cheddar, compute the scheduling simulation during the hyper-period (button http://beru.univ-brest.fr/~singhoff/cheddar/contribs/educational/ubo/ETR15/simu.jpg). From this simulation, check the schedulability of this task set.

**II. Comparing algorithms**

In the sequel, we will experiment and compare 3 classical scheduling policies.   
  
**Exercise 2 :**   
  
We consider two periodic tasks, synchronous and with deadline on request : tasks T1 and T2. They are defined as follow:

* + Task T1 : Capacity = 4, Deadline = Period = 8, Start time = 0
  + Task T2 : Capacity = 5, Deadline = Period = 10, Start time = 0
  + Priority are assigned according to Rate Monotonic.
  + Edit a Cheddar model for this task set, running on a processor composed of one core. This core hosts a preemptive fixed priority scheduler.
  + With Cheddar, compute a scheduling simulation for the hyper-period (button http://beru.univ-brest.fr/~singhoff/cheddar/contribs/educational/ubo/ETR15/simu.jpg).
  + From this simulation, what are the worst case response times of each task ? Can we see missed deadlines ?
  + Change you Cheddar model in order to use a preemptive EDF instead (value Earliest\_Deadline\_First\_Protocol for the Scheduler type attribute). Do again questions 2 and 3.
  + Compare both analysis results. What can you see ? Is it surprising ?
  + Change again your model : now we consider a preemptive EDF with the same task sets, but you must add now an aperiodic task. We do not require that the deadline of this aperiodic task must be met. To define such aperiodic task, change the value for the attribute Task Type (see. figure 5):
    - Periodic task T1 : Period=Deadline=8, Capacity=4, Start time = 0
    - Periodic task T2 : Period=Deadline=16, Capacity=8, Start time = 0
    - **Aperiodic** task T3 : Deadline=15, Capacity=4, Start time = 0
  + Compute again the scheduling simulation for the first 30th units of time (button http://beru.univ-brest.fr/~singhoff/cheddar/contribs/educational/ubo/ETR15/simu.jpg). What can you see now ?
  + What can we conclude about the suitability of EDF for critical real-time systems ? Is it the case with fixed priority scheduling ?

**Exercise 3 :**   
  
We consider two periodic tasks, synchronous, with deadline on request : tasks T1 and T2. They are defined as follow:

* + Task T1 : Period=Deadline=9, Capacity=4, Start time = 0
  + Task T2 : Period=Deadline=8, Capacity=3, Start time = 0

We investigate now another scheduling policy : LLF (Least Laxity First). This policy selects the task to run amoung the ready tasks according to a dynamic priority called 'laxity'. Li(t), the laxity of a task i at time t can be computed by Li(t)= Deadline - remaining(t) where remaining(t) is the remaining capacity of the task at time t.

* + Edit a Cheddar model for this task set running on a processor composed of one core. This core hosts a preemptive EDF scheduler.
  + With Cheddar, compute a scheduling simulation for the hyper-period (button http://beru.univ-brest.fr/~singhoff/cheddar/contribs/educational/ubo/ETR15/simu.jpg).
  + From this simulation, what are the worst case response times of each task ? Can we see missed deadline ?
  + Change your Cheddar model in order to use a LLF policy instead (value Least\_Laxity\_First\_Protocol for the Scheduler type attribute). Do again questions 2 and 3.
  + What can we see about the results produced by both those scheduling policies ? What are the common points and differences ?
  + Conclude about the choice between those two scheduling policies.

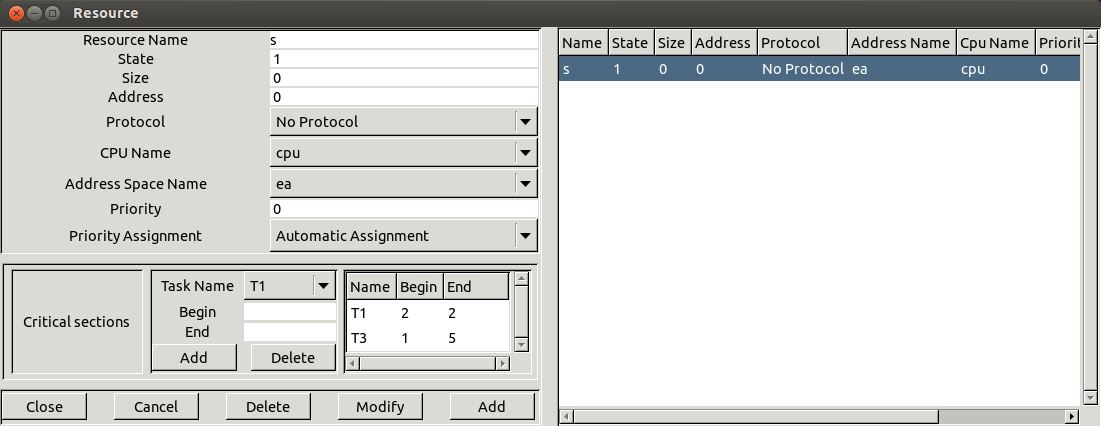
**III. Shared resources**

In the previous exercises, we assumed the tasks as independent. In the sequel, we illustrate the use of Cheddar with dependent tasks : with tasks sharing resources according to classical protocols such as PCP or PIP.   
  
  
**Exercise 4 :**   
  
We consider three periodic tasks, synchronous and with deadlines on request : tasks T1, T2 and T3. They are defined as follow:

* + Task T1 : Period=Deadline=6, Capacity=2, Start time = 0
  + Task T2 : Period=Deadline=8, Capacity=2, Start time = 0
  + Task T3 : Period=Deadline=12, Capacity=5, Start time = 0

We assume a preemptive fixed priority scheduling policy and we apply Rate Monotonic to assign priorities.   
  
We also assume that tasks T1 and T3 share a resource named S. T1 and T3 access to S in mutual exclusion:

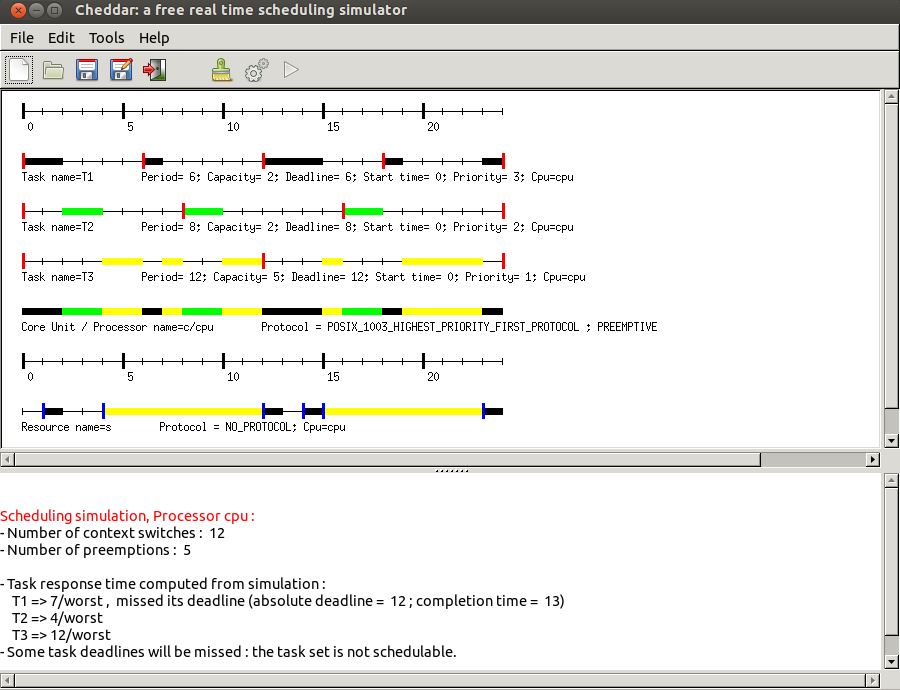
* + T3 needs S during all its capacity.
  + T1 needs S during the 2nd unit of time of its capacity only.
  + We edit a Cheddar model for such task set and this resource S. To define S, we need the Edit/Software/Resource menu which displays the following window:



**Fig. 8 : editing a resource component**

In this window, and for our lab, we need to set the following attributes :

* + - Name : the unique name of the shared resource.
    - CPU Name and Address space name : give where the resource is located.
    - State : is the status of the resource. A Cheddar ADL resource may be seen as a counting semaphore. A Cheddar ADL resource has an integer value. When a task allocates a resource, its counter is decremented. When a task releases a resource, its counter is incremented. When this counter is equal of less than 0, a task which requests the resource is blocked as the resource is already allocated. For our exercises, State must be set to 1.
    - Protocol : is the protocol to apply when a resource is handled by a task. The value No Protocol means that the simulator will apply no priority inheritance and that the blocked tasks on the semaphore are stored in a FIFO queue. In this exercise and in the next one, we will alo use the values Priority Inheritance Protocol (or PIP) and Priority Ceiling Protocol (or PCP) when we need to apply inheritance protocols. Right now, you must set this attribute with the value No Protocol for the resource S.
    - Priority assignment : specify if the ceiling priority of the resource component will be computed automatically by Cheddar, or assigned manually by you. For our exercises, we have to set this attribute to Automatic Assignment.
    - Critical sections : in this part, we specify when each task needs the shared resources. Begin is the start time of each critical section and End is the critical section completion time. The buttons Add and Delete allow the designer to create or remove a critical section for a given resource. As an example, the figure 8 shows how to express that T1 needs the resource at the beginning of the 2nd unit of time of the T2 capacity and releases the resource at the end of this unit of time. Again in this figure, we exlain that T3 needs the resource during all its capacity (i.e. before starting the 1st unit of time and after the 5th unit of time of the T3 capacity).
  + Compute the scheduling simulation on the hyper-period (button http://beru.univ-brest.fr/~singhoff/cheddar/contribs/educational/ubo/ETR15/simu.jpg). You should get the scheduling simulation of the figure 9.



**Fig. 9 : scheduling simulation with a shared resource**

* + On the resource time line, black rectangles show when the resource is allocated. Those rectangles can be colored as the tasks are (see the Several colors option in the Tools/Scheduling/Scheduling Options menu). Furthermore, bleu rectangles show when the resources are locked while red rectangles show when resources are unlocked.
  + In this time line, find when tasks T1 and T2 lock and unlock S.
  + From this time line, can we see priority inversions ? Where ?
  + Change S protocol by Priority Inheritance Protocol (or PIP), and compute again the scheduling simulation. With PIP, a task which blocks another with a higher priority, runs its critical section with the blocked task priority.
  + What can we see in this new scheduling simulation about the shared resource ?
  + Say at which time the priorities of the tasks change due to PIP.
  + With the http://beru.univ-brest.fr/~singhoff/cheddar/contribs/educational/ubo/ETR15/feasibility.jpgbutton, compute the worst case response time with the Joseph et Pandia method and compare those results with the one computed during the scheduling simulation. What can you see ? Explain.
  + From the scheduling simulation, give the blocking time of each task due to the shared resource S.

**Exercise 5 :**   
  
We now see a Cheddar model with two shared resources. We assume two periodic tasks with the following parameters :

* + Task T1 : Period=Deadline=31, Capacity=8, Start time = 0
  + Task T2 : Period=Deadline=30, Capacity=8, Start time = 2
  + You should notice that those tasks are not synchronous ; indeed, they have a different value for their Start time attribute.
  + Prioritie are assigned according to Rate Monotonic.

We use a fixed priority preemptive scheduling policy for such task set.   
  
T1 and T2 require the access to the shared resources R1 and R2 as follows:

* + T1 needs R1 from the 2nd unit of time of its capacity upto the 8th (included).
  + T2 needs R1 from the 6th unit of time of its capacity upto the 8th (included).
  + T1 needs R2 from the 4th unit of time of its capacity upto the 8th (included).
  + T2 needs R2 from the 2nd unit of time of its capacity upto the 8th (included).

We also assume that both R1 and R2 apply PIP (Priority Inheritance Protocol).

* + Edit a Cheddar model for such software architecture and compute the scheduling simulation during the 30th first units of time (button http://beru.univ-brest.fr/~singhoff/cheddar/contribs/educational/ubo/ETR15/simu.jpg).
  + Look for in the time lines where T1 and T2 lock and unlock R1 and R2.
  + What can you see ? Is it surprising ? Explain.
  + We now apply the PCP protocol (Priority Ceiling Protocol in the figure 8). For such a purpose, you have to update both R1 and R2. PCP works as follows:
    - A ceiling priority is assigned to each resource. This priority is equal to the highest priority level of the task accessing the resource. This ceiling priority can be computed automatically by Cheddar if specified in the resource component definition.
    - A task which blocks another with a higher priority level, must run the critical section with the priority of the blocked task (inheritance priority as PIP).
    - When a task tries to lock a resource, the task is blocked if its priority is not higher than all the ceiling priorities already allocated by other tasks. This is a new blocking condition which is specific to PCP.
  + Compute again the scheduling simulation during the 30th units of time with the PCP resources.
  + Show in the time lines when T1 and T2 lock and unlock R1 and R2.
  + Say when the priorities of the tasks change due to PCP.
  + Compare those results with the one of the question 1.

**IV. Exercises on both modeling and verification**

**Exercise 6 :**   
  
We investigate a system embedded in a car which displays various data to the driver. This system is composed of several tasks and we expect to verify their timing behavior.   
  
The system is composed of 5 tasks :

* + Task SENSOR\_1 reads every 10 ms the speed of the car.
  + Task SENSOR\_2 reads every 10 ms the temperature inside the car.
  + Task SENSOR\_3 reads every 40 ms the GPS position of the car.
  + Task DISPLAY\_1 produces on a screen every 12 ms a summary of the information retrieved by the three sensor tasks.
  + Finally, tasks DISPLAY\_2 displays on a second screen the map of the current location of the car. This diplay is made on request by the driver every 6 ms.

An enginer implements all those tasks and then, run them alone on the targetted system. After several measurements, he shows that :

* + Task SENSOR\_2 and DISPLAY\_1 execution times never exceed 2 ms.
  + Task SENSOR\_3 needs between 2 and 4 ms to successfully run its code.
  + Finally, tasks SENSOR\_1 and DISPLAY\_2 have an execution time of less than 1 ms.

We assume that all tasks start their first activation at the same time. Furthermore, all tasks must complete their current activation before they have to start the next one.   
  
We assume an execution platform composed of one processor which provides a preemptive fixed priority scheduler. Priorities range from 0 to 31. 0 is the lowest priority value. The system does not allow designers to assign the same priority value to several tasks.   
  
**Question 1:**

* + Usually, what are the criteria to assign priority to tasks ?
  + Assign a priority level for each task. Motivate your choices.

**Question 2:**   
With Cheddar, from the priority assignment of the previous question, compute the worst case response time with the Joseph and Pandia method for the SENSOR\_1, the SENSOR\_2, and the SENSOR\_3 task.   
  
**Question 3:**   
In fact, tasks SENSOR\_1, SENSOR\_2, and SENSOR\_3 share a memory area protected by a semaphore (i.e. accessed in mutual exclusion). SENSOR\_1 et SENSOR\_2 needs this memory during all their capacity while SENSOR\_3 only needs it during the 2 first units of time of its capacity. The semaphore protecting the memory applies the PIP protocol. Explain why the worst case response times computed in the previous question are not true anymore.   
  
**Question 4:**   
With Cheddar, compute the scheduling of this task set on the 60th first units of time. Say when the shared resource is allocated and released. Say also if some priority inversions occur. Finally, say when the priorities of the tasks change due to PIP.   
  
**Question 5:**   
**For the questions 5 and 6, we now assume that all tasks are independent.**   
  
In order to activate more frequently the tasks, we experiment the scheduling of those tasks on a two processors platforms : processors a and b. Migrations of the tasks are not allowed : we then apply a partitionning approach.   
  
The tasks are defined according to those parameters :

* + Task DISPLAY\_1 : Processor = b ; Capacity = 4, Period=Deadline= 6
  + Task SENSOR\_1 : Processor = b ; Capacity = 2, Period=Deadline= 20
  + Task DISPLAY\_2 : Processor = a ; Capacity = 2, Period=Deadline= 3
  + Task SENSOR\_2 : Processor = a ; Capacity = 2, Period=Deadline= 5
  + Task SENSOR\_3 : Processor = a ; Capacity = 2, Period=Deadline= 5

The tasks are scheduled according to a preemptive fixed priority scheduling policy. Task priorities are assigned according to Rate Monotonic.   
  
Show without scheduling simulation that it exists at least one task which cannot meet its deadline.   
  
**Question 6:**   
In fact, tasks were assigned to processors randomly. All tasks can run on both processors. Processors a and b only differ from their execution speed : a task which runs on processor b runs twice more quickly than on processor a.   
  
Assign tasks to processors a and b such as all task deadlines can be met. **Explain your choice.**   
  
  
  
  
  
  
  
**Exercise 7 :**   
  
This exercise is extracted from [COT 00] and is about a simplified architecture model of the Mars Pathfinder mission. In this exercise, you must look for a design mistake and propose a solution for it. In 1997, Mars Pathfinder casts a mobile robot called Sojourner on Mars. This mobile robot is controled by a multitask software running on a VxWorks target. This software is composed of the following tasks :

|  |  |  |  |
| --- | --- | --- | --- |
| **Tasks** | **Priorities** | **Periods/Deadlines** | **WCET** |
| SCHED\_BUS | 1 | 125 ms | 25 ms |
| DATA | 2 | 125 ms | 25 ms |
| CONTROL | 3 | 250 ms | 25 ms |
| RADIO | 4 | 250 ms | 25 ms |
| VIDEO | 5 | 250 ms | 25 ms |
| MESURE | 6 | 5000 ms | 50 ms |
| FORECAST | 7 | 5000 ms | Between 50 ms and 75 ms |

* + All the tasks are periodic, synchronous, and have deadlines equal to periods.
  + FORECAST is sometimes released for a job of 50 ms, and sometimes for a job of 75 ms, depending of the size of the payload to handle.
  + Priority levels expressed bellow are VxWorks priorities : the lower the value is, the higher the priority level is.
  + DATA, CONTROL, MESURE and FORECAST required a shared data which is accessed by a critical section during all their execution times (i.e all their capacity). To reduce costs, those critical sections are implemented with a mutex that does not use inheritance priority protocol such as PIP or PCP.
  + During the mission of Mars PathFinder, operators noticed that some deadlines were missed, leading to a reboot of the hardware and some losses of data. What are the missed deadlines ? Why ?
  + How to solve this issue ? Apply it on your Cheddar ADL model.

**V. References**

[BUR 97] A. Burns and A. Wellings. Real-time Systems and Programming Languages. Addison Wesley, 1997.   
  
[COT 00] F. Cottet, J. Delacroix, C. Kaiser, and Z. Mammeri. Ordonnancement temps réel. Hermès, 2000.   
  
[DEM 99] I. Demeure and C. Bonnet. Introduction aux systèmes temps réel. Collection pédagogique de télécommunications, Hermès, septembre 1999.   
  
[JOS 86] M. Joseph and P. Pandya. Finding Response Time in a Real-Time System. Computer Journal, 29(5):390--395, 1986.   
  
[KRI 97] C.M. Krishna and K.G. Shin. Real-Time Systems. Mc Graw-Hill International Editions, 1997.

*Updated by Frank Singhoff* [*(singhoff@univ-brest.fr)*](mailto:singhoff@univ-brest.fr) *Last update : April 2016*