GWGUI beta 0.2 User's Manual - GWGUI a Graphic User Interface for the GridWay Metascheduler

Author: IVAN FERNANDEZ HERNANDEZ

February, 2008

Contents

List of Figures

Chapter 1

GWGUI beta 0.2 User's Manual - GWGUI a Graphic User Interface for the Gridway Metascheduler

1.1 Description

This application have been developed for education purposes inside the context of a Project on a Computer Sciences Bachelor. The main goal of this application is to provide a graphical interface to the users of Gridway Metascheduler, so they would be able to interact with it in a simple level, it means to interact with Gridway daemon through its command line language and not through the DRMAA API provided by Gridway. GWGUI uses a simple design to make easier for users to design scripts that will be used to interact with Gridway and to provide them a visual tool to work in the field of task dependency diagrams. In the followings sections we will talk abouts the deferents parts of the application.

1.2 What features has this software?

This software is able to make easier to users to interact with the Gridway Metascheduler during the workflow design phase or later, for sending job templates and needed files for the generated workflows to a Gridway Server host. This software has been developed in Java for many reasons, the most important reason was the portability, although it' s recommendable to use it in Unix based Operating Systems due to the file naming systems (not Windows file path style).

1.3 Requirements

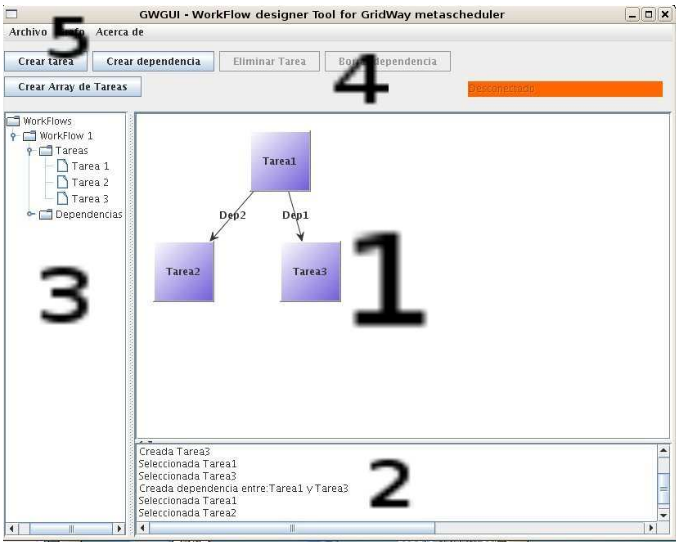
Needed Java Runtime Engine of Sun 1.5.0+ and Unix like operating systems.
Note: Selection of local files within task edition doesn't work on Windows Operating

Systems.

1.4 Main screen

The Figure 1.1 shows the main screen, that is divided in several parts:

1. **Main panel** it allows you to create, to select and to erase tasks and dependencies.
2. **Log panel** it will show all made actions over the graphical interface.
3. **Lateral panel** it shows a hierarchical tree that holds all the open workflows within the application.
4. **Task bar** it holds the create and erase buttoms for tasks and dependencies.
5. **Menu bar** it holds two menus with options for opening, saving and closing work flows. Also options for generation of scripts, uploading them to a Gridway server and even launching them into execution.



```
Creada Tarea3
Seleccionada Tarea1
Seleccionada Tarea3
Creada dependencia entre:Tarea1 y Tarea3
Seleccionada Tarea1
Seleccionada Tarea2
```

Figure 1.1: Main screen sections.

Figure 1.2 shows a open File Menu.

1.5 File menu

1.5.1 Create new WorkFlow

A new workflow will be created and represented in both lateral and main panel by a blue square. This new workflow will have the focus of the application.

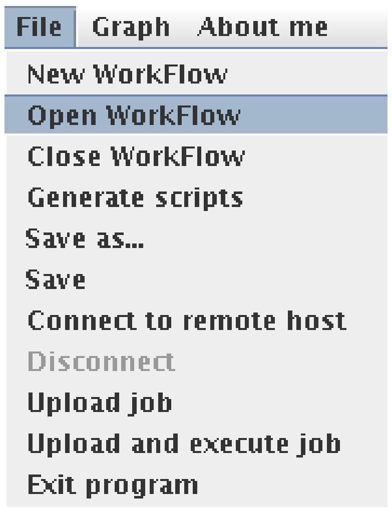


Figure 1.2: File menu.

1.5.2 Open WorkFlow

A File chooser window will be opened to the user allowing him to select a file with that holds predesigned workflows from previous sessions. The file could be stored with any extension but they have an internal format that is tested during the workflow opening. If the internal format was not valid, or holds inconsistent information, an error will be shown to the user.

1.5.3 Close WorkFlow

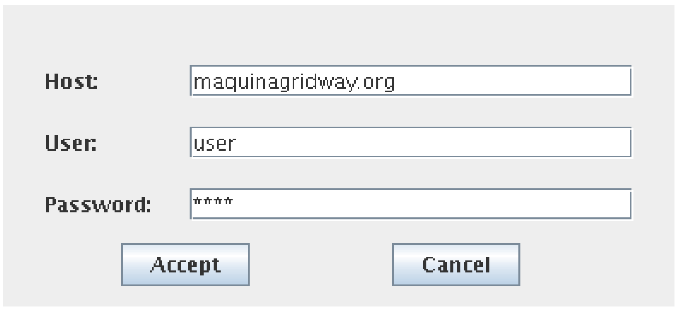
The current selected workflow will be closed, firstly asking the user for it.

1.5.4 Save and Save as

The current selected workflow will be saved by the user.

1.5.5 Connect and Disconnect

The user will create connections with a remote terminal. This connection will be ssh, and will be used for sending files and commands to the remote server with Gridway. The Figure 1.3 shows the edition of the connection parameters made by the user.


Host: maquinagridway.org

User: user

Password: ****

Figure 1.3: Edition of Connection parameters

1.5.6 Generate scripts

The task dependency diagram is processed to generate job templates for *Gridway* and scripts.

1.5.7 Sending jobs

The generated job templates and scripts will be sent to the remote Gridway server. Selected local files will be sent to the folder selected in the remote node.

1.5.8 Sending jobs and launching them into execution

All the previous steps will be down again, and further more, jobs will be launched into execution in the *Gridway* node if all environment variables are well defined on it, and it exits a valid Globus proxy.

1.6 Graph Menu

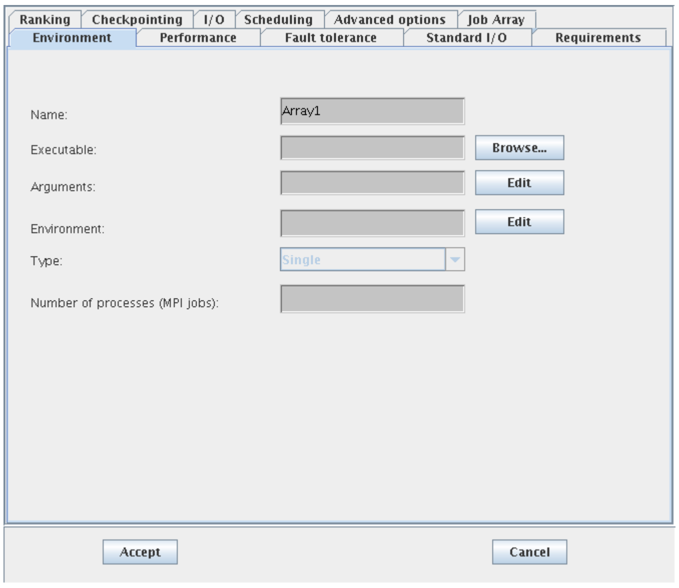
It have the option to reorder graphically the graph, so it would be better readable by the users.

1.7 Task editor

Users can open the task editor performing a double click over a task on the main panel or through the edition option in the context menu. The task editor allow users to config every aspect of *Gridway* tasks.

1.7.1 Environment Tag

It allows users to specify the executable name, arguments and more related task information. The Figure 1.4 shows the *Environment Tag*.



Name: Array1
Executable:
Arguments:
Environment:
Type: Single
Number of processes (MPI jobs):

Figure 1.4: *Environment Tag*

1.7.2 Performance Tag

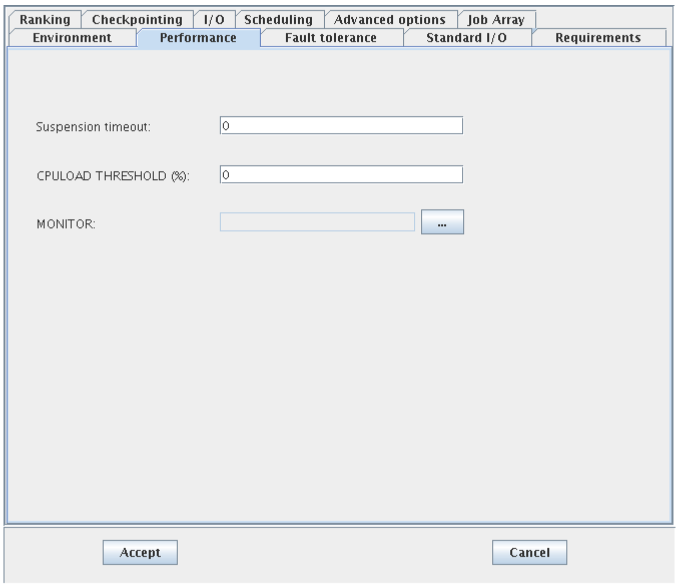
It allows users to specify task performance related parameters. These parameters are used for task migration configuration. The Figure 1.5 shows the *Performance* Tag.

1.7.3 Fault Tolerance Tag

This tag contains the parameters related to fault tolerance in Gridway. The Figure 1.6 shows *Fault Tolerance* Tag.

1.7.4 Standard Streams Tag

It allows users to select what files would be used as standard input and output to the program instance. The Figure 1.7 shows the *Standard streams* tag.

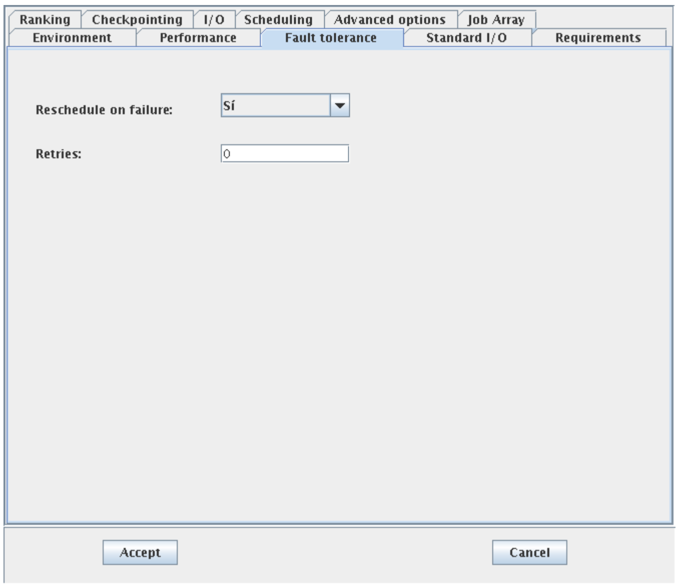


Suspension timeout: 0

CPULOAD THRESHOLD (%): 0

MONITOR:

Figure 1.5: *Performance Tag*



Reschedule on failure: Sí

Retries: 0

Figure 1.6: *Fault Tolerance Tag*

1.7.5 Task Requirements Tag

It allows users to configure the *requirements* task for Gridway. A sintax analyzer is included to test if user inputs are recognized by the *requirements of Gridway* grammar. The Figure 1.8 shows the *Task Requirements Tag*.

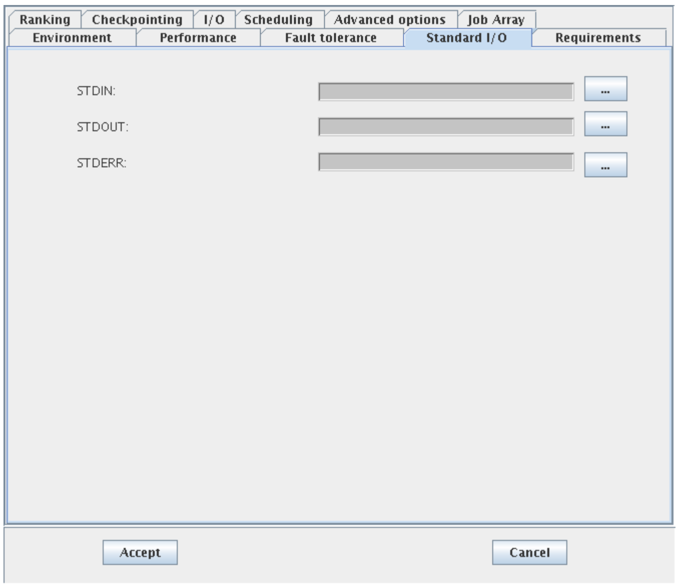


Figure 1.7: *Standard streams Tag.*

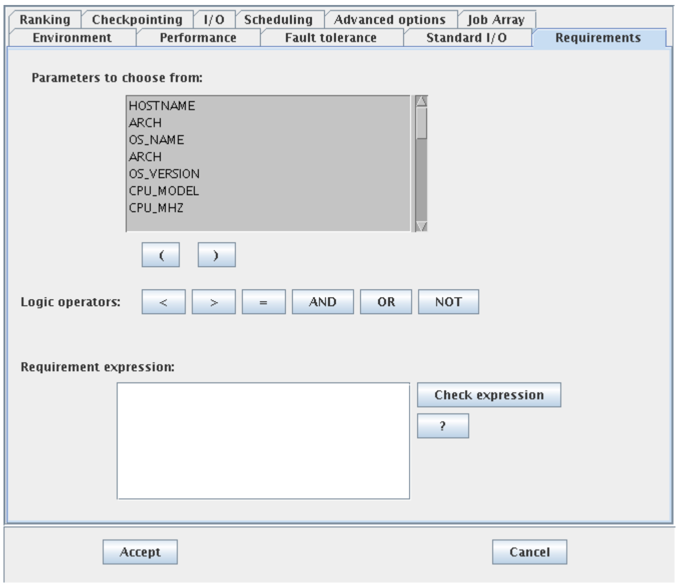


Figure 1.8: *Task Requirements Tag*

1.7.6 Task Ranking Tag

It allows users to configure the *Task ranking parameter* for Gridway. A sintax analyzer is included to test if user inputs are recognized by the *ranking of Gridway* grammar. The Figure 1.9 shows the .

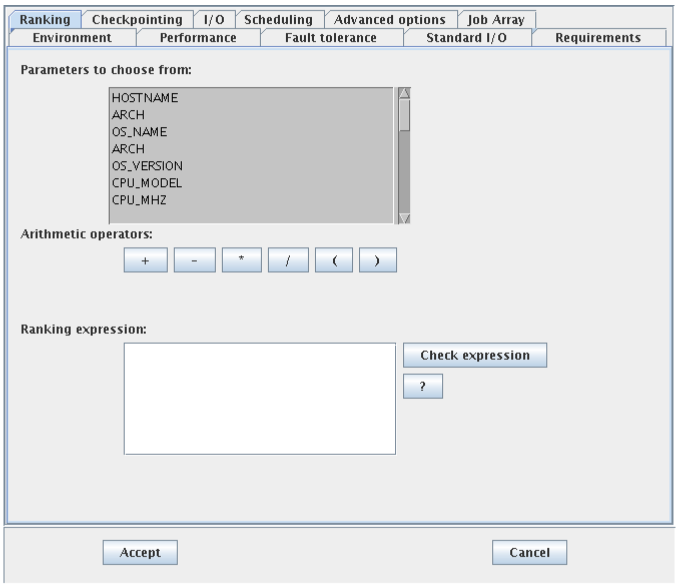
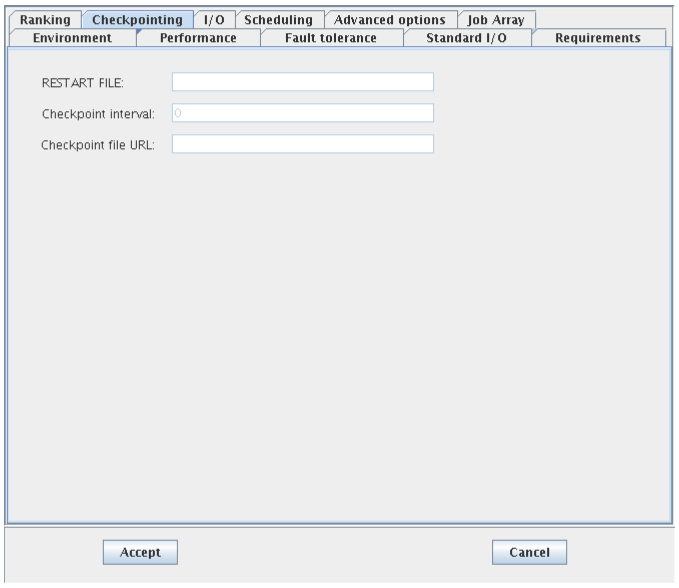


Figure 1.9: *Task Ranking Tag*

1.7.7 Checkpointing Tag

It contains the proper parameters to configure the checkpointing file for the executable instance and the URL where this file is located. The Figure 1.10 shows the *Checkpointing Tag*.



RESTART FILE:
Checkpoint interval: 0
Checkpoint file URL:

Figure 1.10: *Checkpointing Tag*

1.7.8 I/O Tag

It allows to define the files that would be used as input or output to any task. The files could be chosen from the local file system in the client host or in the remove Gridway file system. If the user choose a file that is in a remote host, a popup window will be shown in order to establish a new connection with a Gridway server. The output files for tasks always are defined in the remote file system because these would be copied there when the task finishes its execution and file-staging steps are done. The local files specified in this tag would be copied to the remote server automatically when the user *sends* the work to the remote server through this interface. The Figure 1.11 shows the *I/O Tag*.

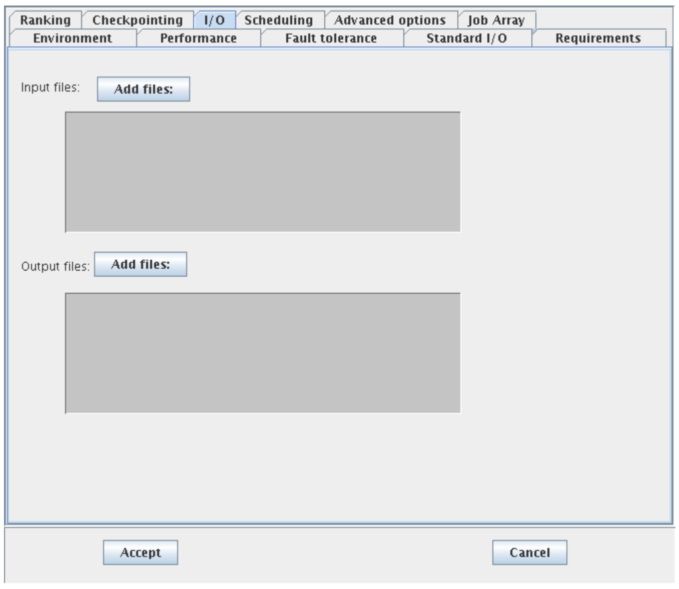
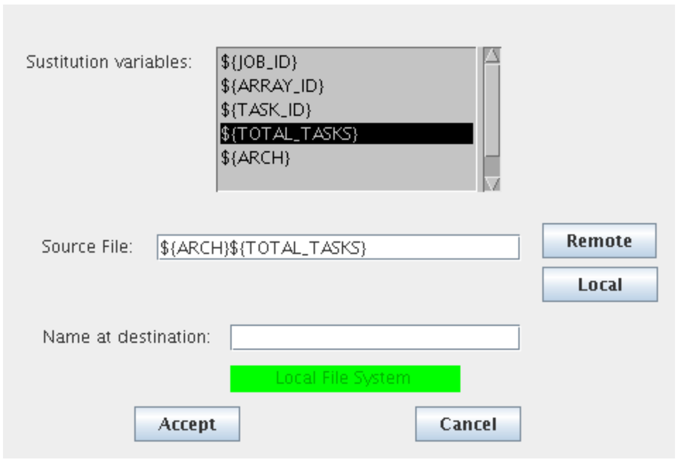


Figure 1.11: *I/O Tag*.

The user is notified during the task I/O edition if a file that appears within a dependency is modified. The Figure 1.12 shows the dialog box for editing task input files. **Note:**
RELATIVE PATHS FOR LOCAL FILES ARE RELATIVE TO THE GWGUI APPLICATION, NOT TO THE WORKFLOW FILE.

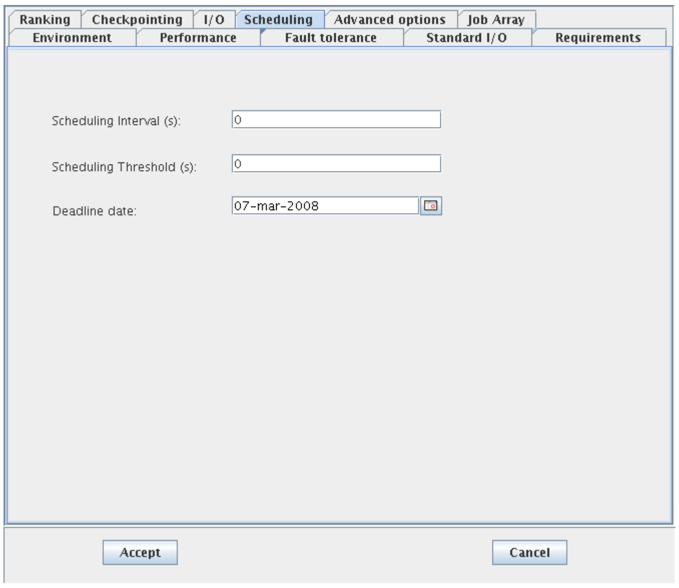
1.7.9 Scheduling Tag

The user can establish the deadline date or the scheduling interval parameter for Gridway, in case of task migration. The Figure 1.13 shows the *Scheduling Tag*.



Source File: ${ARCH}${TOTAL_TASKS}

Figure 1.12: Task input files Editor.



Scheduling Interval (s): 0

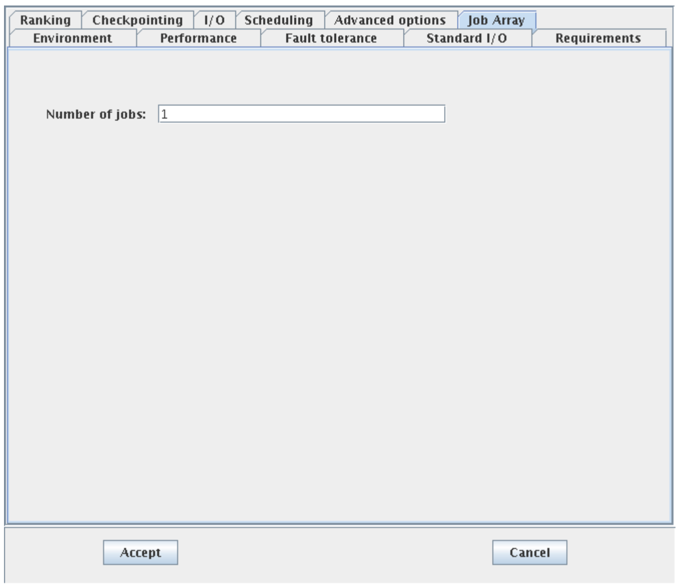
Scheduling Threshold (s): 0

Deadline date: 07-mar-2008

Figure 1.13: *Scheduling Tag.*

1.7.10 Task Array Tag

It only appears while configuring a task array. It allows to configure the number of tasks executed withing the context of a task array. The Figure 1.14 shows the *Task Array Tag*.



Number of jobs: 1

Figure 1.14: *Task Array Tag.*

1.8 Dependency Editor

The Dependency editor allows users to edit the files involved in one dependency. Dependencies between tasks become from the need from these tasks of data generated in other tasks. Information about files involved in the dependency are shown. The files that appears in the dialog box are output files from the first task that could be needed by the second tasks. The Figure 1.15 shows the dependency editor that can be accessed from the context menu by selecting a dependency via one click.

1.9 Execution

This application is implement as a heavy client in Java, so you must execute a *jar* file. The application comes with a directory named *lib*, it is necessary that this directory was in the same path of the *jar file*.

java -jar gwgui.jar

1.10 NAS Grid Benchmarks with GWGUI

The *Nas Grid Benchmarks (NGB)* is a Benchmark developed from the NAS Parallel Benchmarks, that was conceived in 1991 with the objective of evaluating massive parallel systems, very extended in 90s, although none of them follows standards as people

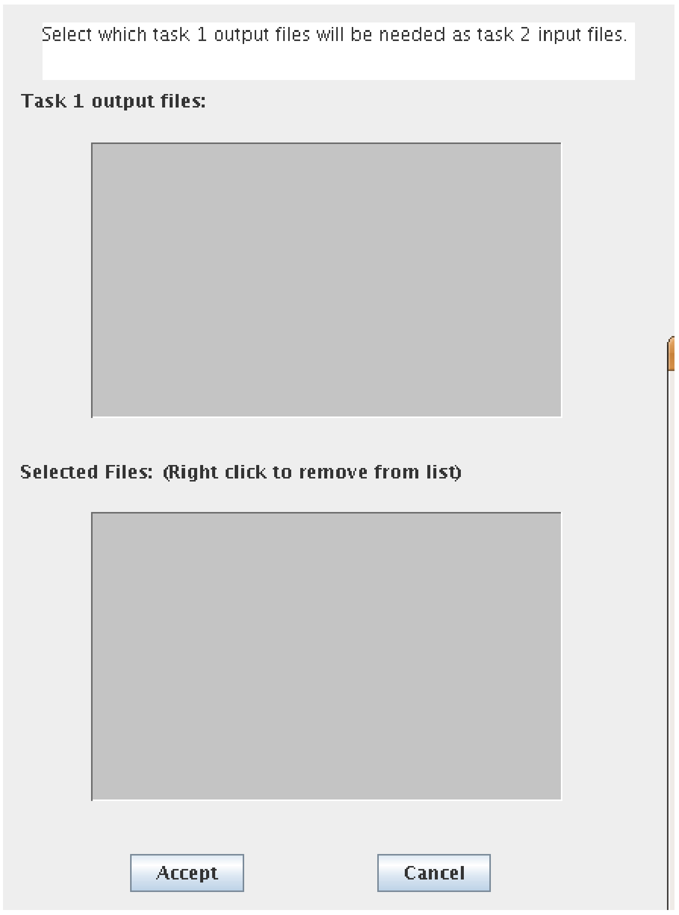

Select which task 1 output files will be needed as task 2 input files.
Task 1 output files:
Selected Files: (Right click to remove from list)
Accept
Cancel

Figure 1.15: Task dependency Editor

does nowadays. Intensive computation tasks were chosen for making up this tests, most of them derived from *NASA* fluid dynamics programs. The first version of the benchmark consisted in a specification of these benchmarks, that let the programmer the low level aspects of the implementation, very dependent on their concrete architecture. The reference code was written in Fortran, so some changes must be done in order to adapt it to a specific architecture. The *NGB* were developed years later for testing distributed architectures such as Grid infrastructures and High performance networks. This tests were developed with some of the original *NPB* tests, but using only tests that could be easily decomposed into independent tasks that doesn't need data from another tasks or if it needs, the amount of data transfered was minimal. These are the Grid favorites ones, due to the high latencies between computing elements in Grid environments, that usually use Wide Area Networks (WAN) for connections.

Description of *NGB*

The NAS Grid Benchmarks consist in a set of fluid dynamics high computation tasks. There are several sets, the implemented are:

- **Embarrassingly Distributed (ED)**. In the jargon of parallel computing, an embarrassingly parallel workload (or embarrassingly parallel problem) is one for which no particular effort is needed to segment the problem into a very large number of parallel tasks, and there is no essential dependency (or communication) between those parallel tasks.

In other words, each step can be computed independently from every other step, thus each step could be made to run on a separate processor to achieve quicker results.

A very common usage of an embarrassingly parallel problem lies within graphics processing units (GPUs) for things like 3D projection since each pixel on the screen can be rendered independently from each other pixel.

Embarrassingly parallel problems are ideally suited to distributed computing over the Internet (e.g. SETI@home), and are also easy to perform on server farms which do not have any of the special infrastructure used in a true supercomputer cluster.

Embarrassingly parallel problems lie at one end of the spectrum of parallelization, the degree to which a computational problem can be readily divided amongst processors. The Figure 1.16 shows a ED tasks diagram.

- **Visualization Pipeline (VP)**. It consists in a set of fluid dynamics programs: BT, MG and FT. Each one needs data from the other ones for starting its processing over the data. There is some parallelism between tasks that can be exploited in a distributed computing system, like Grid. The Figure 1.17 shows the VP tasks diagram.

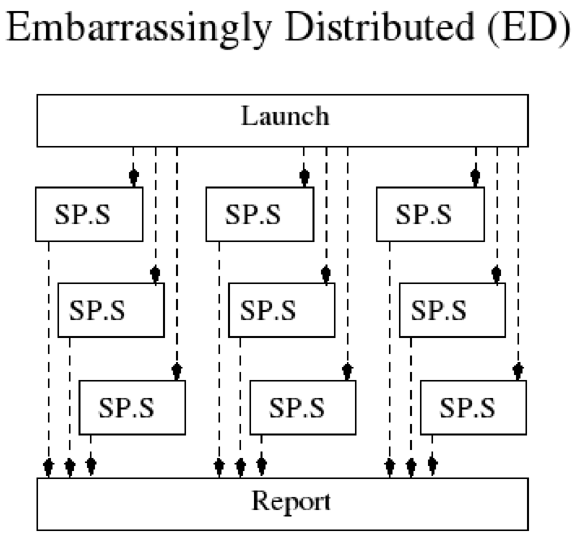


Figure 1.16: ED Benchs

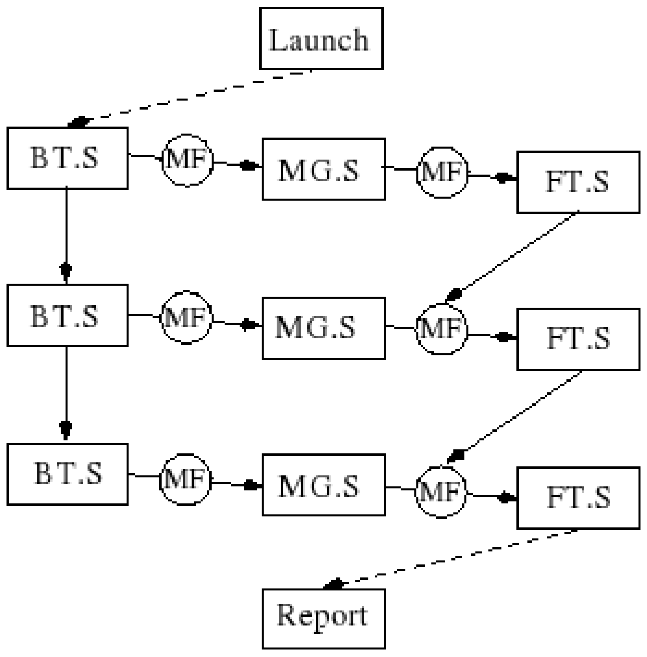


Figure 1.17: VP Benchs

Implementation with GWGUI

These both benchmarks have been modified to be used with GWGUI software. There are two files VP.workflow and ED.workflow. Each one contains the description of the task dependency diagram of each bench. In the Figure 1.17 you can see the task dependency diagram of the VP test. You can explore through the context menus all the features of each task, such as input files, output files, executable, task type, etcetera. The Figure 1.16 shows the task dependency diagram of the ED test. It consits in a unique task divided into nine subtasks. Each parametric task has its own parameter, starting from zero to nine. You can test both tests with your own Gridway Server by connecting GWGUI to it and uploading and executing jobs.

IMPORTANT: Globus environment variable, Globus proxy, and Gridway environment variable must be set up properly for the current user!.

Note:
The *needed* folder must be in the same path as the *GWGUI* main application, in order to execute the workflows *ED* and *VP*. This it is necesary because of the relative paths of the I/O specified files for the tasks involved in these workflows.

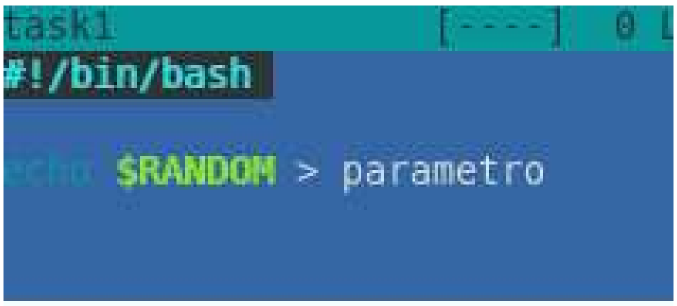
1.11 Quickstart Guide

In this section we are going to create a very simple workflow that does a very simple stupid task. This workflow is composed of four tasks.

- **Task 1**. It creates a random number between 1 and 100 and saves it to a file.
- **Task 2**. It uses the number created by task 1 and generates a file with the phrase *This is* .
- **Task 3**. It uses that number created by task 1 and generates a file with the phrase *a test!*.
- **Task 4**. It uses the files generated by tasks 2 and 3 and generates a file appending the content of these files.

1.11.1 Creating the scripts

It's necessary to write a very simple script for the four tasks. The code for these tasks can be found in the Figures 1.18, 1.19, 1.20 y 1.21.

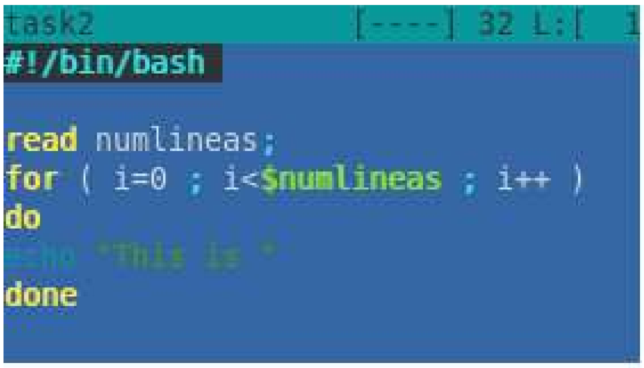

```
task1                    [----]   0 L
#!/bin/bash

echo  $RANDOM > parametro
```

Figure 1.18: Task1 code

1.11.2 Creating an empty workflow project

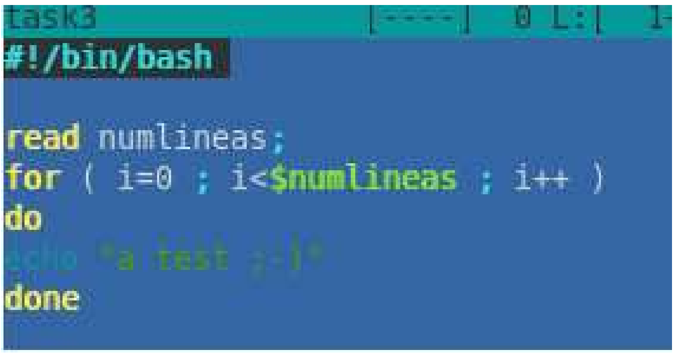
The Figure 1.22 shows the application just before creating an empty workflow project.



```
task2                [----] 32 L:[  1
#!/bin/bash

read numlineas;
for ( i=0 ; i<$numlineas ; i++ )
do
echo "This is "
done
```

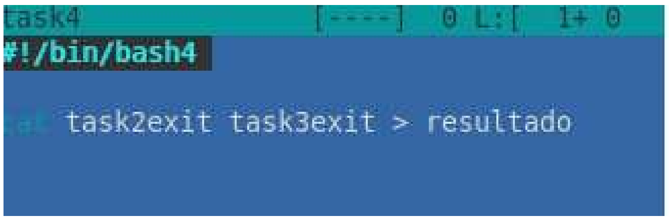
Figure 1.19: Task2 code



```
task3                [----]  0 L:[  1
#!/bin/bash

read numlineas;
for ( i=0 ; i<$numlineas ; i++ )
do
echo "a test :-)"
done
```

Figure 1.20: Task3 code



```
task4                [----]  0 L:[  1+ 0
#!/bin/bash4

cat task2exit task3exit > resultado
```

Figure 1.21: Task4 code

1.11.3 Creating the tasks in the recently created workflow project

The Figure 1.23 shows the application just before creating the four tasks. The next steps will be to configure all these tasks.

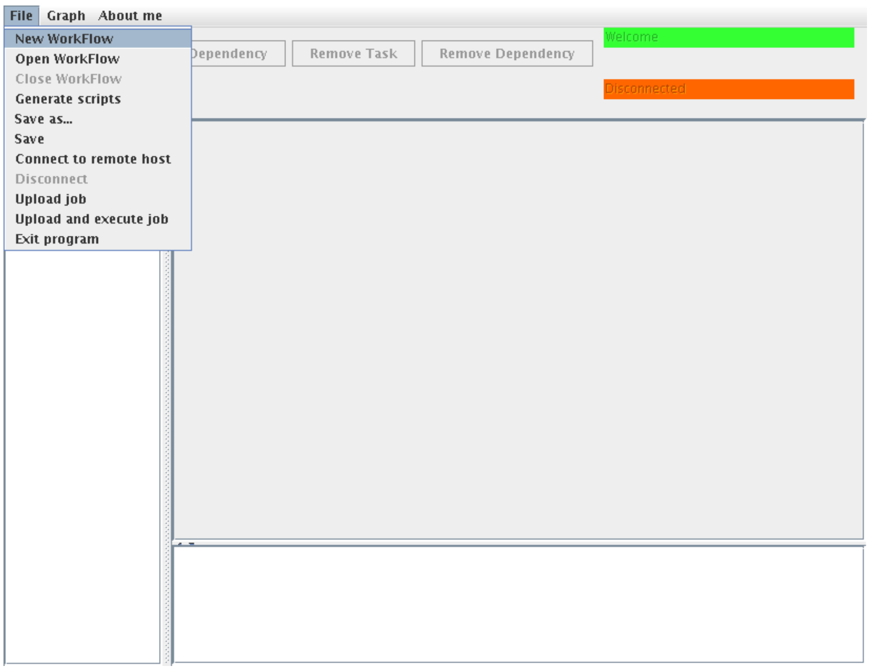
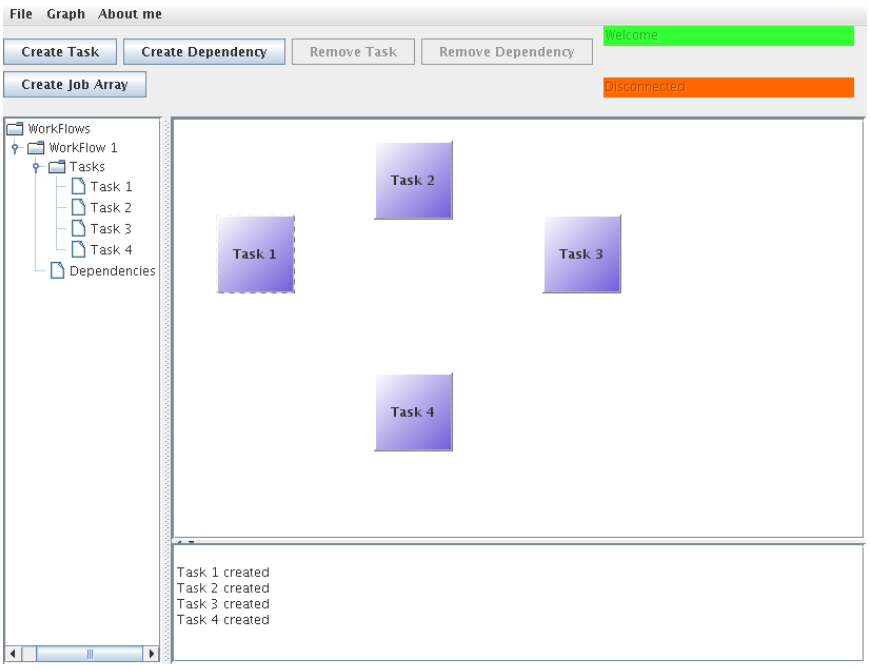


Figure 1.22: Creating an empty workflow

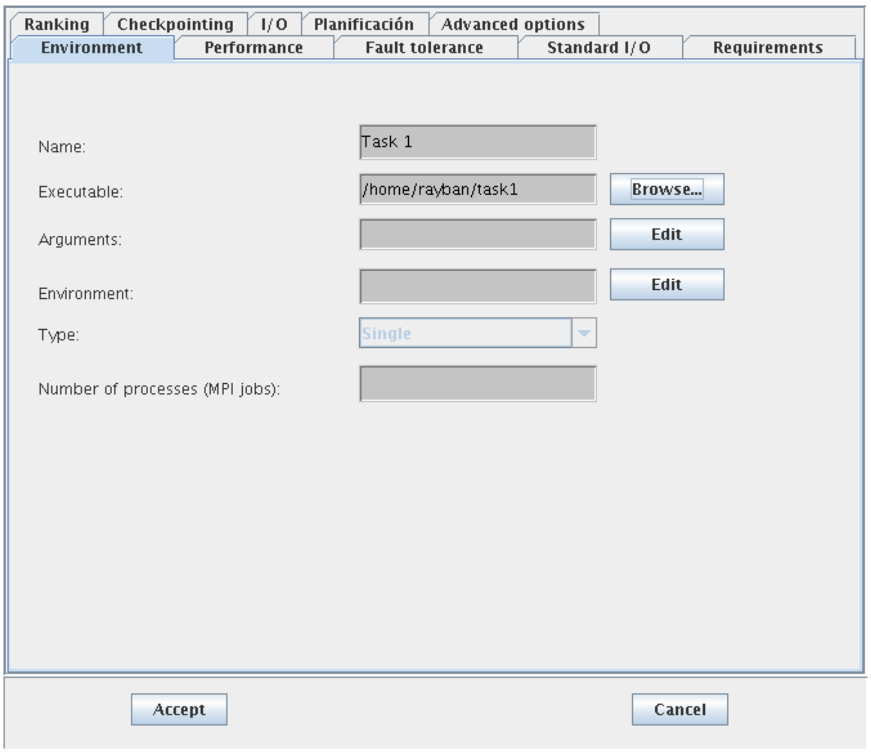


```
Task 1 created
Task 2 created
Task 3 created
Task 4 created
```

Figure 1.23: Four tasks created

Configuring task 1

The task 1 has one executable, the script seen in Figure 1.18, no parameters, but one output file: *parametro*. The *environment tag* must be as seen in Figure 1.24.



Name: Task 1
Executable: /home/rayban/task1
Arguments:
Environment:
Type: Single
Number of processes (MPI jobs):

Figure 1.24: *Environment Tag* of Task 1.

Configuring task 2

The task 2 has one executable, the script seen in Figure 1.19, also it has an input standard stream, the file *parametro* created by task 1. The Figure 1.25 shows the *I/O stream tag* for task 2. Also it has an output stream *task2exit*.

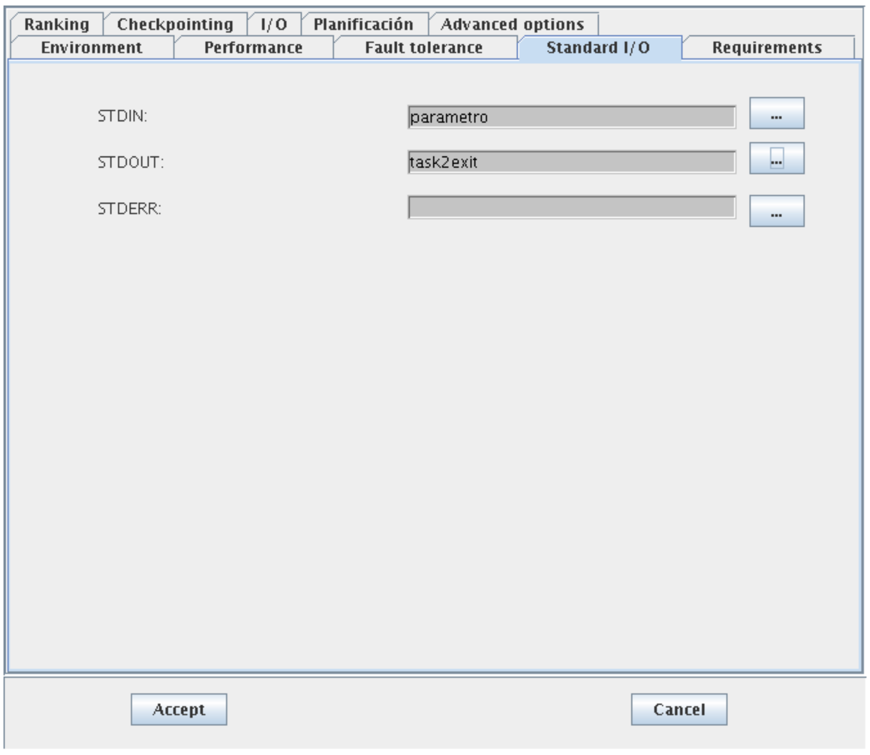
Configuring task 3

The task3 has one executable, the script seen in Figure 1.20, also it has an input standard stream, the file *parametro* created by task 1. The Figure 1.26 shows the *I/O stream tag* for task 3. Also it has an output stream *task3exit*.

Configuring task 4

This task needs the files created by task 2 and 3, so it needs two input files: *task2exit* and *task3exit*. Also it needs and output file with the result of the workflow execution. The *I/O tag* is shown in the Figure 1.27.

The Figure 1.28 shows the workflow formed with 4 tasks and dependencies. The next step is to send it to a Gridway Server.



STDIN: parametro

STDOUT: task2exit

STDERR:

Figure 1.25: *I/O stream Tag* of Task 2.

1.11.4 Generating Gridway job templates and job scripts

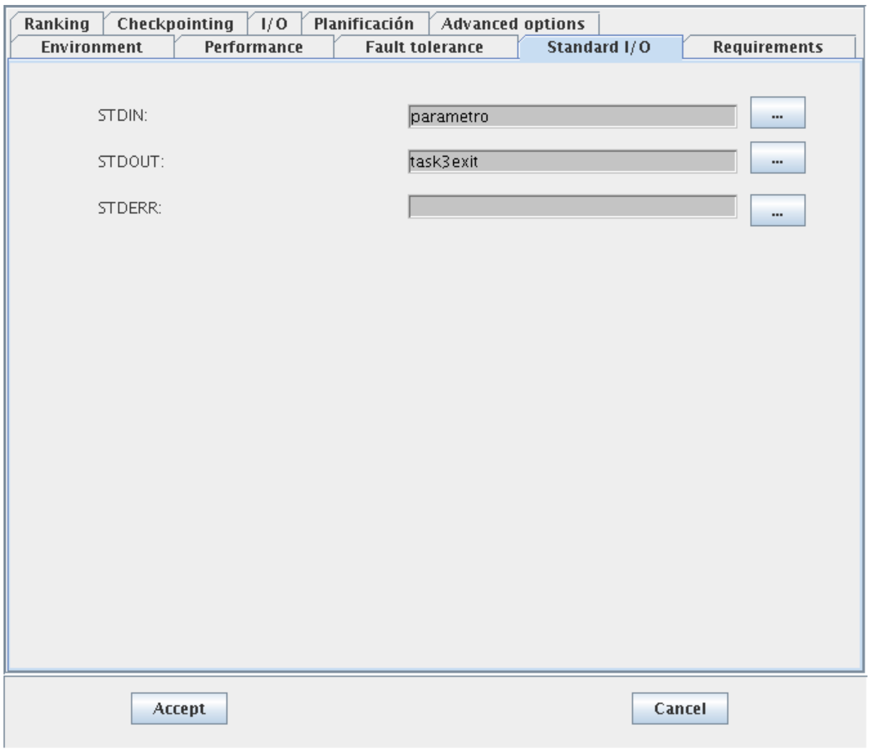
Once the workflow is properly configurated, we are going to generate the job templates and job script. In the File Menu there is an option: *Generate scripts*. Click on it and choose the location of the scripts. The Figure 1.29 shows the job templates and job script generated by *GWGUI*. The following code is the shell script for launching the tasks and also it has expressed the dependencies between tasks.

```
#!/bin/sh

Task1_12345=`gwsubmit -v -t  Task 1.jt | cut -f2 -d ':'`
Task2_12345=`gwsubmit -v -t Task2.jt -d "$Task1_12345 "| cut -f2 -d ':' `
Task3_12345=`gwsubmit -v -t Task3.jt -d "$Task1_12345 "| cut -f2 -d ':' `
Task4_12345=`gwsubmit -v -t Task4.jt -d "$Task3_12345 $Task2_12345 "| cut -f2 -d ':' `
echo $Task1_12345
echo $Task2_12345
echo $Task3_12345
echo $Task4_12345
```

1.11.5 Uploading scripts

The scripts can be uploaded manually to a Gridway server or through the *GWGUI* application. Choosing the option *Upload Job* from File Menu, it's possible to generate scripts in a temp folder and then uploading it in an autonomous way to the



STDIN: parametro

STDOUT: task3exit

STDERR:

Figure 1.26: *I/O stream Tag* of Task 3.

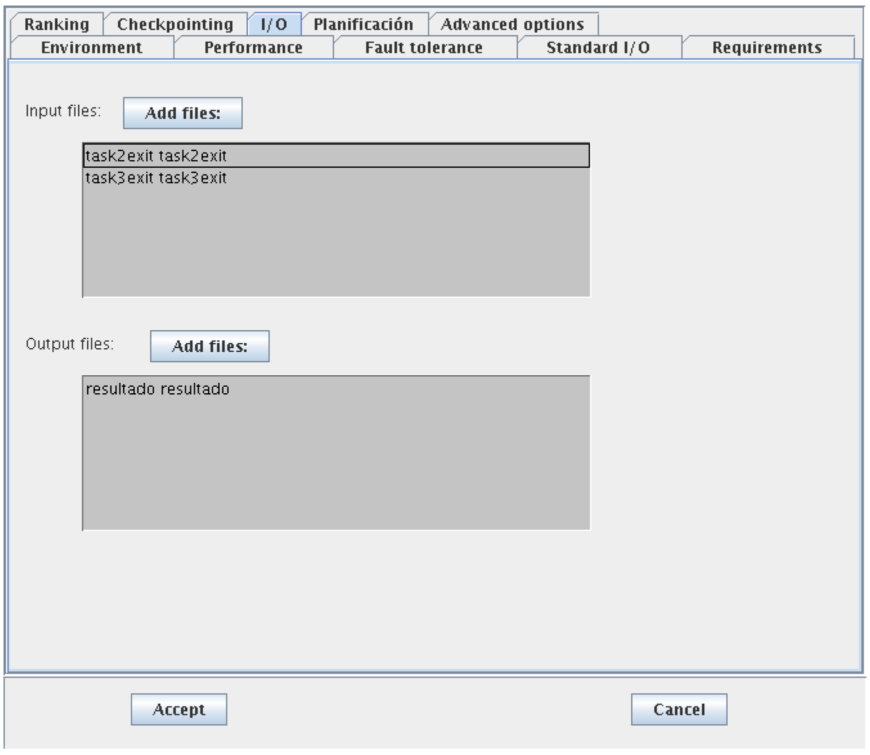
Gridway server.

1.11.6 Executing job

Once the scripts are generated and uploaded to a Gridway server is necessary to give execution privileges to the file recently uploaded with *.sh* extension, and then executing it. This files contains the appropriate parameters for Gridway metascheduler (job templates and dependencies). The job can be generated, uploaded and executed in a autonomous way through *GWGUI*. This can be done by choosing the option *Upload and execute job* from File Menu. This will generate the scripts in a temp folder, it will upload the files to a remote server and then it will check if *Globus Environment* and *Gridway Environment* have been set up properly.

1.12 Troubleshooting

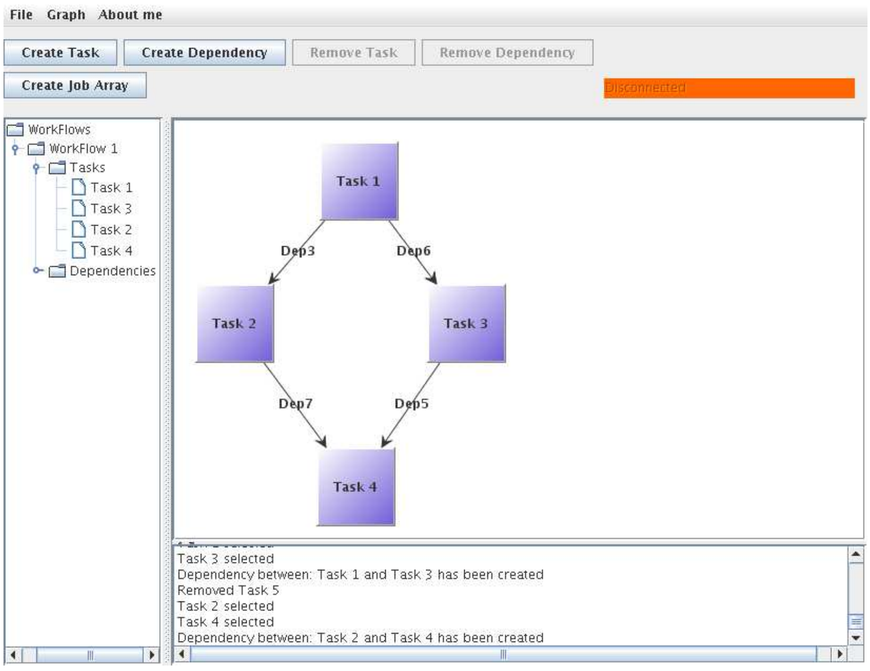
GWGUI is only in its second beta so some errors could be found in many areas. Also, some features are disabled until the next version that will arrive soon. I will be happy to get notice some of errors that may appear so together we can improve this application. Let me know whatever you want, errors, suggestions, improvements,



Input files:
task2exit task2exit
task3exit task3exit

Output files:
resultado resultado

Figure 1.27: *I/O Tag* for Task 4.



```
Task 3 selected
Dependency between: Task 1 and Task 3 has been created
Removed Task 5
Task 2 selected
Task 4 selected
Dependency between: Task 2 and Task 4 has been created
```

Figure 1.28: Workflow with the 4 tasks.

etcetera, to rayban1984@hotmail.com.

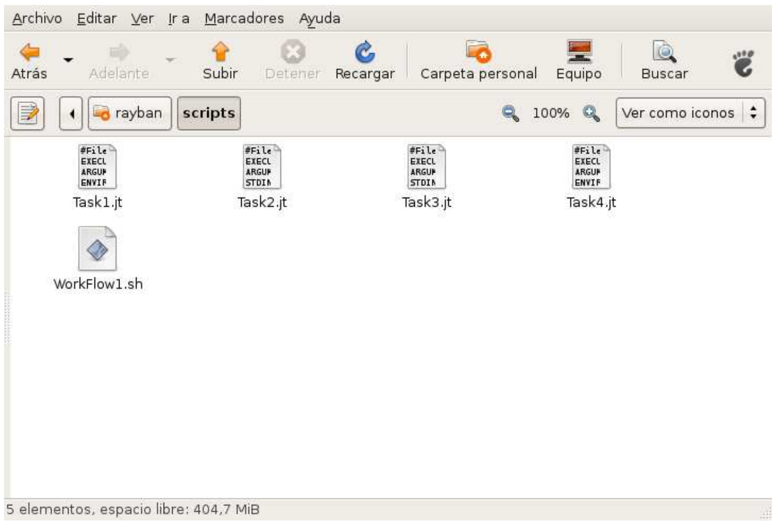


Figure 1.29: Job templates generated for the workflow.

1.13 Next features

This features will be added to *GWGUI* in next builds:

- Support for connections through proxy servers.
- Multilanguage support.
- Context help.
- More enabled features (disabled in this version).
- What you tell me...

Greetings and enjoy the software!.