AALBORG UNIVERSITY
FALCULTY OF ENGINEERING AND SCIENCE
DEPARTMENT OF COMPUTER SCIENCE

Fredrik Bejers Vej 7E , DK-9220 Aalborg

INTEGRATING NORDUGRID SYSTEM WITH UPPALL VERIFICATION TOOL

Project group B1-207
SSE4
June 2004

Preface

This report is our Master Thesis in Computer Science at Aalborg University, Department of Computer Science, Distributed Systems and Semantic Unit.

Since our 3rd semester, our project has dealt with D-Uppaal and NorduGrid. During our last semester, we implemented own experiments running Distributed version of Uppaal on NorduGrid to verify large systems. With the experience from last semester, we could conclude that NorduGrid is a scientific method to implement verifying large systems, which deal with huge resources.

And in this semester, we have continued our work towards providing a convenient user interface for the users to implement verifying large systems.

Acknowledgement:

We are grateful to Josva Kleist for supervising our project throughout our semesters of investigating NorduGrid, and for many inspiring and constructive discussion about the concepts of NorduGrid and its advantages.

And, in addition, we would like to thank Gerd Behrmann, Ulrik Larsen for their help with Distributed Uppaal.

Aalborg University, June 2004.

--------------------------------------	---
Dong Liu	Hoang Thi Thu Huong

Contents

CHAPTER 1: INTRODUCTION

Nowadays, when application systems are bigger and more complex, they require extra computing power including resources (hardware and software) to solve huge problems; however clusters are expensive to acquire and run, especially if one only needs the computing power from time to time. New technology called Grid with its architecture opens a new approach to bring a lot of computing power to the desktop. With one computer connecting to Grid system, it can be easy to extend more resource including software and hardware from many other clusters in many others locations not only local, implement some works that deal before with using local resources.

So what is Grid? How can it bring more computing power to the desktop? What is different between Grid and other current technologies? Can we use any current technology to extend resource, not Grid? And how users interact with a grid system? These questions will be answered in the next chapter – Background – when we introduce about Grid technology more detail.

However, besides the advantage – easily extend resource, current Grid systems are still difficult for any new Grid user because of command line system that requires the users must have intimate knowledge about the usages of the grid system. Alternatively, Grid system is independent with application, then, with the ordinary application users, using Grid system is not as simple as they expected. It will be more efficient if Grid technology is integrated with the application as one utility of application, automatically run some steps, reduce complexity and bring more comfort to the users.

Imagine, when we integrate application with grid systems, the users only need to choose on utility on application, application will automatic connect with Grid, do some tasks (which before, users must run on commands system), and return the last result. The new system will bring more comfort to end-users, instead by the current system, in which the users must be trained some knowledge to use Grid with a quite complex commands from connect to Grid system to get back the results from system. So, if grid technology is to be used by ordinary application users then it must be expected to integrate with the application as a replacement for two independent systems.

Based on this idea, in this project, we try to integrate Grid with one application named model checking using Uppaal, which is a tool suite for validation and symbolic model-checking of real-time systems based on timed automata and it is developed jointly by Uppsala University and Aalborg University. The problem of Uppaal is deal with lacking of huge resources when verifying larger systems. This can be solved by Grid system – it is demonstrated by our last semester project: Grid can help us to verify large systems [01].

In the previous work, we did some experiments to prove usefulness of Grid systems in looking for available resources for verifying large systems using D-Uppaal (distributed version of Uppaal) on NorduGrid, which is a lightweight Grid solution. From the results of experiments, we concluded that Grid could help us to verify large systems. On the other hand, during the time we worked with Grid, all the steps from connect to NorduGrid to get results from system had been executed by using command system. This proceedings suggested us integrate application with Grid system then it is easier for end-user in using.

So, in sum, the motivation of our semester project is integrate D-Uppaal (the distributed version of verification system Uppaal) with NorduGrid. The new system allows the users to verify large systems using NorduGrid resources and this task can be done by Uppaal user interface, by one button click, not by commands system as before.

The basic definition, architecture and user guide of NorduGrid, Uppaal and D-Uppaal will be also described in the next chapter, in which we explain clearly why we choose these systems for our experiment. From this knowledge, users have one general view about the essential of integrating application with Grid technology.

CHAPTER 2: TECHNOLOGIES

In this chapter, we present about some technologies, which we use to implement experiment on this project. That is some general information about Grid, NorduGrid, Uppaal and D-Uppaal verification systems.

In addition, we will explain why we choose them, how to work with these technologies. This chapter is the background for designer and programmer for integrating NorduGrid and Uppaal user together.

2.1 Grid

As we concerned in Introduction, Grid can help us to extend resource for applications that deal with huge resource requirement. So the first question is "What is Grid?"

Grid refers to technologies and infrastructure that enable coordinate resource sharing and problem solving in dynamic, multi-institution virtual organization [02].

With Grid system, we can use own computer not only to look for the data without taking care where the data is stored as Web technology but also to get back from system the results, which can be done by any other available cluster on Grid system.

To bring these above features, Grid technologies comprise protocols, services and tools include:

- Security solution to ensure virtual organization sharing rules.
- Resource management protocols and services that support when looking for resources to solve problems.
- Information query protocols and services provide attributes about resources, organizations and services.

But can we use any current technology to extend resource, not Grid? Based on theory, Web also can help us to provide a global distributed system allowing sharing resources. Why we need new technology as Grid?

Even the current Web technologies address communication and information exchange among computers but it does not provide integrating approaches to the coordinated use of resources at multiple sites for computation. This is one of real advantage of using Grid and we try to promote in this project.

In this report, we would like to insist on advantage of Grid through on the business view: As we know, resource, which includes hardware and software, is expensive and scarce. To solve one big problem, which deal with resource as memory, software we can go out and order a new computer that has already required software, but, after solving problem, we will not use all of its power even the price to buy this computer is not cheap. That is not the best way to save money, when economic is the important goal.

On the other side, if we have one computer, which connect to Grid system, even that computer is not strong; we still can resolve huge problems by submitting job to Grid. This Grid technology allows system automatically looks for available resource to crack your problems and return results. That is really what we would like to be done.

So, with Grid system, it is easy to access to huge resource that satisfy the request criteria, use every capability of resources.

However, Grid is independent system with application. The users, who want to submit job on Grid system, must have some knowledge about it, especially the command system from connect to Grid to get back the results. That is not quite easy processing as the users expected. In the next part, we will describe some basic commands to do these tasks.

In this project, we choose NorduGrid to work with. In the next part, we will describe some information about it and how users interact with it.

2.2 NorduGrid

In this project, we will work with NorduGrid, and with NorduGrid, the computing resources that can be accessed through different sizes clusters from many different locations in the world.

In this project, we only need to learn about how the users interact with NorduGrid system. So, we will introduce the basic command system to interact with NorduGrid, from submitting job to getting back results from NorduGrid system.

To start working with NorduGrid, firstly, the user must be NorduGrid user. This can be done by install NorduGrid Client and get authentication to access Grid resources. We assume that the user has been NorduGrid user already. So the last thing the user needs to know when working with NorduGrid is commands system to submit job and get back result from this system.

2.2.1 grid-proxy-init

To start grid proxy, we use ***grid-proxy-init*** command. After start grid-proxy, user can submit job on NorduGrid during the default time is one day.

When running this command, user must input his NorduGrid password. System will check username and password, if the user is NorduGrid user, and password is right, then, system connected to NorduGrid, otherwise, system will display error for user.

Options:

-help, -usage	Displays usage
-version	Displays version
-debug	Enables extra debug output

-q	Quiet mode, minimal output
-verify	Verifies certificate to make proxy for
-pwstdin	Allows passphrase from stdin
-limited	Creates a limited globus proxy
-independent	Creates a independent globus proxy
-old	Creates a legacy globus proxy
-valid <h:m>	Proxy is valid for h hours and m minutes
-hours <hours>	Deprecated support of hours option
-cert <certfile>	Non-standard location of user certificate
-key <keyfile>	Non-standard location of user key
-certdir <certdir>	Non-standard location of trusted cert dir
-out <proxyfile>	Non-standard location of new proxy cert

2.2.2 grid-proxy-destroy

After working with NorduGrid, user can disconnect by this command. It will destroy grid-proxy and user cannot submit any job on NorduGrid except he connects again by ***grid-proxy-init*** command.

Options:

-help, -usage	Displays usage
-version	Displays version
-debug	Display debugging information
-dryrun	Prints what files would have been destroyed
-default	Destroys file at default proxy location

2.2.3 xRSL file

To submit jobs on NorduGrid, the user must write an ***xRSL*** (extended Resource Specification Language) file, in which specify job requirements and parameters for submission. The constructor of ***xRSL*** file is similar to scripts for local queuing systems, but includes some additional attributes as job name, executable location and parameters, location of input and output files of the job, architecture, memory, disk and CPU time requirements...

This is one ***xRSL*** example (with minimum necessary parameters)

File test.xrsl:

```
& (executable = test.sh)
  (jobname = test)
  (stdout = test.out)
  (stderr = test.err)
```

However, besides those basic parameters, in ***xRSL*** file can have more parameters, which influence to the efficient of processing job as number of CPUs, Runtime Environments.

2.2.4 ngsub

With ***xRSL*** file, users can submit job on NorduGrid:

Syntax: ***ngsub*** [option] [xrsl]

Example: ***ngsub –f test.xrsl*** (submit job in ***test.xrsl*** file on NorduGrid)

Options:

-c, -cluster [-]name	explicity select or reject a specific cluster
-C, -clustlist [-]filename	list of clusters to select or reject
-g, -giisurl url	url to a central GIIS
-G, -giislist filename	list of GIIS urls
-f, -file filename	xrslfile describing the job to be submitted
-o, -joblist filename	file where the jobids will be stored

-dryrun	add dryrun option to the xrsl
-dumpxrsl	do not submit - dump transformed xrsl to stdout
-t, -timeout time	timeout for MDS queries in seconds (default 40)
-d, -debug	debuglevel
-x, -anonymous	use anonymous bind for MDS queries (default)
-X, -gsi	use gsi-gssapi bind for MDS queries
-v, -version	print version information
-h, -help	print this help

2.2.5 ngstat

To obtain the status of all jobs, we use ***ngstat*** command

Syntax: ***ngstat*** [option] [job]

Example ***ngstat –a*** (obtain status of all user’s job)

Options: -a, -all	all jobs
-i, -joblist filename	file containing a list of jobids
-c, -cluster [-]name	explicity select or reject a specific cluster
-C, -clustlist [-]filename	list of clusters to select or reject
-s, -status statusstr	only select jobs whose status is statusstr
-g, -giisurl url	url to a central GIIS
-G, -giislist filename	list of GIIS urls
-q, -queues	show information about clusters and queues
-l, -long	long format (more information)
-t, -timeout time	timeout for MDS queries in seconds (default 40)
-d, -debug debuglevel	0 = none, 1 = some, 2 = more, 3 = a lot
-x, -anonymous	use anonymous bind for MDS queries (default)
-X, -gsi	use gsi-gssapi bind for MDS queries
-v, -version	print version information
-h, -help	print this help

The status of job will be listed as follow:

Accepted:	job submitted but not yet processed
Preparing:	input files are being retrieved
Submitting:	interaction with LRMS ongoing
Finishing:	output files are being transferred
Finished:	job is finished
Canceling:	job is being cancelled
Deleted:	job is removed due to expiration time

2.2.6 ngget

After finishing job, if users would like to get back the results from NorduGrid system, they can use ***ngget*** command.

Syntax: ***ngget*** [option] [job]

Example: ***ngget –a*** (see all result of all jobs, which user submitted on NorduGrid)

Options:

-a, -all	all jobs
-i, -joblist filename	file containing a list of jobids
-c, -cluster [-]name	explicity select or reject a specific cluster
-C, -clustlist [-]filename	list of clusters to select or reject
-s, -status statusstr	only select jobs whose status is statusstr
-dir	download directory
-j, -usejobname	use the jobname instead of the short ID
-keep	keep files on gatekeeper (do not clean)
-t, -timeout time	timeout for MDS queries in seconds (default 40)
-d, -debug debuglevel	0 = none, 1 = some, 2 = more, 3 = a lot
-x, -anonymous	use anonymous bind for MDS queries (default)
-X, -gsi	use gsi-gssapi bind for MDS queries
-v, -version	print version information

-h, -help	print this help

2.2.7 ngkill

When user wants to cancel his jobs, running this command can do it.

Syntax: ***ngkill*** [option] [job]

Example: ***ngkill –a*** (cancel all user's job on NorduGrid)

Options:

-a, -all	all jobs
-i, -joblist filename	file containing a list of jobids
-c, -cluster [-]name	explicity select or reject a specific cluster
-C, -clustlist [-]filename	list of clusters to select or reject
-s, -status statusstr	only select jobs whose status is statusstr
-keep	keep files on gatekeeper (do not clean)
-t, -timeout time	timeout for MDS queries in seconds (default 40)
-d, -debug debuglevel	0 = none, 1 = some, 2 = more, 3 = a lot
-x, -anonymous	use anonymous bind for MDS queries (default)
-X, -gsi	use gsi-gssapi bind for MDS queries
-v, -version	print version information
-h, -help	print this help

That is some basic commands, which the users usually use to submit jobs and get back results from NorduGrid. Besides that, NorduGrid supports more commands as Capturing job status (***ngcat***), Re-submitting jobs (***ngresub***), Cleaning up after jobs (***ngclean***).

As we can see, it is not simple to submit job on NorduGrid, from connecting NorduGrid system, writing ***xRSL*** file, submitting job to getting back results. All of these steps will be easier for the end-users if it is automatically. And, that is also the motivation of this our project.

2.3 Uppaal

The fundamental idea behind the verification tool Uppaal is to model a system as a network of timed automata and test the model for invariant and reachability properties. The tool is appropriate for systems that can be modelled as a network of communicating processes.

Uppaal consists three main elements as following:

- System Editor: allows the user to describe and edit the timed-automata system. The system timed automata consists of global declarations, a timed-automaton templates, process assignment and system definition sections.
- The Simulator allows the user to virtually interact with the system described. The simulator shows the system state by displaying the states of compound automata and the values of variables. The simulator allows the user to choose enabled transitions manually or randomly. It also has a feature of displaying the history of events in sequence chart.
- The Verifier accepts the user formulated properties to be verified on a particular timed automata model, and displays the result of verification: true or false depending on whether the property was satisfied or not, and an event trace example if the property proof requires one.

There are two ways to use Uppaal: the graphical user interface or the command line program ***verifyTA***. Input of Uppaal is the system specification, which consists of a network of processes that are composed of location. The simulation will run interactively the system to check that it works as intended. Then we can ask the verifier to check reachability properties, i.e., if a certain state is reachable or not [03].

However, as many other verification tools, the current version of UPPAAL deals with explosion problem (the size if state-space grows exponentially in the number of concurrent components in the model). D-Uppaal (distributed version of Uppaal) is one interesting method that can help the users to improve this weakness of Uppaal when verifying large system.

So why can D-Uppaal do such thing? This question will be clarified in the next part, and that is why we choose D-Uppaal engine for our new system.

2.4 D-Uppaal

When verifying systems, the verification system must do a lot of calculations data. A faster computer can calculate more data than a slower computer in the same time. But, another solution is connecting many slow computers and makes them calculate the data together and this solution also can reduce time. The interest in parallel and distributed algorithm of D-Uppaal allows us to run on one cluster and we can verify larger systems more than using Uppaal.

The real advantage of using NorduGrid is that the users can get access to clusters having compute power larger than a single workstation, but the power can only utilized if the users run a parallel version of Uppaal because the users can get as much processing power as expected from Grid system. This feature helps us to use every capability of Grid system.

That is also the reason why we choose running D-Uppaal engine (***dvserver***) on NorduGrid, not Uppaal engine (***verifyTA***).

Current version of D-Uppaal is working as following:

- Using Uppaal user interface for modelling system, and input all queries, save in two files: ***.xml*** and ***.q*** file.
- Using convert program (independent with Uppaal) to convert two Uppaal files (***.xml*** and ***.q*** file) to three D-Uppaal input files (***model.xml***, ***status.xml*** and ***job.xml***)
- Create a new ***models.list*** file, which contains name of ***model.xml*** and ***job.xml*** files.
- Using D-Uppaal engine (***dvserver***) to verify systems.

The new version of D-Uppaal will get 2 input files as Uppaal files (***.xml*** and ***.q*** file) and the way it works will be the same with ***verifyTA***. That means:

- Using Uppaal user interface for modelling system, and input all queries, save in two files: ***.xml*** and ***.q*** file.
- Using D-Uppaal engine (***dvserver***) to verify systems with two parameters is two above files. After that, we will get results directly from system, not in ***status.xml*** file as current version of D-Uppaal.

In this project, we design a new system, which uses D-Uppaal new version engine, not current version.

CHAPTER 3: REQUIREMENTS ANALYSIS

From the idea that is bringing more comfort to the users when using application on NorduGrid system to use maximum resources power, in this project, we try to implement D-Uppaal on NorduGrid to verify large systems but not as prior project. We modify current Uppaal source code to allow user run our utility directly from the user interface. Before designing and programming, we must analyze requirements, based on which we can design the new system that provides the best utilities for users.

3.1 No commands system

To start imagining how new system works; we view how the current system works to verify large systems using D-Uppaal engine (***dvserver***) and NorduGrid (as we did in the last semester project). Here, we assume to use new version of D-Uppaal. The new version of D-Uppaal engine will execute as the current Uppaal engine, get two file ***.xml*** and ***.q*** file as parameters.

First, we need to use the Uppaal verification tool to construct a formal model of system that represents its possible behavior. After that, to validate the properties of that system, we input the queries. Two above tasks are done by Uppaal user interface. They must be saved in two files: ***.xml*** file and ***.q*** file.

After that, to submit verifying jobs on NorduGrid, we must go to commands system, create a new ***xRSL*** file, which describes jobs to be submitted with executable file is ***dvserver***, input files is ***.xml*** and ***.q*** file. And if the users wonder to prove the efficient of jobs processing, they need to declare some more parameters (that we concerned in ***xRSL*** structure in NorduGrid, last Technologies chapter).

Now, the users are ready to submit jobs on NorduGrid, this work can be done by some more commands from connecting to NorduGrid (***grid-proxy-init***), submit jobs (***ngsub***) to get result back (***ngget***). These commands have many options, which are not easy to remember all. And, not all the time, submitting jobs is successful. When the users write ***xRSL*** files not right format, they must do it again.

Compare with current Uppaal, to verify one property, the user choose and click on '***Model Check***' button on the user interface. Uppaal will return result in a while. We can see how complicate to verify large system using D-Uppaal on NorduGrid, the users must work with many commands. With this disadvantage, the users prefer to integrate new utility, which allows them to submit verifying job on NorduGrid and all processing will not by command system as now, but through the new user interface.

Now, we can describe the new system as the users expected: The new system allows users to verify large system from Uppaal user interface. After modeling and input queries, user can choose one query to verify. After a while, the new system will return results on user interface, too. So, users will not work with command system any more.

With the programmer view, this requirement can be analyzed as: To verify large system, we can use D-Uppaal engine (***dvserver***) and NorduGrid resource. But, the users do not want to work with command system, only on the user interface, then, all jobs processing (connect to Nordugrid, generate ***xRSL***, submit job) will run automatically and in the background. What the users can see is the last result on the user interface.

3.2 Using more CPUs

As we know, when verifying large systems, the verification tool requires a lot of resource. And clearly, using two CPUs will bring more processor power and memory than one CPU. It will finish the verification job faster.

Users always expect that their job will return results as soon as possible, so we can do it by increase the number CPUs to process this job. How to do it when submitting job?

In ***xRSL*** file, there is one parameter - '***Count***'. With value of this parameter, system will use the number of CPUs as the value of '***Count***' parameter for job processing. Is it right that this number as big as possible then the job efficient will be better?

In Grid systems, when the user requires a number of CPUs for job, system will look for any cluster that has enough CPUs as required. Not every time, it can be successful at once, job would be queue. During waiting time, if the users choose smaller number of CPUs, job might be processed by another cluster and return results to the users. However, if the number of CPUs is too small, verification task can waste time for get back result.

If we do not control this value for '***Count***' parameter, system will default this value is one. With this value, that can be too small for almost verification tasks of large system.

So, with this analysis, that is better if we can control this parameter value. In the new system, ***xRSL*** files will be generated automatically and we cannot fix this value (as above problems when it is too big or too small). Therefore, we need to program one function, which allows changing this parameter value before submitting job on NorduGrid.

3.3 Implement many tasks in the same time

Verifying some properties in the same time is what users often do in Uppaal system, so, when developing the new system, users also expect to be able to verify two or more properties simultaneously.

Instead of waiting the results of verification tasks, the users can choose others properties to submit verifying job at that time. When programming and testing, we need to be sure that the new system will not lock after submitting one job. If that, users cannot do others verification in a long time (if verification task takes a long time).

3.4 Security

Only the NorduGrid users can be submit job on NorduGrid. This feature of NorduGrid requires the users must be login before submitting job to NorduGrid. In the first time submitting job, system will ask the users to input NorduGrid password. This function of the new system must be done by user interface in Uppaal, not by command system. So, we must program new function check whether user login to NorduGrid or not. If not, then, require the users input password to connect to NorduGrid. If user has connected, he can submit job after that.

But, when the users close application, new system must disconnect to NorduGrid automatically, then other user cannot open application and submit job on NorduGrid.

With this requirement, when the user closes application, we must check status of grid-proxy. If system disconnected to NorduGrid, then, we no need to do anything. Otherwise, system must be disconnected to NorduGrid automatically.

3.5 Advance feature …

Not only number CPUs impact to submitting jobs on NorduGrid, but also these other values and parameters as cluster, disk, runtime environment, architecture. If new the system can provide function that support for the users to change value of those parameters for promoting all power of NorduGrid, that will be very nice for the users, who has knowledge about those system variables.

This requirement suggests us to program one new independent function in current Uppaal, this function allows users change value of those parameters, which can impact to efficient of jobs processing as above. From this function, advance users easily control these parameters by the user interface. So, all interact with NorduGrid system can be done not by command system as before, but by user interface. We hope this system will be nice as users expected.

CHAPTER 4: DESIGN

After analyzing all user requirements, we will start designing the new system based on the idea of the last chapter. This chapter will describe the design of the new system including system architecture, user interface, system process and classes.

From the first user requirement, as in the last chapter, we analyzed is that new system will have one more function, which allows the user to verify large system using D-Uppaal engine (***dvserver***) and NorduGrid resources. As a result, we get the following the new system architecture as follow:

4.1 System architecture

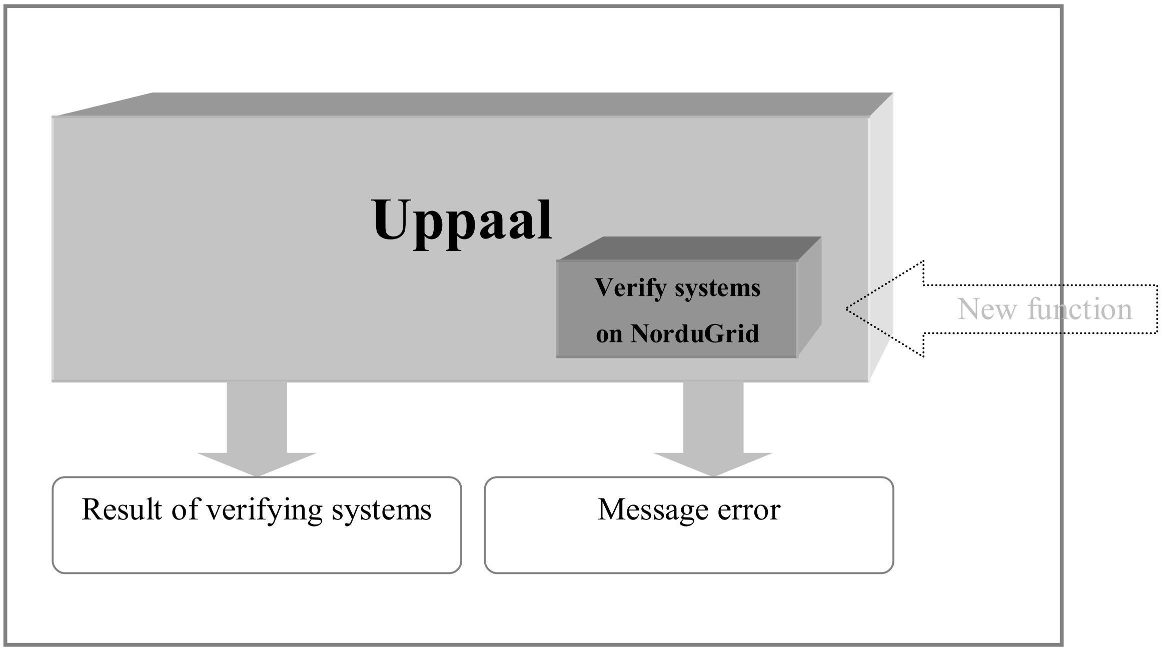


Figure DES-01: System architecture

Description:

New function here is verifying large system on NorduGrid and use D-Uppaal engine (***dvserver***). This function will be added to the user interface of Uppaal, so users can choose it for verifying large systems.

After the users choose this function, system will submit jobs on NorduGrid, use NorduGrid resources to execute and download results from NorduGrid. The results returned will be the same when using current Uppaal to verify other systems. It will return message ‘Property is satisfied’ or ‘Property is not satisfied’. In case the processing has error, the system will display error for the users.

All the rest of processing with NorduGrid as generating ***xRSL*** file, running commands to submit jobs, check whether jobs have finished yet to download results will be done automatically in the background. Then, the users will not work with NorduGrid through command system any more.

That is general view outside of the new system. The detail of working interface and process will be described in the following:

4.2 User interface

The current Uppaal allows user to check properties of a model by choosing '***Model Check***' button on the user interface. Now, we will add one more option for users to verify large systems using D-Uppaal engine (***dvserver***) on NorduGrid (as the first user requirement) and this function will display on Uppaal user interface.

So, with user's view, this function will work the same way as with '***Model Check***' function, gets system model and query, and displays result on user interface after processing.

With this view, we will create one new button named '***Grid Check***' for the new function. When users click on this button, all the process of interacting with NorduGrid will be run on the background and after getting back results; it will display this result on user interface.

When the users work with NorduGrid, they must be NorduGrid users. System will check it by require user input NorduGrid password. In the new system, this checking will be done by user interface, too. So, base on this analysis, in the new system, we have to change Uppaal source code to satisfy.

4.2.1 Verifier user interface

In tab Verifier of current Uppaal version, we will add one button is '***Grid Check***' as button '***Model Check***' and when user clicks this button; the processing that allows user to verify large systems on NorduGrid will be invoked.

In the current Uppaal user interface, in ***Verifier*** tab, there are four buttons: ***Model Check, Insert, Remove*** and ***Comments***. As we talk above, we will add one button labeled

Grid Check, on which when the users click, the new system allows submitting verifying job on NorduGrid.

After submitting job, processing job, system will return result to user interface, in ***Status*** text box. This result will be the same when using Uppaal to verifying, it will display message ‘Property is satisfied’ or ‘Property is not satisfied’.

4.2.2 Check NorduGrid user password

When one user submits jobs on NorduGrid, the new system will check the authentication of the user (check the status of user proxy certificate). If the grid-proxy has not been created yet or it has expired, the system will ask NorduGrid password. This is the user interface to ask for NorduGrid password:

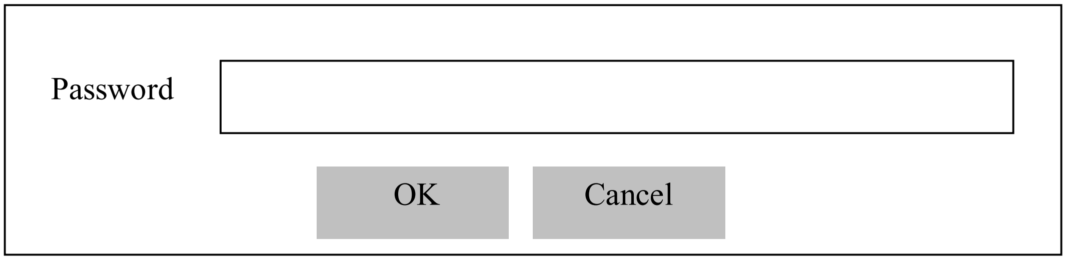


Figure DES-02: Check NorduGrid user interface

<u>Description:</u>

- When starting, the new system will check NorduGrid proxy-status; if it has expired or not created then system will display message “***Your proxy is expired or not created. Please, login again***”
- After message, display interface which is allows user to input password.
- User inputs password to text box and push ***OK*** button, this password will be displayed with format as the “*” character.

- System checks password. If password is valid then message the time proxy valid until and start submitting jobs on NorduGrid, otherwise, display error "***User has no authentication to work with NorduGrid or Password is not valid***". After that, re-display the interface to allow user re-login.
- If user pushes on ***Cancel*** button, then close program.

4.2.3 Grid monitor and xRSL file parameters

In the user requirements analysis, to improve the efficient of job processing, we need to add some more parameters in ***xRSL*** file as number of CPUs, cluster, memory, disk In some special application, we need to specify Runtime Environment (to run D-Uppaal in NorduGrid, we need Runtime Environment is MPICH). So, for some users, it is necessary to have a function that they can change value of these parameters. To satisfy this requirement, we will add one more function on Uppaal user interface for this task.

This utility will be added to menu of current Uppaal user interface. So, in the menu of system, we will add one more column is 'Grid'. It will have two functions: Grid monitor show all information about clusters on NorduGrid and Grid parameters allow users change parameters to specify cluster, number of CPU, memory … before submitting jobs.

4.2.3.1 Grid monitor

When user choose Grid monitor, the new system will display information about all cluster on NorduGrid. The information as alias, location, host certificate, architecture, number of jobs, number of CPUs, middleware, runtime environment… will be displayed as following:

Cluster	benedict.aau.dk	…	…	…
Alias	Aalborg Grid Gateway	…	…	…
Number of CPUs	46	…	…	…

Number of used CPUs	19	…	…	…
Number of running jobs	19	…	…	…
Number of queue jobs	10	…	…	…
Runtime environments	efd-1.0.0.1 localdisk-2.0.0.1 lam-7.0.0.0 mpich-1.2.5.0 atlas-8.0.1.0	…	…	…
Architecture	i686	…	…	…
Middleware	nordugrid-0.4.1.0 globus-2.4.3.9	…	…	…
Memory on each node	1024 MB	…	…	…

Figure DES-03: NorduGrid's cluster information

From this information, the users will know more detail about system, from that, they can choose specific cluster or some other information for submitting jobs.

4.2.3.2 Parameters

As we describe above, this function will allow users changes the parameters when submitting jobs on NorduGrid:

User interface of this function will be as follow:

Cluster	(List all cluster)
Number of CPUs	
Rerun	Yes/ No
Start Time	

Figure DES-04: Parameters

Description:

- The cluster will be listed and one more option is 'No specify'. When users choose specific cluster, display number of CPUs of this cluster. User can change this number. Runtime environment will list all options of this cluster, then user can choose (or choose No Specify option).
- When job submit is failed, if user want to submit it again, choose Rerun option is 'Yes'. Otherwise, choose 'No'.
- 'Start Time' option allow user submit job at anytime, may be, not now. Option 'No Specify' means jobs will be submitted at current time.

These are two changes on current Uppaal user interface. The next part we describe the system process, from that, we can view how the new system work.

4.3 System process

This part will describe the processing flow of new system, which allows users verifying large systems on NorduGrid.

With this processing flow, the programmer can see clearly how the new system will work, and from that we can design detail classes.

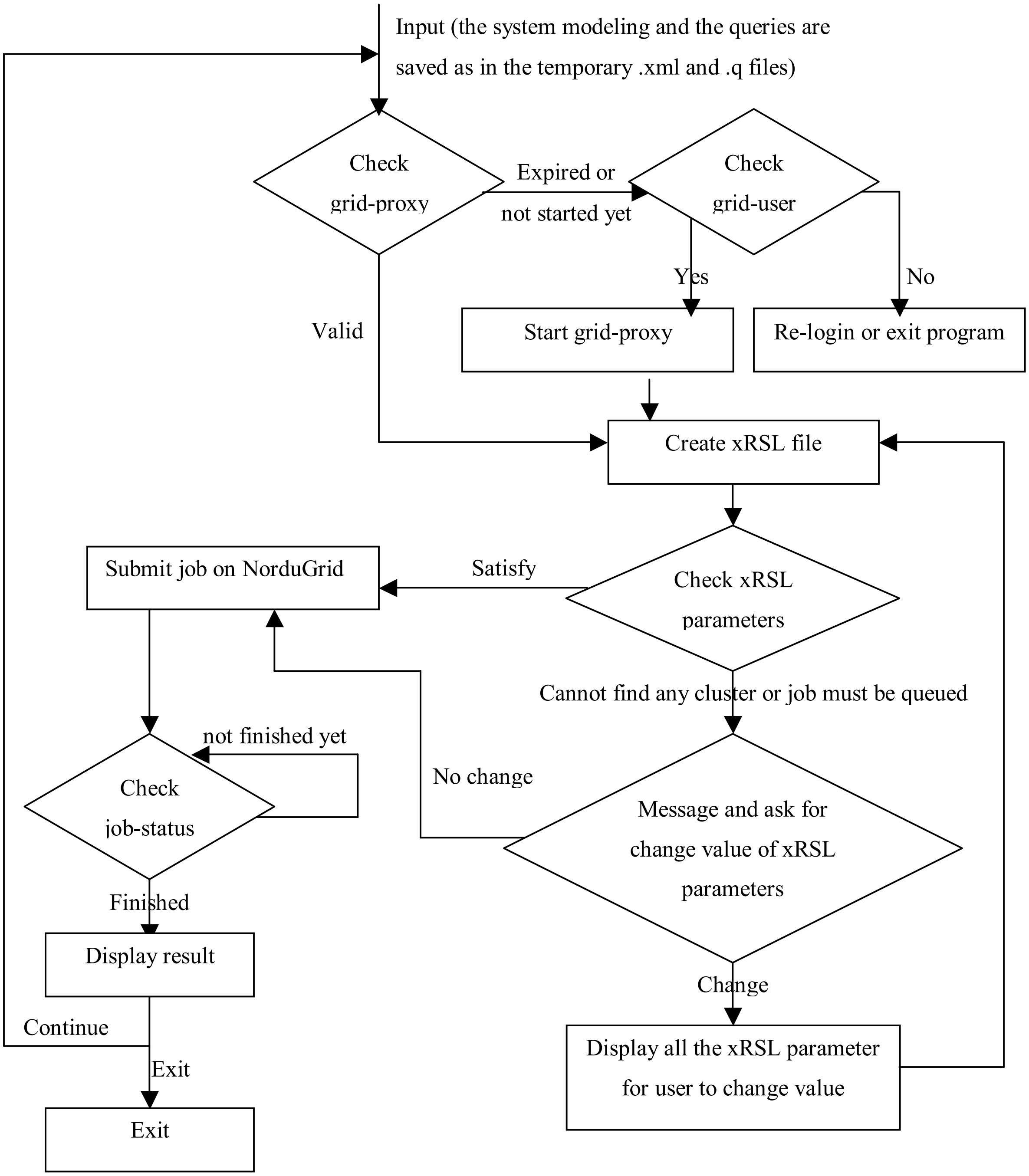

Input (the system modeling and the queries are saved as in the temporary .xml and .q files)
Check grid-proxy
Expired or not started yet
Check grid-user
Valid
Yes
No
Start grid-proxy
Re-login or exit program
Create xRSL file
Submit job on NorduGrid
Satisfy
Check xRSL parameters
Cannot find any cluster or job must be queued
not finished yet
Check job-status
No change
Message and ask for change value of xRSL parameters
Finished
Display result
Change
Continue
Display all the xRSL parameter for user to change value
Exit
Exit

Figure DES-05: System process

Description:

- The new system gets input and saves in temporary files are ***.xml*** file and ***.q*** file in the working folder.
- Check grid proxy, if it is valid then allow submitting job, otherwise, if it has expired or not been created yet, and then ask user permission to init grid-proxy. If the user is not the NorduGrid user, then message error or required to re-login, otherwise, start grid-proxy to submit job.
- Create ***xRSL*** file.
- Check cluster and CPUs number required, amount of memory to run verifying job. Here, there are two cases will be happen:
    - One user could potentially choose a number of CPUs larger than the number of CPUs in the cluster. In this case, this should be forbidden, as the job will not be able to run on the cluster. So, system will display error for users to change the number of CPUs. After user changes it, system will check again.
    - The second case is user might also choose a number of CPUs larger than the number of CPUs free. System will display to ask whether user want to change number of CPUs or waiting in queue. If users accept waiting, then submit job on NorduGrid. If not, display Parameter user interface and user can change parameter here.
- If cluster has enough CPUs then submit jobs on NorduGrid.
- During the processing of jobs, check job-status, display job-status on screen. If the status if ***FINISHED***, then display verifying result.
- After displaying results of jobs, allow user choose exit program or continue verifying other systems (submit other jobs on NorduGrid).

4.4 Classes design

From system architecture, user interface, system process that we design above, now, we will design the classes for programming the new system. Before that, we will analyze what we will do?

As in the system architecture, we will add one more function on current Uppaal user interface. This function will be called when user click on one new button named 'Grid Check'. To add one more button on Uppaal user interface, we need to find out, which class draws this interface. In this class, we will program new source code, which draw '***Grid Check***' button as '***Model Check***' button. And, when user clicks on this button, it will invoke one new class, which allows verifying system on NorduGrid.

Therefore, we have to do:

- Find out which class in Uppaal source code, which draws Uppaal user interface.
- Create a new button in user interface as current '***Model Check***' button.
- When click on this button, it will invoke new class named ***GridVerification*** to verify system on NorduGrid. This new class will be designed and programmed to run on the background, hidden from the user.

To check NorduGrid user authentication, we must program one user interface for user to input password.

To implement as description in user interface design, we have to do:

- In new ***GridVerification*** class, invoke another class call ***CheckTimeOut***, which check status of grid-proxy.
- In this class, run ***grid-proxy-info*** with option ***–timeleft*** to check how long time grid-proxy is valid.

- If grid-proxy is valid then invoke class named ***SinglePanel*** that implements submitting job task on NorduGrid.
- If grid-proxy is not existed or expired then message and display interface for user to input password.
- Get passwords input string as '*' string on user interface.
- Get passwords from ***stdin*** and run ***grid-proxy-init*** with option ***–pwstdin***. This option allows sending password to NorduGrid system directly from ***stdin*** then password will not kept in anyplace on computer to ensure security.
- Check grid-proxy again, if it is valid then invoke class that implements submitting job on NorduGrid.
- Otherwise, message and repeat all steps for user to re-login.

And to add new utility that is Grid monitor and xRSL parameters, we have to do:

- Look for the class draw the menu in Uppaal user interface source code. From that, add more column in menu, that is Grid and it has two small function is Grid monitor and ***xRSL*** parameters.
- When get user activation, invoke new class name ***GridMonitor*** and ***ParameterValue***.
- ***GridMonitor*** class will get all information (this task can be done by command ***ngstat –q-l*** in NorduGrid command system) about cluster on NorduGrid and display as in user interface design.
- After getting all information of clusters, system will display on user interface by ***ClusterInforTable*** class that is invoked in ***GridMonitor*** class.
- ***ParameterValue*** class will get the value of ***xRSL*** parameter in one temporary file call ***parameter.tmp*** and display on the user interface. These values are saved from the last time, when the user changes them. If this is the first time, file ***parameter.tmp*** does not exist, system displays all value as 'No specify'.
- After getting values from file, ***ParameterValue*** class invokes new class name ***ParaValueTable***, which draw table to display as user interface design. The new class allows user to change value of parameters. When user change cluster, system will check whether this cluster exist or not. If not, display error 'this

cluster does not exist. Please choose another cluster, or choose Grid Monitor to see clusters information'. When the user finish changing parameters values, system will save these parameters values in ***parameter.tmp*** file.

That is all the idea to change and create new classes on current Uppaal source code. Now, we will analyze Uppaal source code and from that, we have classes interface for new the systems.

4.4.1 Uppaal code analysis

In Uppaal, there is ***SystemInspector*** class, in which create the action for '***Model Check***' button in ***Verifier*** tab on Uppaal user interface.

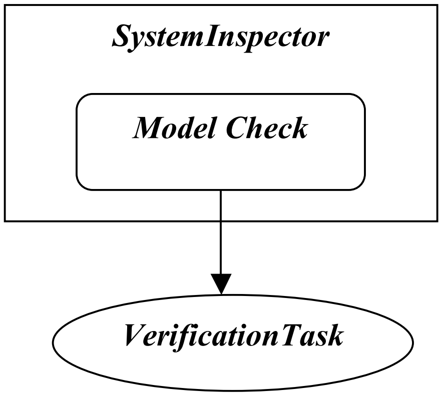


Figure DES-06: ***SystemInspector*** class and ***Model Check*** action

- In ***SystemInspector*** class, it invokes ***GUIAction*** class to draw '***Model Check***' button and invokes ***VerificationTask*** class when get action event to implement verifying task by Uppaal.
- As we analyze above, we have to change in ***SystemInspector*** class.

4.4.2 Idea to program for new system

- Add a new button '***Grid Check***' button on verifier page in the Uppaal GUI.
- Add a new class named ***VerificationGrid*** to deal with the verifying job when get action event from user interface (When user clicks on '***Grid Check***' button, then invoke this class to verify system using D-Uppaal engine and NorduGrid).
- Program new classes to do the detailed functions after submitting grid jobs as check status of grid-proxy, submit job, check status of job and when job is finished, display results to user interface.

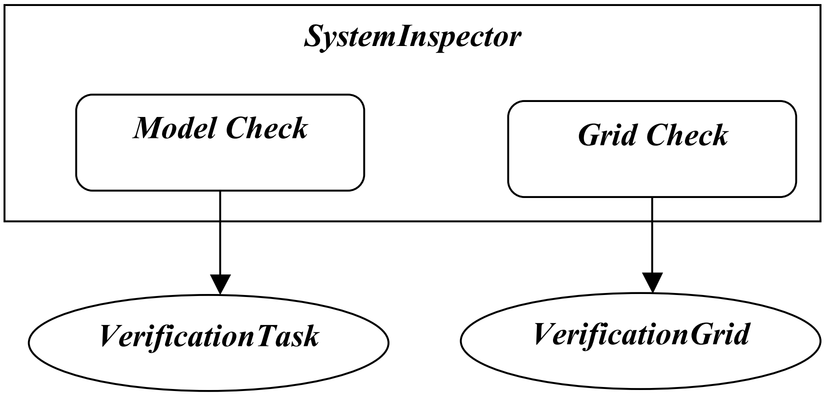


Figure DES-07: The flow of New ***SystemInspector*** class

4.4.3 New classes called by VerificationGrid

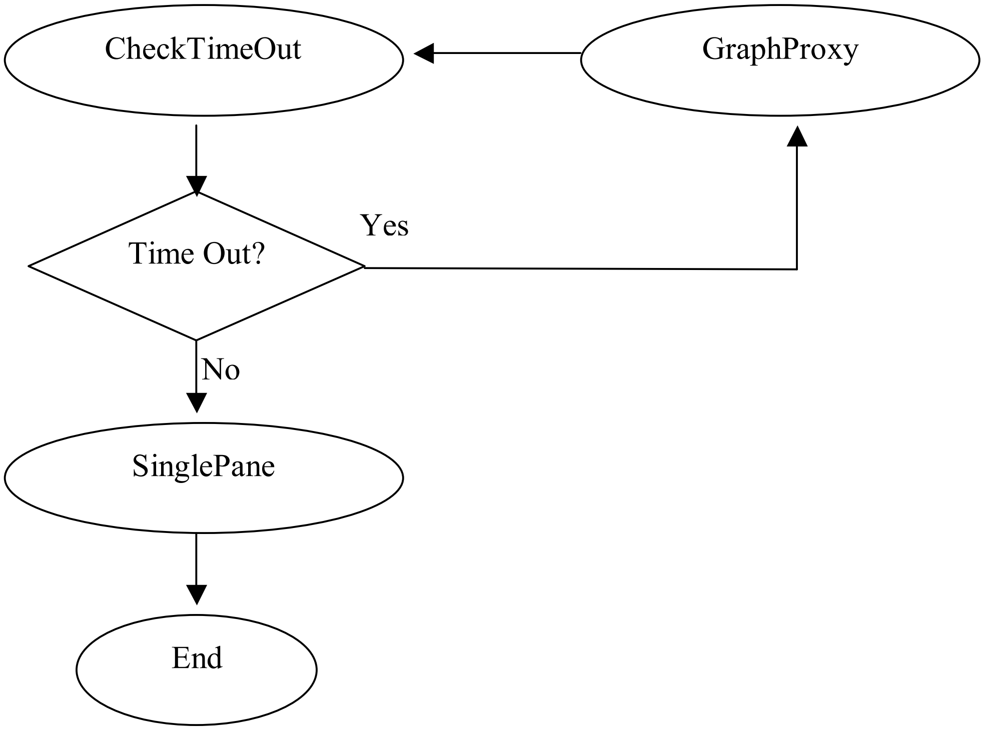


Figure DES-08: New classes flow chart

<u>Description:</u>

Three classes are invoked by method in ***VerificationGrid*** class during the procedure. They are ***CheckTimeOut***, ***GraphProxy*** and ***SinglePanel***.

- ***CheckTimeOut*** class checks whether the grid-proxy status is valid or not when he submit job on NorduGrid. If the proxy is invalid or expired, then call user interface that allows user input password to init grid-proxy.

  Detail:

  - Run grid-proxy-info with option ***–timeleft*** to know current status of grid-proxy.

  - If system returns value -1 then grid-proxy is expired, else if it returns message error then grid-proxy has not been initialized yet, else if value is bigger than 0 then grid-proxy is still valid.
  - In cases of grid-proxy is expired or invalid, invoke class ***GraphProxy*** and allow user input grid password. Otherwise, invoke ***SinglePanel*** class.

- ***GraphProxy*** class will display user interface to get the NorduGrid user's password to run ***grid-proxy-init***. If the password is not accepted then message error, otherwise, user can commit job to NorduGrid by clicking the "***Grid Check***" again.

  Detail:

  - User interface as in Figure DES-03
  - When user input password, change the display format to string of '*' characters.
  - Run ***grid-proxy-init*** with option ***–pwstdin***.
  - Check the status of grid-proxy as the above class, if it is valid, then returns, otherwise, message error that user has no authentication or invalid password.

- ***SinglePanel*** class will do all the rest of work to submit job on NorduGrid and download result to display into user interface:

  Detail:

  - Save as model system and query as temporary files in working folder.
  - Create ***xRSL*** file, the parameters will take from the system parameters.
  - Submit job on NorduGrid.
  - Check job status. When status of job is FINISHED then download result and display to the user interface.
  - Delete all temporary files.

4.4.4 Classes for Grid Monitor

Following are the classes' specification of Grid Monitor

– ***GridMonitorWindow***

This class has two main functions. First, it generates a window for survey nordugrid cluster information. Secondly, it invokes the method of ***CollectClusterData*** class periodically to collect grid status information.

– ***CollectClustersData***

This class not only calls the system command "***ngstat -q -l***", but also provides other methods to analyze the data, which is achieved by former command. Follows are the details of these methods.

– ***CollectClusterInfo***

This method gets and stores the information contains name of cluster, alias of cluster, location of cluster and architecture. Because there are many clusters are working in the NorduGrid at the same time, the method need to distinguish every cluster and remember their names. Based on these names, it calls following methods to get the data of every cluster.

– ***CpuData***

This method gets and stores the information contains type of CPU, number of CPU, number of used CPU. It achieves the number of available CPU by subtracting the used CPU number from total CPU number.

– ***JobData***

This method gets and stores the information contains number of running jobs, number of queued jobs. If the cluster provides the maximum number of running jobs and queued jobs, the method can achieve the number of available running jobs and available queued jobs. If the cluster provides the max number of running jobs per local user, it also stores this datum.

– ***MemoryData***

This method gets and stores the size of memory in each node of the specific cluster.

– ***ScratchData***

This method gets and stores the information of scratch directory such as size and free space in scratch directory in one cluster if it provides.

– ***CacheData***

This method gets and stores the information of cache such as size and free space in cache directory in one cluster if it provides.

– ***RuntimeEnvironmentData***

This method gets and stores the items of runtime environment of the cluster.

– ***MidwareData***

This method collects the items of installed middleware of the cluster.

4.4.5 Class for Parameters

Follows are the class specification of parameters

– ***ClusterSetupWindow***

This class has two main functions. One is to figure out the window of setting up cluster parameter, the other is to list the dialog box of the parameters will be set up. To get the precise value of some parameters, the class needs to invoke the method of ***ReadClusterData***. Some parameters such as rerun times and start time only need wait for user's input.

– ***ReadClusterData***

This method invokes the method of ***CollectClustersData*** class in the back end. It gets the data from ***CollectClustersData***, displays the range of available number in the dialog box. After that, it calls the ***CheckInput*** method. If there is no an alarm flag, it calls the ***SaveParameter*** method. Else, it sends out a warning message to tell user where is wrong in his setting and give him related advice and the chance to reset.

– ***CheckInput***

This method reads users' input parameters and compares them with the data such as the CPU numbers achieved from ***CollectClustersData***. If the user doesn't input the parameters the method will set the default values. If all the parameters are compatible with the collected data, it returns without the alarm flag. If not, the method will save the conflict items and return with an alarm flag.

– ***SetReRun***

This method accepts user's setting of number of rerun and saves it.

– ***SetStartTime***

This method accepts user's setting of start time and saves it.

– ***SaveParameter***

It saves the customer's setting into a temporary space. When the task is to be submitted, the content of that temporary space will be added into the ***xRSL*** script.

CHAPTER 5: IMPLEMENTATION

In this chapter we introduce our program detail after general design, includes Uppaal source code investigation, we will show what we change and add new in Uppaal source code to have a new system as user expected. But, due the time, we did not finish all the system requirements, then, we will concern about evaluation in the next chapter – Evaluation.

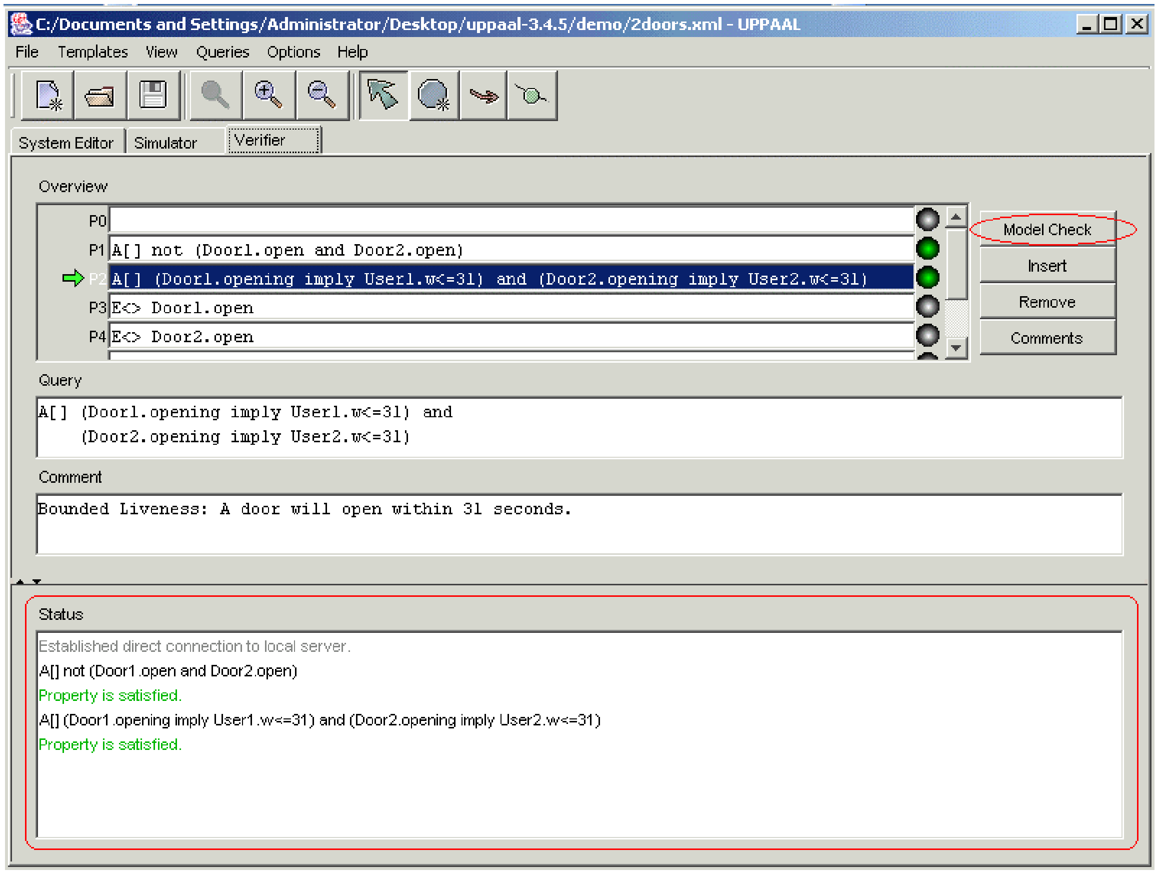
In design, we are going to program with the new version of D-Uppaal engine (***dvserver***). But, till this time, this D-Uppaal new version has not finished yet. Besides that, the input files of D-Uppaal will be the same with the input files of current Uppaal. Then in this Implementation, we use Uppaal engine (***verifyTA***) instead of D-Uppaal engine (***dvserver***).

To modify current Uppaal system, firstly, we need to understand the constructor and some important classes in Uppaal source code. So, the first part in this chapter, we will write about Uppaal source code.

5.1 Structure of Uppaal

The source codes of Uppaal GUI are stored in ***GUI*** folder and five subdirectories of ***GUI***. There is one important class that we emphasize on: ***System Inspector***.

The ***SystemInspector*** provides the graphic interface, which includes buttons and windows in original Uppaal. For example, it provides the button of ***Model Check*** and the status window to show the result of local verification. When the ***Model Check*** button is pressed, the ***SystemInspector*** will invoke the method in ***VerificationTask*** class to perform the actual verifying and show the result on status window.



```
P0
P1 A[] not (Door1.open and Door2.open)
P2 A[] (Door1.opening imply User1.w<=31) and (Door2.opening imply User2.w<=31)
P3 E<> Door1.open
P4 E<> Door2.open
```

```
A[] (Door1.opening imply User1.w<=31) and
    (Door2.opening imply User2.w<=31)
```

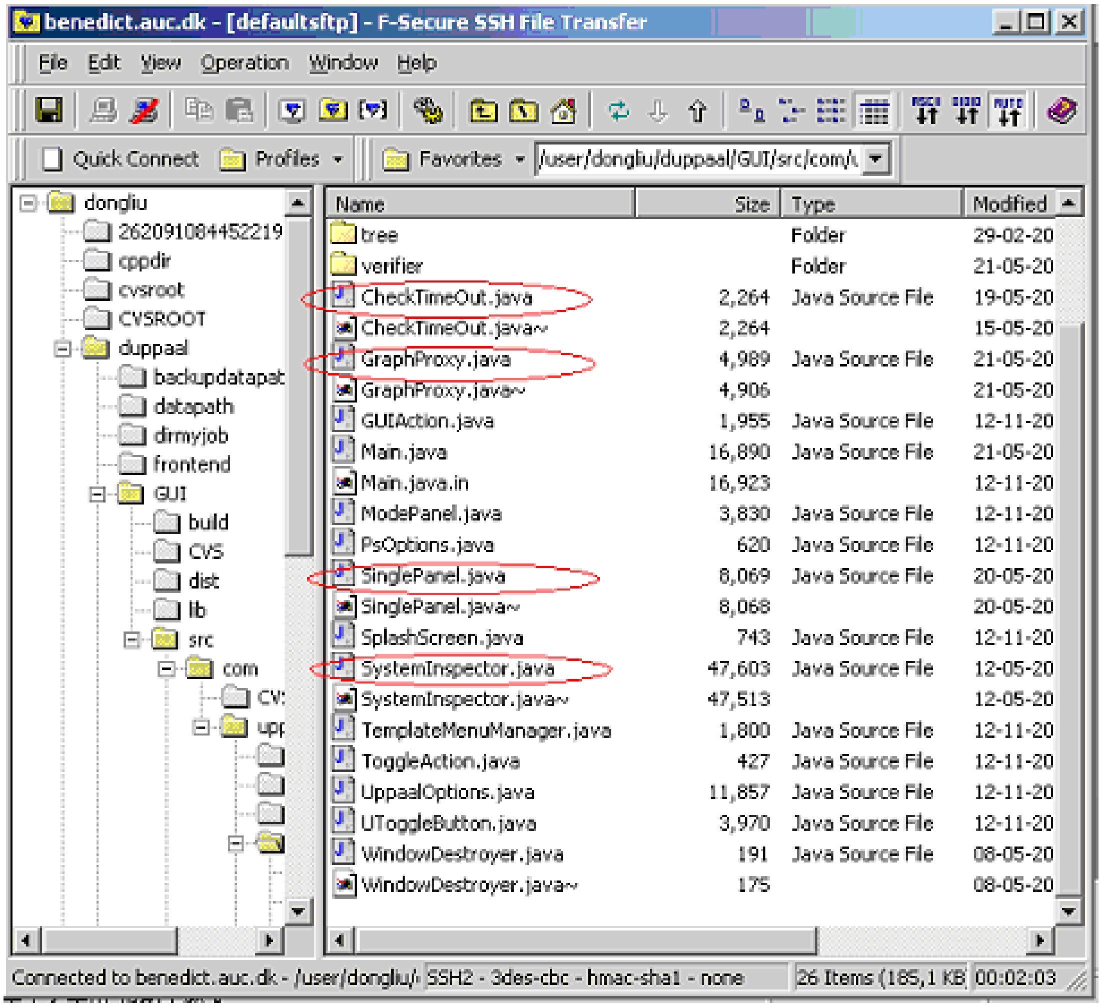
```
Bounded Liveness: A door will open within 31 seconds.
```

```
Established direct connection to local server.
A[] not (Door1.open and Door2.open)
Property is satisfied.
A[] (Door1.opening imply User1.w<=31) and (Door2.opening imply User2.w<=31)
Property is satisfied.
```

Figure IMP-01: The ***SystemInspector*** class starts the verification after users click the ***Model Check*** button.

Because the ***SystemInspector*** class is stored in ***GUI*** folder, the ***VerificationTask*** class is in ***GUI/verifier*** folder, we need to modify or add new programs in ***GUI*** and ***GUI/verifier*** directories. For example, we add three new classes in ***GUI*** fold, one in ***GUI/verifier***.

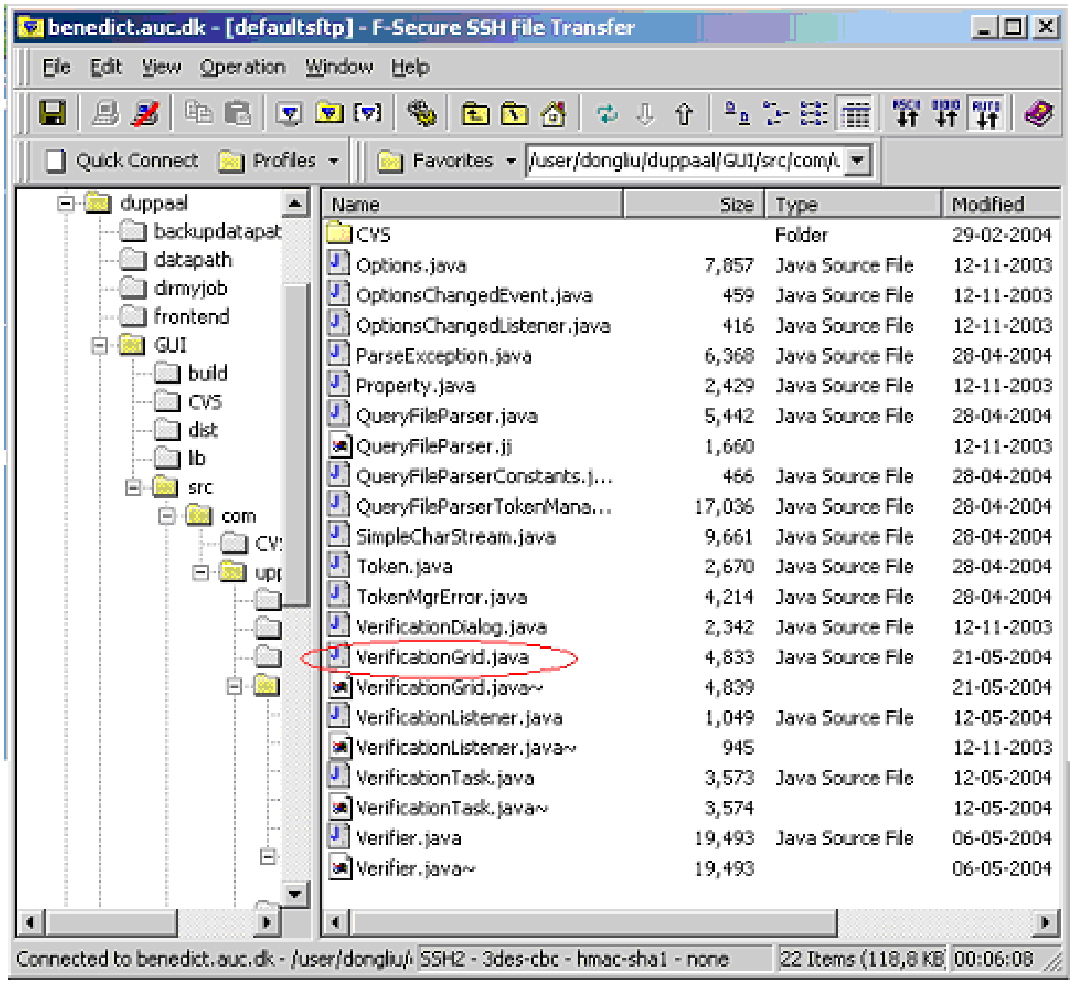
Follows display the new classes in ***GUI*** and ***GUI/verifier***.



Name	Size	Type	Modified
tree		Folder	29-02-20
verifier		Folder	21-05-20
CheckTimeOut.java	2,264	Java Source File	19-05-20
CheckTimeOut.java~	2,264		15-05-20
GraphProxy.java	4,989	Java Source File	21-05-20
GraphProxy.java~	4,906		21-05-20
GUIAction.java	1,955	Java Source File	12-11-20
Main.java	16,890	Java Source File	21-05-20
Main.java.in	16,923		12-11-20
ModePanel.java	3,830	Java Source File	12-11-20
PsOptions.java	620	Java Source File	12-11-20
SinglePanel.java	8,069	Java Source File	20-05-20
SinglePanel.java~	8,068		20-05-20
SplashScreen.java	743	Java Source File	12-11-20
SystemInspector.java	47,603	Java Source File	12-05-20
SystemInspector.java~	47,513		12-05-20
TemplateMenuManager.java	1,800	Java Source File	12-11-20
ToggleAction.java	427	Java Source File	12-11-20
UppaalOptions.java	11,857	Java Source File	12-11-20
UToggleButton.java	3,970	Java Source File	12-11-20
WindowDestroyer.java	191	Java Source File	08-05-20
WindowDestroyer.java~	175		08-05-20

Connected to benedict.auc.dk - /user/dongliu/ | SSH2 - 3des-cbc - hmac-sha1 - none | 26 Items (185,1 KB | 00:02:03

Figure IMP-02: One modified class (***SystemInspector***) and three new classes (***CheckTimeOut, GraphProxy, SinglePanel***) in folder of ***GUI***.



/user/dongliu/duppaal/GUI/src/com/u

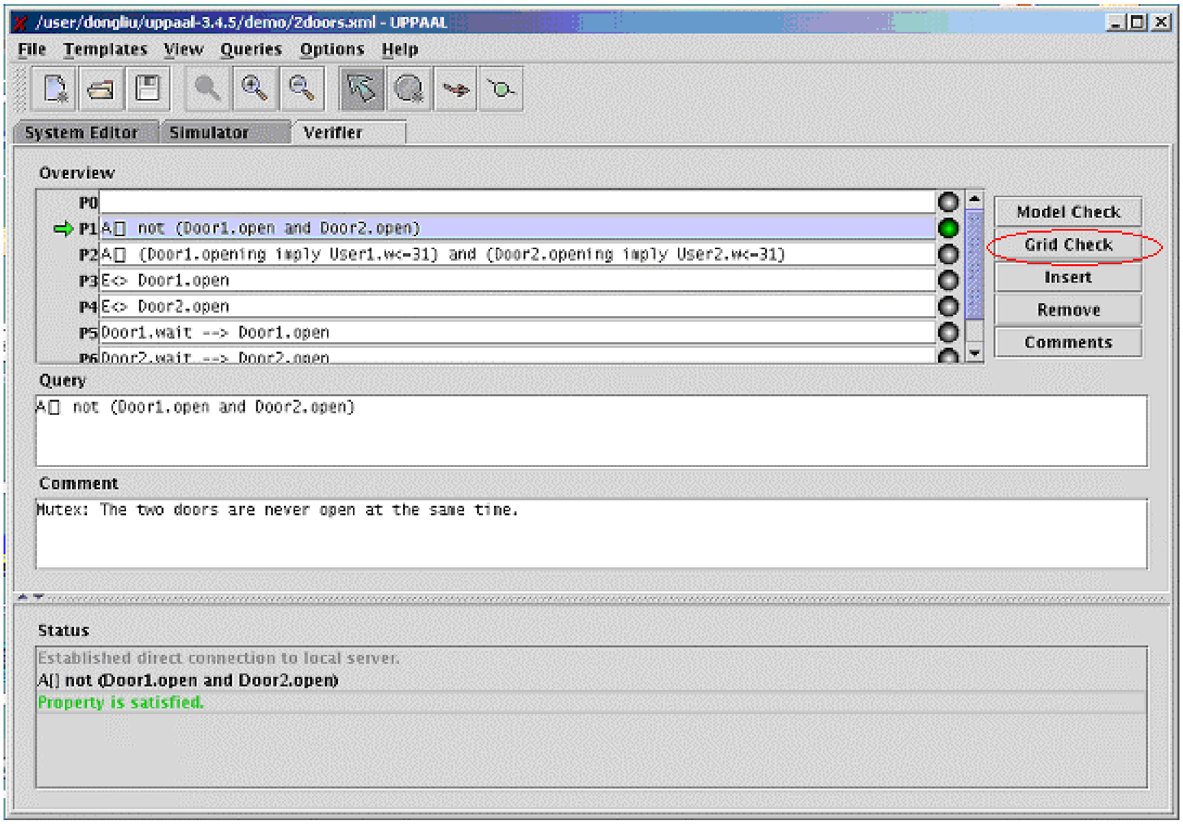
Name	Size	Type	Modified
CVS		Folder	29-02-2004
Options.java	7,857	Java Source File	12-11-2003
OptionsChangedEvent.java	459	Java Source File	12-11-2003
OptionsChangedListener.java	416	Java Source File	12-11-2003
ParseException.java	6,368	Java Source File	28-04-2004
Property.java	2,429	Java Source File	12-11-2003
QueryFileParser.java	5,442	Java Source File	28-04-2004
QueryFileParser.jj	1,660		12-11-2003
QueryFileParserConstants.j...	466	Java Source File	28-04-2004
QueryFileParserTokenMana...	17,036	Java Source File	28-04-2004
SimpleCharStream.java	9,661	Java Source File	28-04-2004
Token.java	2,670	Java Source File	28-04-2004
TokenMgrError.java	4,214	Java Source File	28-04-2004
VerificationDialog.java	2,342	Java Source File	12-11-2003
VerificationGrid.java	4,833	Java Source File	21-05-2004
VerificationGrid.java~	4,839		21-05-2004
VerificationListener.java	1,049	Java Source File	12-05-2004
VerificationListener.java~	945		12-11-2003
VerificationTask.java	3,573	Java Source File	12-05-2004
VerificationTask.java~	3,574		12-05-2004
Verifier.java	19,493	Java Source File	06-05-2004
Verifier.java~	19,493		06-05-2004

Connected to benedict.auc.dk - /user/dongliu/ SSH2 - 3des-cbc - hmac-sha1 - none 22 Items (118,8 KB 00:06:08

Figure IMP-03: In folder of ***GUI/verifier***, there is a new class named ***VerificationGrid***

5.2 The working flow of classes and invocation relationship

The ***SystemInspector*** provides the graphic interface, which includes buttons and windows in original Uppaal. The new version of it adds the "***Grid Check***" button. When users click the button, the property, which the users choose is stored as a temporary query file in the working directory, a model file (***.xml***) is also stored in the working directory. Here, the query file uses the same prefix as the model file.



```
P0
P1 A□ not (Door1.open and Door2.open)
P2 A□ (Door1.opening imply User1.w<=31) and (Door2.opening imply User2.w<=31)
P3 E<> Door1.open
P4 E<> Door2.open
P5 Door1.wait --> Door1.open
P6 Door2.wait --> Door2.open
```

Query: A□ not (Door1.open and Door2.open)

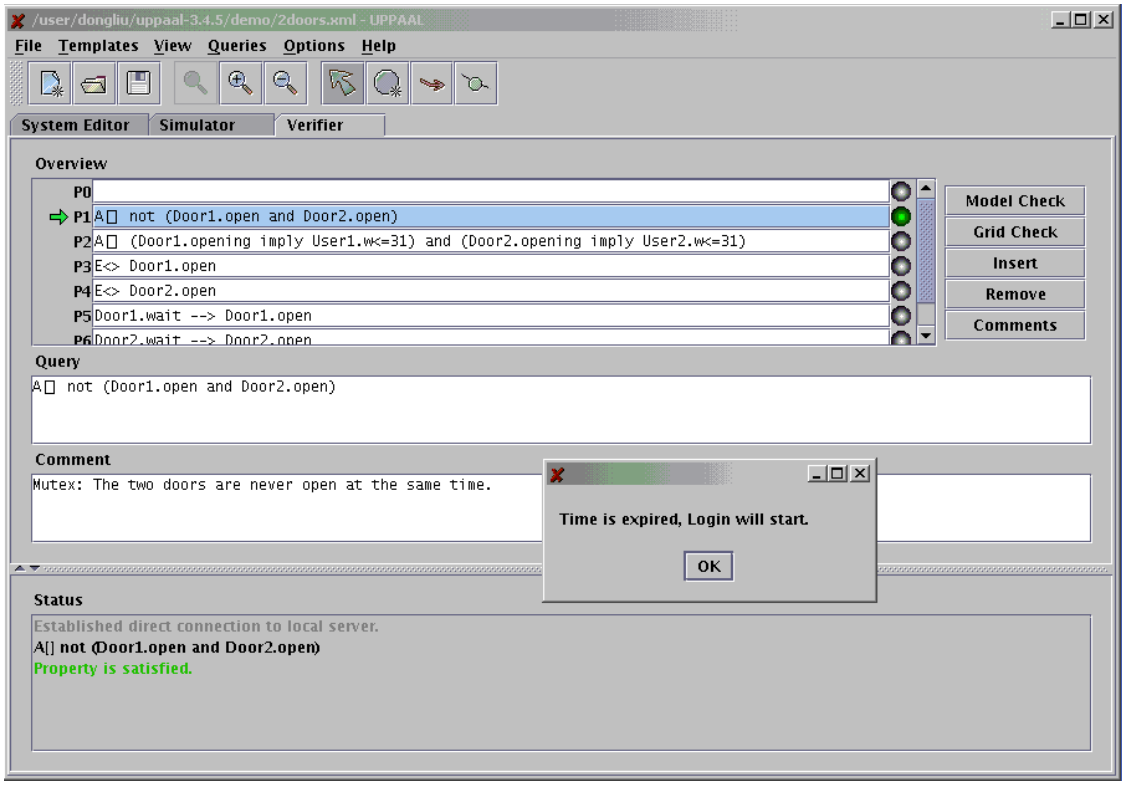
Comment: Mutex: The two doors are never open at the same time.

```
Established direct connection to local server.
A[] not (Door1.open and Door2.open)
Property is satisfied.
```

Figure IMP-04: "***Grid Check***" button in the Uppaal GUI

This button is defined in ***SystemInspector*** class

After that, the method in ***VerificationGrid*** class grasps the control stick. It receives the parameters, which contain the name of temporary query file and the name of model file. Then it invokes the constructor in ***CheckTimeOut*** class to do the job. After ***CheckTimeOut*** constructor writes the job result into a specific file, the method in ***VerificationGrid*** reads it and displays the content on the Uppaal graphic interface. Here is a problem. Because it takes several minutes to verify the job and get back the result from the grid, we have to wait for a long time to submit other jobs. To resolve this problem, the thread mechanism is used to guarantee the submit jobs parallel. The thread is used to guarantee the jobs can be submitted while other jobs are performing.



```
P0
P1 A□ not (Door1.open and Door2.open)
P2 A□ (Door1.opening imply User1.w<=31) and (Door2.opening imply User2.w<=31)
P3 E<> Door1.open
P4 E<> Door2.open
P5 Door1.wait --> Door1.open
P6 Door2.wait --> Door2.open
```

Query: A□ not (Door1.open and Door2.open)

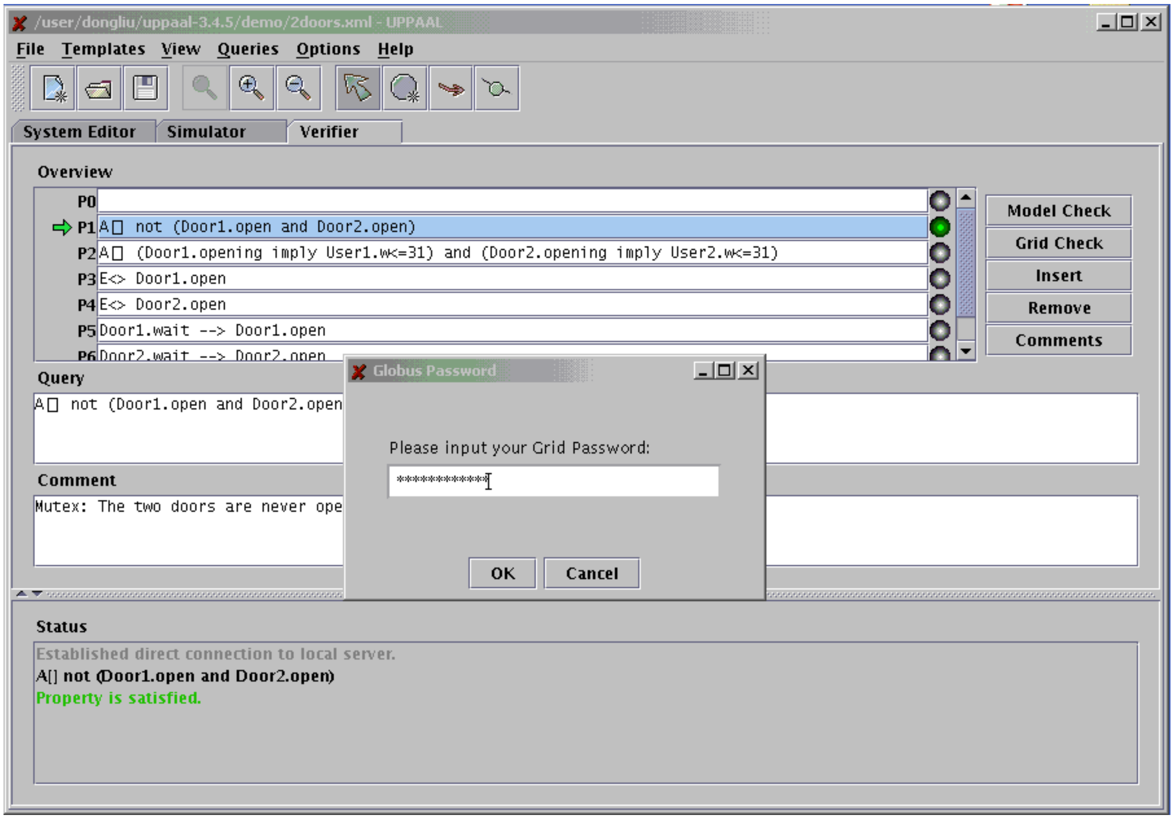
Comment: Mutex: The two doors are never open at the same time.

Time is expired, Login will start.

```
Established direct connection to local server.
A[] not (Door1.open and Door2.open)
Property is satisfied.
```

Figure IMP-05: Property “P1” is chosen. When the user submits it, ***CheckTimeOut*** finds his time is out

The ***CheckTimeOut*** checks whether a user can use the grid system more than two minutes. Such an operation is necessary because the grid system distributes every user a fixed period to use the grid after he logs on. Grid always checks users’ privilege while users submit the jobs. If the grid finds the user has no valid permission any longer, the user’s job will not be submitted to grid. Therefore, we design this mechanism in initial phase to help users avoid meeting this interruption from Nordugrid suddenly. If the result of time check is more than two minutes, the user’s job can be forward to ***SinglePanel*** class. In other words, Nordugrid processes the job directly. If it is not, user must provide his password to Nordugrid. The ***GraphProxy*** is invoked. After he passes the authentication check, he needs to submit his job again.



/user/dongliu/uppaal-3.4.5/demo/2doors.xml - UPPAAL

```
P0
P1 A□ not (Door1.open and Door2.open)
P2 A□ (Door1.opening imply User1.w<=31) and (Door2.opening imply User2.w<=31)
P3 E<> Door1.open
P4 E<> Door2.open
P5 Door1.wait --> Door1.open
P6 Door2.wait --> Door2.open
```

Query: A□ not (Door1.open and Door2.open

Comment: Mutex: The two doors are never ope

Globus Password

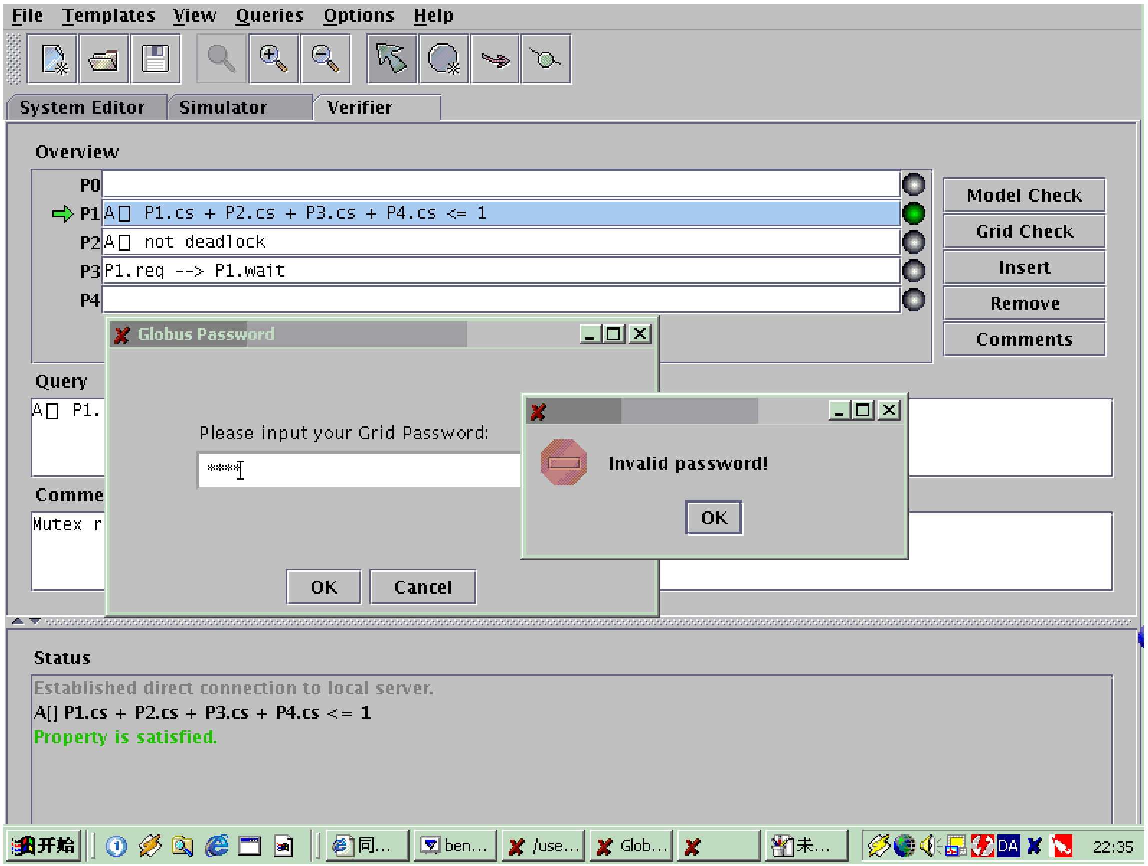
Please input your Grid Password: *************

Status:

```
Established direct connection to local server.
A[] not (Door1.open and Door2.open)
Property is satisfied.
```

Figure IMP-06: ***GraphProxy*** asks user’s grid password

GraphProxy opens a window and regards the user's input as password. In the beginning, we tried to run the command of "***echo*** *password* | ***grid-proxy-init -pwstdin***" (***grid-proxy-init -pwstdin*** is the command of check user password in Nordugrid) in Java Program like we ran it under unix shell. This command was successful under unix shell, but it failed in java implementation: The program always stopped at the first part of "***echo*** *password*". To resolve it, we change the design: we dispatch an ***Inputstream*** for the "***echo*** *password*" and an ***Outputstream*** for the "***grid-proxy-init***". The input string for "*echo password*" is stored in "***in***" ***Inputstream***, then all the content of "***in***" is written to "***out***". It means the "***grid-proxy-init***" deals with the password in back end. After that, the "***in***" ***Inputstream*** is dispatched for "***grid-proxy-init***" to get the response information of grid security check. This time, all of message projects onto a popup dialog window.



P0
P1 A□ P1.cs + P2.cs + P3.cs + P4.cs <= 1
P2 A□ not deadlock
P3 P1.req --> P1.wait
P4

A□ P1.

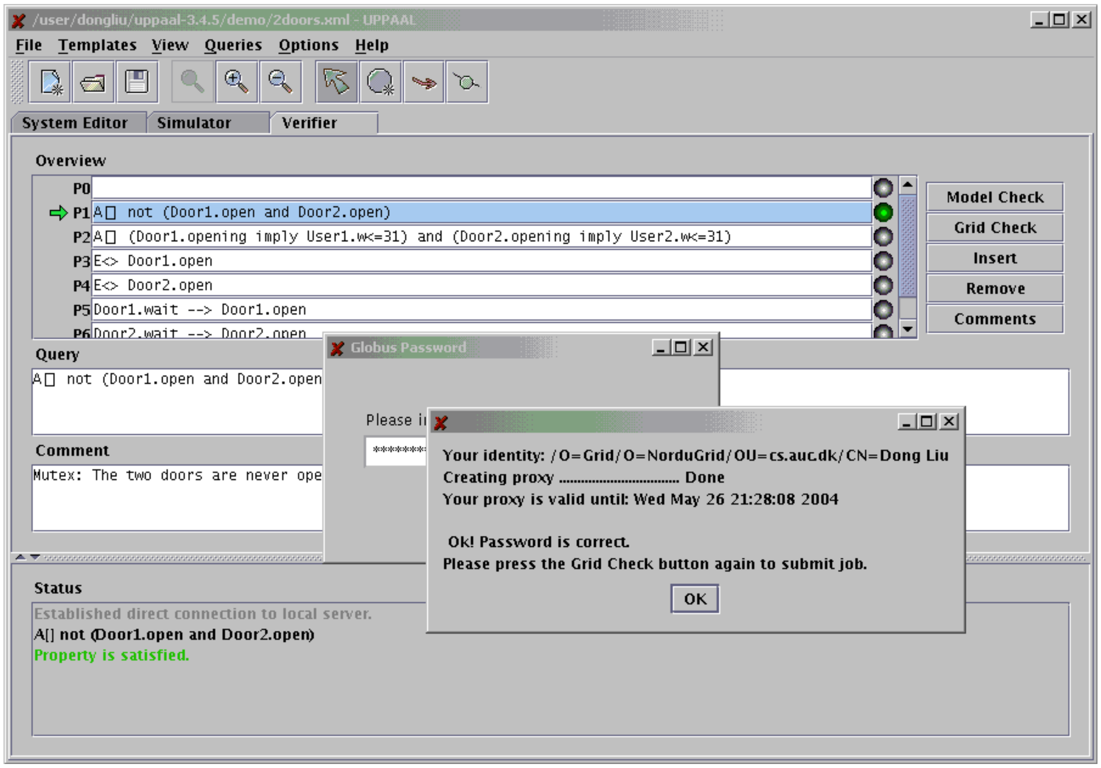
Mutex r

Please input your Grid Password:
****

Invalid password!

```
Established direct connection to local server.
A[] P1.cs + P2.cs + P3.cs + P4.cs <= 1
Property is satisfied.
```

Figure IMP-07: The user doesn't pass the authentication check



```
P0
P1 A□ not (Door1.open and Door2.open)
P2 A□ (Door1.opening imply User1.w<=31) and (Door2.opening imply User2.w<=31)
P3 E<> Door1.open
P4 E<> Door2.open
P5 Door1.wait --> Door1.open
P6 Door2.wait --> Door2.open
```

Query: A□ not (Door1.open and Door2.open

Comment: Mutex: The two doors are never ope

Please i

****

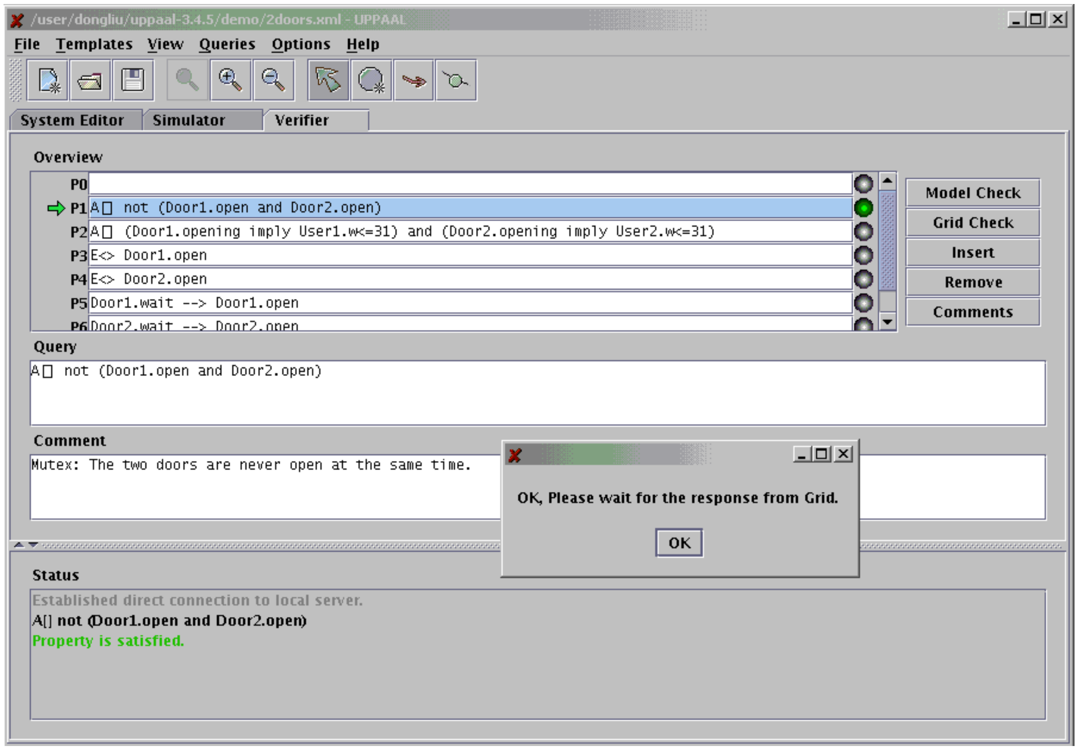
Your identity: /O=Grid/O=NorduGrid/OU=cs.auc.dk/CN=Dong Liu
Creating proxy .... Done
Your proxy is valid until: Wed May 26 21:28:08 2004

Ok! Password is correct.
Please press the Grid Check button again to submit job.

Established direct connection to local server.
A[] not (Door1.open and Door2.open)
Property is satisfied.

Figure IMP-08: The user passes the authentication check

SinglePanel receives the parameters (names of the model file and the query file), which are delivered by ***CheckTimeOut*** at first. According to the file names, it creates the ***.xrsl*** script in the working space. The ***.xrsl*** file has the same prefix as the query file and the model file. It specifies the model file and query file which are used by ***dvserver*** in Nordugrid. It also includes the file name of job result no matter how the result is satisfied. Then ***SinglePanel*** submits the job onto Nordugrid. Besides, this class opens an invisible window as a pool to hold the returned message from Nordugrid. Every time the content updates, the ***SinglePanel*** checks the data to look for the desired string. If it found, relevant operation will start. For example, after the job is submitted, ***SinglePanel*** periodically checks the Nordugrid status information in the pool to search the string of "***IS FINISHED***". If it appears, it means Nordugrid finishes the job. ***SinglePanel*** can perform latter tasks. Finally, the ***SinglePanel*** downloads the job result from Nordugrid and store it in a specific directory.



/user/dongliu/uppaal-3.4.5/demo/2doors.xml - UPPAAL

Overview

```
P0
P1 A[] not (Door1.open and Door2.open)
P2 A[] (Door1.opening imply User1.w<=31) and (Door2.opening imply User2.w<=31)
P3 E<> Door1.open
P4 E<> Door2.open
P5 Door1.wait --> Door1.open
P6 Door2.wait --> Door2.open
```

Query

```
A[] not (Door1.open and Door2.open)
```

Comment

```
Mutex: The two doors are never open at the same time.
```

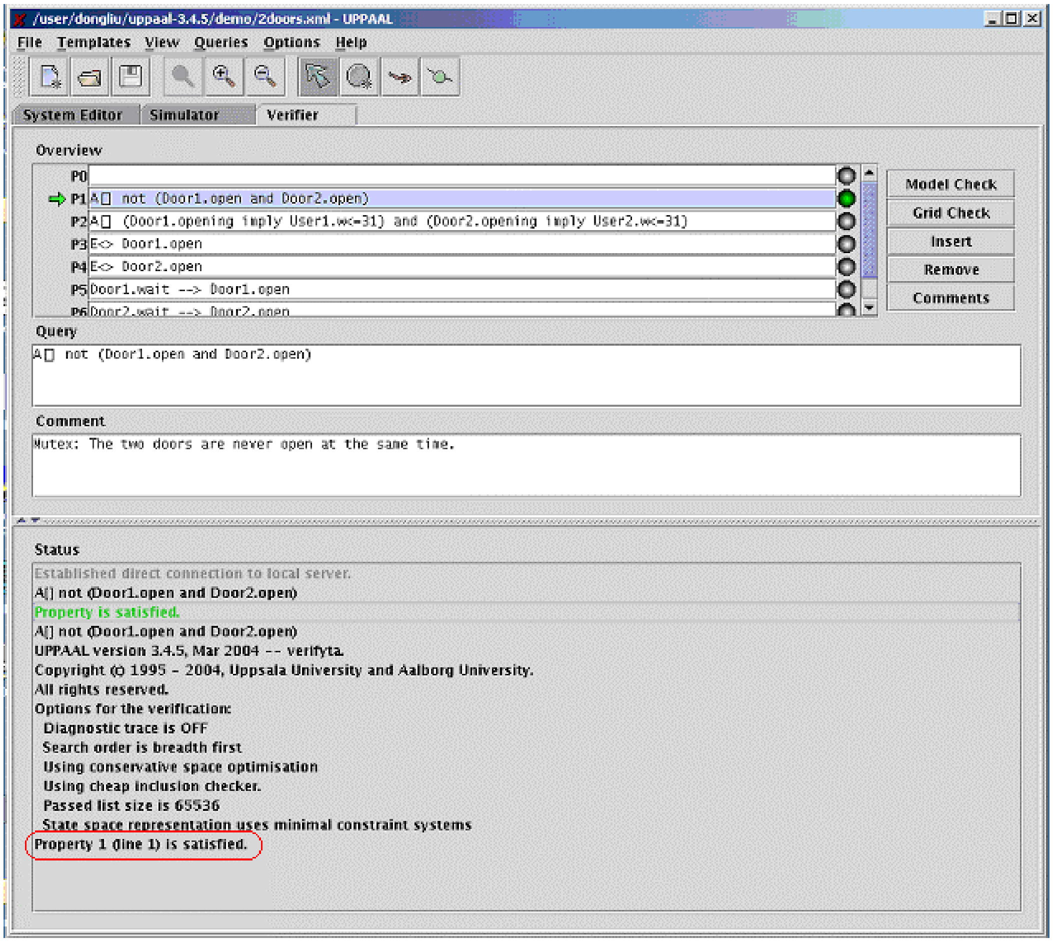
OK, Please wait for the response from Grid.

Status

```
Established direct connection to local server.
A[] not (Door1.open and Door2.open)
Property is satisfied.
```

Figure IMP-09: The SinglePanel begins to process the job.

The result displays in the status window of Uppaal GUI: If the result is positive, the last line of status information specifies the property ***is satisfied***. Else it specifies the property ***is not satisfied***.



```
P0
P1 A[] not (Door1.open and Door2.open)
P2 A[] (Door1.opening imply User1.w<=31) and (Door2.opening imply User2.w<=31)
P3 E<> Door1.open
P4 E<> Door2.open
P5 Door1.wait --> Door1.open
P6 Door2.wait --> Door2.open
```

Query:

```
A[] not (Door1.open and Door2.open)
```

Comment:

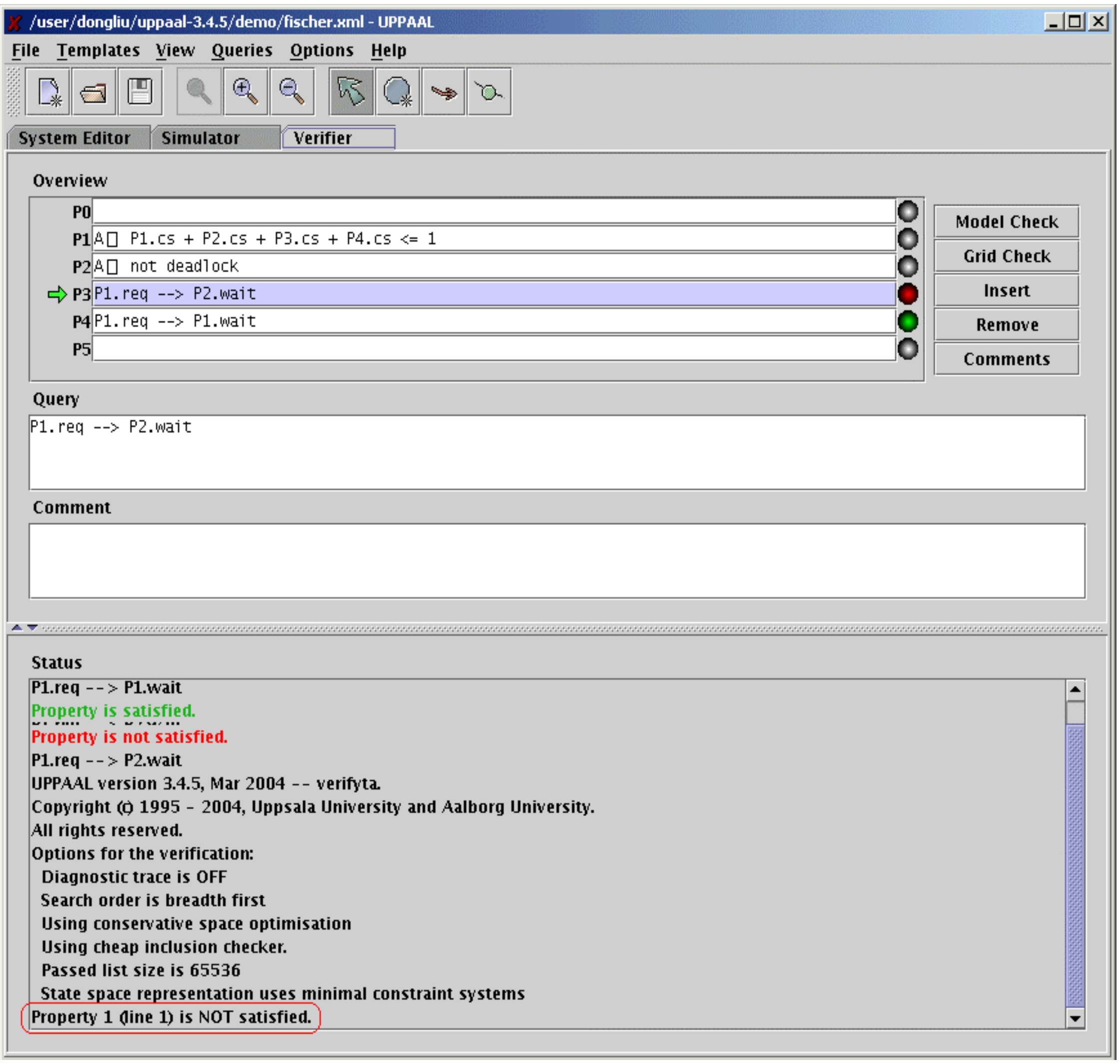
```
Mutex: The two doors are never open at the same time.
```

Status:

```
Established direct connection to local server.
A[] not (Door1.open and Door2.open)
Property is satisfied.
A[] not (Door1.open and Door2.open)
UPPAAL version 3.4.5, Mar 2004 -- verifyta.
Copyright (c) 1995 - 2004, Uppsala University and Aalborg University.
All rights reserved.
Options for the verification:
  Diagnostic trace is OFF
  Search order is breadth first
  Using conservative space optimisation
  Using cheap inclusion checker.
  Passed list size is 65536
  State space representation uses minimal constraint systems
Property 1 (line 1) is satisfied.
```

Figure IMP-10: The result displays on the status window.

The last line means the result is positive.



```
P0
P1 A□ P1.cs + P2.cs + P3.cs + P4.cs <= 1
P2 A□ not deadlock
P3 P1.req --> P2.wait
P4 P1.req --> P1.wait
P5
```

Query: P1.req --> P2.wait

```
P1.req --> P1.wait
Property is satisfied.
Property is not satisfied.
P1.req --> P2.wait
UPPAAL version 3.4.5, Mar 2004 -- verifyta.
Copyright (c) 1995 - 2004, Uppsala University and Aalborg University.
All rights reserved.
Options for the verification:
  Diagnostic trace is OFF
  Search order is breadth first
  Using conservative space optimisation
  Using cheap inclusion checker.
  Passed list size is 65536
  State space representation uses minimal constraint systems
Property 1 (line 1) is NOT satisfied.
```

Figure IMP-11: The result displays on the status window.

The last line means the result is negative.

5.3 Detailed description of main classes

5.3.1 SystemInspector

As we known, it provides the "***Grid Check***" button on the panel. When the user chooses property to verify on grid, it encapsulates the property as the query file and delivers it with model file to ***VerificationGrid***.

To acquire this function, we add three new methods in existed ***SystemInspector*** class. They are ***gridCheckAction***, ***gridReceived*** and ***setGridCheckActionEnable***.

The ***gridCheckAction*** is an instance of ***GUIAction*** class, which has been defined in original Uppaal. It defines the "***Grid Check***" button. When the user clicks this button, a boolean variable ***gridFlag*** is assigned a true value. This variable will be used later to prevent the control flow entering the "***Model Check***" which means verifying system in local machine. Besides, the property he chooses in "overview" window is selected and stored as query file in the working directory. Below is the code of ***gridCheckAction***.

```
----------------------------------------------------------------------------------------------------
     gridCheckAction = new GUIAction( "Grid Check" ) {
                 public void actionPerformed(ActionEvent e) {
                             gridFlag = true;
                             verify(verifier.getQueries());
                 }
              };
----------------------------------------------------------------------------------------------------
```

The function of ***gridReceived(vector)*** method is to read the content of vector and show them on the status window of Uppaal GUI.

setGridCheckActionEnable(boolean) makes sure the "***Grid Check***" button is not available temporarily when the user's job is submitting.

Following code checks whether ***gridFlag*** is true. If so, the method in ***VerificationGrid*** is invoked to deal with the user's job. The variable '***props'*** is the property, which is chosen by the user. It is stored in working directory as a query file. The ***jobName*** is the model file name. After the invocation of ***VerificationGrid***, the ***gridFlag*** is set to false since the user may do some local verifying job in next turn. If the ***gridFlag*** is false, as former mentioned, the method in original ***VerificationTask*** is invoked to verify the job in local machine.

```
----------------------------------------------------------------------------------------------------
if (gridFlag == true) {
    String jobName =
    loadFile.getName().substring(0,loadFile.getName().lastIndexOf(".xml"));
    theGdtask = new VerificationGrid(props,1000,jobName, theListener);
    theGdtask.run();
    gridFlag = false;
    }
else {
    // Start the background task
    theTask = new VerificationTask((UppaalSystem)system.dereference(), engine, props,
1000, theListener);
     theTask.run();
    };
----------------------------------------------------------------------------------------------------
```

5.3.2 VerificationGrid

This class locates in ***GUI/verifier*** fold. It receives the property and model filename from the upper class ***SystemInspector*** then it invokes ***CheckTimeOut*** to submit the job, after the job result is downloaded from grid, display it.

In the following definition of ***VerificationGrid*** constructor, we see it deals with the property ***props*** and the model filename ***jobName***.

```
----------------------------------------------------------------------------------------------------
    public VerificationGrid(Vector props, int delay, String jobName,
                                    VerificationListener listener)
      {
            this.props = props;
            this.delay = delay;
            this.jobName = jobName;
            ……
      }
----------------------------------------------------------------------------------------------------
```

The ***run*** method uses thread mechanism to control multiple jobs can run at the same time. The loop method deals with the actual job in the thread. So after the thread starts, the user doesn't need to worry about the processing job blocks following tasks.

```
----------------------------------------------------------------------------------------------------
      public void run() {
            if (thread == null) {
              thread = new Thread() {
                        public void run() {
                          listener.gridReceived(loop());
                        }
              };
              thread.start();
```

```
                }
        }
------------------------------------------------------------------------------------------------------------
```

The ***loop*** method finishes three tasks in turn. Firstly, it stores the content of ***props*** in the working directory with the same previous name with model filename (see the clause with black bold). Secondly, loop calls the method in ***CheckTimeOut*** class with ***jobName*** as the parameter. Finally, the loop calls ***readFile*** method to put the content of downloaded job result into buffer.

```
------------------------------------------------------------------------------------------------------------
    private Vector loop() {
                String downldir ="";
                String downldfile ="";
                Vector vof = new Vector();
                File fq =  new File(jobName +".q");
                try {
                PrintWriter pw = new PrintWriter (new FileOutputStream (fq));
                pw.print((String)props.lastElement());
                pw.close();
                } catch (IOException e1) {
                System.out.println("Problem in creating query file.");
                }
                ……
                CheckTimeOut ct = new CheckTimeOut(jobName);
                downldir = System.getProperty("DOWNLDIR");
                downldfile = System.getProperty("DOWNLDFILE");
                ……
                try {
                readFile(downldfile, vof);
                } catch  (IOException e1) {
```

```
            System.out.println("Problems in read "+ downldfile);
            }
            ……
    }
-----------------------------------------------------------------------------------------------------
```

The main idea of ***readFile*** method is to copy the content from a file (job result) to a vector. Considering it is better to show the original property with its verification result, we design it to add the property into the vector before read the job result file.

```
-----------------------------------------------------------------------------------------------------
    public Vector readFile(String fileName,Vector vof) throws IOException {
            BufferedReader istream = new BufferedReader(new
            InputStreamReader(new FileInputStream(fileName)));
            String text = "";
            vof.addElement(props.lastElement());
            while((text = istream.readLine()) != null) {
                    vof.addElement(text);
            }
            return vof;
      }
-----------------------------------------------------------------------------------------------------
```

5.3.3 CheckTimeOut

It invokes the "***grid-proxy-info -timeleft***" system command to get the information of user's valid time. The output is saved in the ***timeValue*** variable. If the ***timeValue*** is less than 120 seconds, the warning message of time out appears and ***GraphProxy*** will check user's password. Otherwise, user's job will be submitted directly by constructor of ***SinglePanel*** class.

```
--------------------------------------------------------------------------------------------------
    ……
    Process ctprc = rt.exec("grid-proxy-info -timeleft");
    DataInputStream in = new DataInputStream(ctprc.getInputStream());
    try {
            while ((timestring = in.readLine()) != null) {
            timeValue = Integer.valueOf(timestring).intValue();
            }
            } catch (IOException e) {
                        System.exit(0);
            }

    if (timeValue < 120 ) {
    JFrame jframe = new JFrame();
    JOptionPane.showMessageDialog(jframe,"Time is expired, Login will
    start.","",JOptionPane.PLAIN_MESSAGE);
    GraphProxy gp = new GraphProxy();
    gp.setVisible(true);
            }
            else {
    SinglePanel sp = new SinglePanel(jname);
            }
            ……
--------------------------------------------------------------------------------------------------
```

5.3.4 GraphProxy

It pops up a window, reads and checks the input password, masks it with star characters. The process ***wxpr*** performs the ***echo password*** command and variable ***in*** accepts the actual input of user’s password. When the process ***tpr*** performs the ***grid-proxy-init –pwstdin*** command, the content of ***in*** is delivered to ***out*** which is the output stream of ***tpr***. Actually, the output stream of ***tpr*** is the input password, which is fed to ***grid-proxy-init*** command. The output of grid password check is put into ***in*** again. The ***msgOut*** achieves the content of in to be showed on a dialog window actuated by ***GraphProxy***.

```
-------------------------------------------------------------------------------------------------------
    Process wxpr= rt.exec("echo "+passwd);
            in = wxpr.getInputStream();

    Process tpr = rt.exec("grid-proxy-init -pwstdin");
            OutputStream out =tpr.getOutputStream();
            int b;
            while((b = in.read()) != -1 ) {
                        out.write(b);
            }
            wxpr.waitFor();
            in.close();
            out.close();
            in = tpr.getInputStream();
            String guiOut = null;
            String msgOut = "";
            BufferedReader msgBr = new BufferedReader(new
            InputStreamReader(in));
            while((guiOut = msgBr.readLine()) != null) {
                        msgOut = msgOut + guiOut +"\n";
                        }
-------------------------------------------------------------------------------------------------------
```

5.3.5 SinglePanel

It invokes an invisible window, creates the ***xRSL***, submits the job and gets the status data from the grid. It also checks the data to find out the desired string, downloads the job result from Nordugrid.

The main methods in ***SinglePanel*** class are ***createXrsl(jobname)***, ***runAuto()*** and ***systemCall(command)***. After ***createXrsl*** method returns the true value, the ***runAuto*** method starts. It calls several ***systemCall*** during the job procedure.

```
----------------------------------------------------------------------------------------------------
    if (createXrsl(jobname)) {
                xrslName = jobname;
                runAuto();
                }
                ……
                }
----------------------------------------------------------------------------------------------------
```

The ***createXrsl*** creates the ***xrsl*** script according to the given jobname. It writes the necessary parameters such as executable filename, input files name and standard output file name into the ***xrsl*** script. If the file operation finishes successfully, the method returns true, else it returns false. The ***SinglePanel*** depends on this Boolean value to decide whether submits the job to Nordugrid because the correct ***xrsl*** script is a necessary component of Nordugrid job.

```
----------------------------------------------------------------------------------------------------
    public boolean createXrsl (String objectFilename) throws IOException {
                File fQ = new File(objectFilename +".q");
                boolean jobflag = false;
                if(fQ.exists()) {
                File f = new File(objectFilename+".xrsl");
                PrintWriter pw = new PrintWriter (new FileOutputStream (f));
                pw.print ("&\n");
                pw.print ("(executable=\"bin-Linux/verifyta\")\n");
                pw.print ("(arguments= "+objectFilename+".xml
                "+objectFilename+".q)\n");
                pw.print ("(inputFiles= ("+ objectFilename +".xml"+"
                \"/user/dongliu/uppaal-3.4.5/demo/"+objectFilename +".xml"+"\") ("+
                objectFilename +".q"+" \"\"))\n");
                pw.print ("(stdout= "+objectFilename+".dat)\n");
                pw.print ("(join=\"yes\")\n");
                pw.close ();
                JFrame jframe = new JFrame();
                JOptionPane.showMessageDialog(jframe,"OK, Please wait for the
                response from Grid. \n","",JOptionPane.PLAIN_MESSAGE);
                jobflag = true;
                            }
                else {
                JFrame jframe = new JFrame();
                JOptionPane.showMessageDialog(jframe,"Insufficient files. Can't create
                the xRSL file. \n","",JOptionPane.ERROR_MESSAGE);
                jobflag = false;
                            }
                return jobflag;
                }
------------------------------------------------------------------------------------------------
```

The ***autoRun*** method is the key of ***SinglePanel*** class. It performs submitting job, periodically checks job status until the finished flag appears. It also downloads the job result at the end.

When the job is submitted, the grid status message is stored in an invisible pool named ***msgArea***. ***autoRun*** regards the "***IS FINISHED***" string as the signal of normal termination. It checks the msgArea every ten seconds. To reduce the unnecessary workload, ***autoRun*** always remembers the accurate identify number of current job process named ***jobid**,* which is assigned by Nordugrid. When the ***autoRun*** needs to download the job, it only downloads the job exactly. So the latter process can simply finds and opens the download file without worry about making a wrong choice. The ***downldir*** and ***downldfile*** specify the position of download file. The ***readFile*** method in ***VerificationGrid*** class will use these two values when it opens the download file and reads the content.

```
public void runAuto() {

String testString;
systemCall("ngsub -x" +" -f "+ xrslName + ".xrsl");
testString = msgArea.getText();
if (search(testString,JOB_SUBMITTED_LABEL) == true) {
jobid = testString.substring(testString.lastIndexOf(JOB_SUBMITTED_LABEL));
jobid = jobid.replaceFirst(JOB_SUBMITTED_LABEL,"");
jobid = jobid.trim();
}
systemCall("ngstat " + jobid);
testString = msgArea.getText();
if (search(testString,"ngstat: No jobs") == false) {
while (search(testString, JOB_FINISHED_LABEL) == false ) {
testString = msgArea.getText();
systemCall("ngstat " + jobid);
try {   // To delay ten seconds.
        Thread.currentThread().sleep(10000);
}  catch (InterruptedException e1) {};
}
        systemCall("ngcat " + jobid);
        systemCall("ngget " + jobid);
```

```
            testString = msgArea.getText();
    if (search(testString,NGGET_FINISHED_LABEL) == true) {
            localdir = System.getProperty("user.dir");
            downldir = testString.substring(testString.lastIndexOf(localdir));
            downldir = downldir.replaceFirst(localdir + "/","");
            downldir = downldir.replaceFirst("\n" + UNUSEFUL_STRING + "\n","");
            downldir = downldir.trim();
            }
    }
    }
            else {
    msgArea.setText("");
                    }
    }
----------------------------------------------------------------------------------------------------
```

CHAPTER 6: EVALUATION

In this chapter, we first sum up the major results of our work and then we evaluate the implementation of our new system, which we have programmed. Finally, we use the experience from this and prior projects to make suggestion for the future works.

6.1 Our works

The works we have done in this project can roughly divided into three main tasks:

- Analyzed problems and users requirements.
- Designed a system that satisfies users.
- Programmed.

In first task, from the experience in prior project last semester, we analyzed problems when running verify large systems using D-Uppaal engine and NorduGrid resource. This analysis is based on end-users expectation.

From these user requirements, we start designing to modify current system to have a new system. This design gave us general view what we need to do, and why we choose that way. Beside that, we think about some problems will be happen when using new system to prove the efficient of it.

Due the time, we could not program all as we designed. In Implementation, we only wrote about what we did and evaluate it.

The next part, we will evaluate our new system and implementation, future works that can make system will be nicer with users.

6.2 Implementation evaluation

Until now, our system can allow s users to submit job on NorduGrid, usingD-Uppaal engine (***dvserver***) to verify large systems. Besides that, it ensures about security and users can do many tasks in the same time.

However, we did not have enough time to develop a smart system, which can be more efficient in job processing as allowing users choose number of CPUs or other parameters in ***xRSL*** file.

CHAPTER 7: CONCLUSION

We did not finish implementing a new system as design, but current new system is an evidence to conclude that integrating application with Grid system will bring more comfort for the users.

Besides that, in this experiment, we implement with specific application and NorduGrid system. Alternative, NorduGrid is only one specific system of Grid systems. But, through the result of this project, we can program all the function in design and find out how to make integration easier. That can be a future work for anyone who interested on this approach.

Through this project, we understand more about Grid advantage in extending resource and solve big problems with parallel processing. Especially, we know how to integrate this advantage to application, make it more efficient and comfort. We hope in the future work, we can use our knowledge that we collect from these projects to our new projects, when we use another applications that require lot money for extending resource.

Chapter 8: Figures-List

Figure IMP-04: “Grid Check” button in the Uppaal GUI. This button is defined in SystemInspector class

Figure IMP-05: Property “P1” is chosen. When the user submits it, CheckTimeOut finds his time is out

Figure IMP-06: GraphProxy asks user’s grid password

Figure IMP-07: The user doesn’t pass the authentication check

Figure IMP-08: The user passes the authentication check

Figure IMP-09: The SinglePanel begins to process the job.

Figure IMP-10: The result displays on the status window. The last line means the result is positive.

Figure IMP-11: The result displays on the status window. The last line means the result is negative.

CHAPTER 9: REFERENCES

[01] Huong H.T.T, Dong Liu, - *Can Grid help us to verify large systems* – Allborg university, Denmark (2003)

[02] I. Foster, C. Kesselman, eds. - *The Grid: Blue Print for a New Computing Infrastructure* - Morgan Kaufmann, San Francisco, Calif. (1999).

[03] Alexandre David, Tobias Amnell – *Uppaal 2k: Small tutorial* -2002