Web Experiment Manager: On the Distinctive Features Useful in an Application for Experiment Management

Matías Alberto Gavinowich

gmatias@gmail.com

Facultad de Ingeniería, Universidad de Buenos Aires, Av. Paseo Colón 850, (1063) Ciudad de Buenos Aires, Argentina

Abstract. This work presents an application which allows to manage experiments in an environment such as a cluster or a grid. The application allows scientists to define, store, monitor, control, query, and get the results of their experiments from a friendly interface. By using the application, scientists are spared the need to deal with computing technical issues, which very often they find themselves taking care of.
The focus of this work lies in identifying those aspects that can be improved in existing tools, compiling them into a well-defined list of distinctive features, and addressing them. Some of these distinctive features are: support for parameter-sweep applications, being experiment-oriented and considering the need of scientists to share information. The application includes innovative features which could prompt future lines of work.

1 Introduction

Conducting experiments is a complex undertaking, both because of the complexity of the studies themselves and due to the challenge of managing all the information involved. Oftentimes, experiments are so elaborate that they require a high amount of processing power, which is where clusters, grids, and high throughput computing in general come into the picture. Experiments can share aspects of their design, and within the same experiment analyzing several variations may be needed.

The software industry does not usually focus on producing specific tools for scientists needing to manage their experiments, let alone making such software freely available. The majority of the software products that address the needs of the scientific community are developed in-house by the very same members of the community. Many times, scientists are forced to deal with computing technical details – such as the operating system commands needed to access a cluster or a grid – because they lack a tool that hides that complexity.

This article presents an Experiment Manager that allows scientists and researchers to define, store, monitor, control, query, and get the results of experiments in a high throughput computing environment. The focus of this work

lies in identifying those aspects that are not addressed – or not addressed extensively – by existing tools, compiling them into a well-defined list of hereafter called 'distinctive features', and highlighting conceptual, functional, and technical concerns that need to be taken into account in order to provide those features. Furthermore, the application includes an innovative experimental feature that allows to plug additional actions at certain extension points – called hooks — in the application, allowing to automatically act on the results, even creating a feedback loop by which actions on the Experiment Manager itself can be triggered.

The work presented here builds on many ideas proposed in [1], which presents an experiment manager designed to be used mainly in a standalone environment, and focused on the CoG Kit [2] – a high level API for accessing Grid services – with an emphasis in the Globus Toolkit. Contrary to that proposal, the Experiment Manager presented here includes a web interface and focuses on Condor [3]. Focusing on Condor does not prevent using the software in a large scale scenario, thanks to features such as flocking and Condor-G, at the same time that it allows the Experiment Manager to better suit smaller scale environments such as a research laboratory at a given institution. The effort needed to set up a Condor based infrastructure is also usually considerably smaller than what is involved in setting up a Globus based one.

The Experiment Manager discussed here focuses on a specific type of experiments, known as parameter-sweep. Parameter-sweep studies involve running a given application several times while varying some input parameter. The relevance of such studies is highlighted in existing work such as [4] and [5]. A use case, presented in [1], could consist in producing several variations of a given genetic sequence and then looking up each of them in a huge database in order to identify the sequence, its family, or its similarity with other sequences. Parameter-sweep experiments are especially demanding as to their management, since they involve several inputs and several outputs, which must be kept track of. They appear in a number of disciplines which include but are not restricted to biology, physics, and even computing.

As mentioned before, the Experiment Manager includes a web interface – based on components known as portlets [6] – which can be integrated to existing portals at research institutions; the tool is consequently named Web Experiment Manager. Many institutions use web portals to provide researchers access to several applications from a consistent interface [7]. From the web interface, scientists can define experiments, which can include variations that will each translate to a job on the underlying high throughput computing system. This underlying system will from now on be also referred to as native system or remote system.

The architecture and design of the Web Experiment Manager was crafted with flexibility, maintainability and extensibility in mind. The functionality included aims to address needs which appear not to be all solved – at least completely – by just one existing tool, especially a freely available portlet based one. In brief, it provides scientists with a tool that allows them to manage the life cycle of (especially parameter-sweep) experiments from a friendly interface and

without needing to deal with computing technical issues, in a high throughput computing environment.

The following sections present some concepts used with the Experiment Manager, the architecture of the application, its scope, the distinctive features which were considered and how the decisions made allowed to implement them, sample usage, and some closing remarks.

2 Concepts

The Experiment Manager includes concepts which are used throughout the application in communicating with the user, as well as in the code the application is comprised of. While this concepts are mostly self-explanatory, since they can be intuitively matched to their counterparts in reality, they are briefly described in this section.

Experiment Type An entity that encapsulates a reusable experiment design. It is associated with an application in the native system which supports that specific experiment type. For instance, an experiment type could be the study of genetic sequences, supported by an application which performs such a task.

Experiment The main entity of the application, it represents an experiment.

Experiment Case An entity which represents a variation of an experiment. In a parameter-sweep study, each run of the application that supports the corresponding experiment type, i.e. each parameter tested, is represented by an experiment case. For instance, each of the variations of a genetic sequence to be looked up in a database constitute an experiment case. This concept is further refined into the idea of single experiment cases and batches of experiment cases (comprising several single cases).

Experiment Case Status The executing status of an experiment case (for instance, Idle, Running, or Completed).

Experiment Status The overall executing status of an experiment, defined as the summarized executing status of its comprised cases (for instance, Running: 2 Idle: 1).

Experiment Case Result The result of an experiment case, consisting of all the output files generated by the associated job in the native system.

Experiment Result The result of an experiment, consisting of the results of all the experiment cases that have produced results at the moment at which the experiment result is requested.

3 Architecture

The application is structured as an n-tier project, which helps in achieving a decoupled architecture and in making the application flexible, maintainable and extensible. In addition to that, each layer exposes interfaces to its client layers –

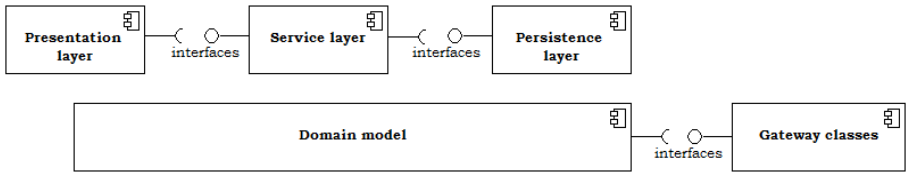

Fig. 1. Architecture

since concrete classes are not referenced, implementations can be easily changed, in many cases with no need to recompile or redeploy.

The diagram (Fig. 1) shows several of the layers of the application. The presentation layer – which contains the code for the interface of the application and may or may not be web based, or even be replaced by another application that uses the functionality of the Experiment Manager – accesses the functionality included in the application through the interfaces exposed by the service layer (which follows the service layer pattern [8]). The service layer – which provides functionality for experiment storage, execution, and monitoring – in turn makes use of the interfaces in the persistence layer in order to store information. The persistence layer is organized according to the DAO pattern [9]. The domain model – which contains entities, such as ExperimentType, Experiment, ExperimentCase, ExperimentStatus, ExperimentResult, that clearly map to their counterparts in reality – supports the rest of the system and also contains the logic that hides the existence of a native system, which it accesses in a decoupled fashion by means of the interfaces of the gateway classes. The gateway classes, which follow the gateway pattern [8], define interfaces with methods which any implementation to access a native system must include.

The high-level design is completed by some additional components: classes which contain logic that concerns the environment of the application ('application classes'); classes used to implement the functionality which allows to plug additional actions at defined extension points ('hook classes'); and classes that correspond to a supplementary utility which allows to define new experiment types.

Many of the technologies used in the application have standards associated to them, with which the Experiment Manager complies, making the application very portable. The language of choice was Java. For the web presentation layer the MVC pattern was followed, and the JavaServer Faces 1.1 framework defined by the JSR 127 [10] standard was used. The interface is based on portlets, and the applications adheres to the Portlet API 1.0 standard defined by the JSR 168 [10]. In order to ensure a consistent look & feel, cascading style sheets (CSS) were used. The views are jsp pages. The Spring framework is used in the middle layers of the application, which allows to change implementations easily – in many cases with the sole modification of a configuration file. The application provides a persistence mechanism for Db4o, an object oriented database system, which helped keep the domain model clean and close to reality. An implementation of

the gateway classes for Condor is provided, this implementations makes use of the Condor SOAP API.

Thanks to the decoupled architecture which was obtained – and with the help of Spring – different implementations for the aforementioned components can be used. Also, several design patterns from [8,9,11] contributed in achieving a well-rounded, flexible, maintainable and extensible design. As a result, the portal, the presentation layer, the native system, or the persistence layer (including the database system used) could potentially be changed with minimal or no impact on the rest of the application. So could be other aspects such as the implementation of the services, the look & feel, the language of the application, and of course the code plugged in the hooks. The application could even be reused as an API for experiment management.

4 Scope

This section describes the scope of the Experiment Manager. Each functionality is accompanied by a brief comment.

Experiment Type Definition An administrative user can create a new experiment type, which implies that some application in the native system will thereafter be accessible by means of the Experiment Manager. This is the only functionality implemented by a separate, supplementary application, which is a command line utility that administrative users can access remotely (e.g. from a ssh console).

Experiment Definition End users can define experiments that belong to one of the existing experiment types using this functionality. All users with access to a certain experiment type can work on experiments of that type, regardless of which user defined it in the first place.

Experiment Case Definition End users can define experiment cases within a given experiment. The information for each case is given in a file with the input for the corresponding variation. Furthermore, several cases can be defined in a single action , in the form of a batch of experiment cases, and in this case a zip file containing an input file for each simple case has to be provided. Any user with access to the corresponding experiment type can add cases to any existing experiment of that type.

Query of Experiment Status and Ability to Stop the Running Cases End users can query the status of the experiments and the experiment cases. They can also stop the execution of a specific experiment case or of all the experiment cases of a given experiment which are running at the time of requesting they be stopped.

Experiment Result Retrieval End users can download results directly from the portal. For experiment case results, a zip file containing all the files corresponding to that case is provided. For experiment results, a zip file containing a directory for each experiment case is generated; in each directory all files corresponding to the case can be found.

Experiment Search Based on Metadata End users can search experiments based on their metadata (name, description, author, creation date). The search functionality is very flexible, allowing to use a search criteria that includes any, all, or none of the metadata. For metadata information consisting of text, the search can be done on parts of the text, even regardless of casing issues.

Query of Experiment Details End users can view detailed information about the experiments. This includes experiment metadata as well as information on the cases of a given experiment. For each experiment case, the original input can be retrieved. There also exists the ability to edit certain information; a basic concurrency management mechanism has been implemented to prevent inconsistencies.

Experiment Persistence Experiments are automatically stored by the Experiment Manager using a database.

Automatic Monitoring Experiment cases are periodically checked for results, which are automatically retrieved from the native system and made available at the location of the Experiment Manager.

5 Distinctive Features

5.1 Existing Tools

Several tools were analyzed as part of the effort to determine which contributions could be made to the state of the art. This section presents some of them, which are remarkable for having one or more of the following characteristics: being feature-rich, well-rounded, freely available, or somehow an example of a aspect worth mentioning.

A special focus has been placed on portlet based solutions. Among these, OGCE [12] and GridPortlets [13] stand out as two interesting freely available tools.

The Nimrod Toolkit [14], which includes a portal that is not based on portlet technology, is arguably one of the most complete tools that were analyzed. Another tool which is not portlet based is a web portal found among the Condor tools at [3]. Both are available for download.

The National University of Singapore uses portlets in a portal of its own [15]. The National Library of Medicine makes available an interface for searching genetic sequences in its databases as a web portal [16], illustrating a use case mentioned in the Introduction; this web portal is not based on portlet technology.

Following there is a description of each of the areas in which opportunities for making contributions were identified. For each of them, there is also an explanation of how they were included in the Experiment Manager.

5.2 Support for Parameter-sweep Applications

Few tools provide support for parameter-sweep studies. Among the portlet based ones, an example is [15]. The Nimrod Toolkit [14] is arguably one of the most

complete tools as to its parameter-sweep support. Most of the portlet based tools that allow to access clusters and grids from a web portal allow to specify just one input argument, which is usually restricted to be just text.

Some of the tools that do support parameter-sweep applications are not freely available, are specific to the portals for which they were developed, or are not specifically suited for use with Condor. Without parameter-sweep support, each variation must be launched individually, and variation information, inputs and outputs must be organized manually. The Experiment Manager presented here supports parameter-sweep applications.

Several aspects of the design collaborate in implementing this distinctive feature. First, each variation in a parameter-sweep study is represented by a full-blown entity of the system – the experiment case. Moreover, the concept of a batch of experiment cases allows to define several cases in one action, which becomes very important considering the number of variations that can be involved. In the functionality of querying of experiment status, care has been taken to present the cases clearly and in relation to their containing experiment. This also applies to the functionality of querying of experiment details, which also includes the ability to retrieve the original input of any case, therefore preventing losing track of any variation. In the functionality of experiment result retrieval, the use of zip files, organized with a directory for each case, helps in providing a convenient way to access the output of the studies.

5.3 Focus on Experiments

Most portlet based tools available for accessing clusters and grids are generic and allow to execute commands on the underlying system, but they treat those commands as exactly that – executables. While this is reasonable for such tools, dealing with just 'executables' prevents from associating with them any information which does not pertain to executables. In the case of the Experiment Manager, it is known beforehand that experiments – rather than plain executables – will be run. It is then advantageous to turn experiments into first-level concepts of the system as proposed in [1]. Furthermore, the Experiment Manager make variations first-level entities as well, in the form of experiment cases. The tool developed is experiment-oriented, which allows to associate experiment metadata (such as name, description, author or creation date) to the experiments. Having experiments as first-level entities also allows to relate experiment cases that belong together.

The implementation of this distinctive feature is very clear in the functionalities of experiment definition and experiment case definition, which treat experiments and their cases as full-blown concepts. The ability to search experiments based on metadata would not be possible if the experiments were not treated as such by the application. The implementation of a persistence mechanism for an object oriented database also reinforces this feature, since the experiments, the experiment cases, and all other entities are stored as such. Indeed, every functionality of the Experiment Manager shows how this feature has been considered.

5.4 Support for Collaboration

Many of the available portlet based solutions, while suited for integration into web portals, provide users their own workspace and do not particularly address the need of researchers to work together in teams and share information. Those tools which store information generally do so using the name of the user of the portal – users access the portlets as they personalized them, but they are the only ones that can see the information they contributed. Many of the solutions which are not based on portlets also do not address this concern.

The Experiment Manager specifically considers that scientists may be part of research groups and need to share experiment information. All members of a given research group are to be able to work with any of the experiments defined by members of the team.

The decision to provide a web interface is especially helpful in achieving this distinctive feature, web applications are very well suited to sharing information. The incorporation of the concept of an experiment type also aids in implementing this feature. By including this concept, there exists the ability to relate a given set of experiments to a certain type, and then all users that should be able to access experiments of a specific experiment type can be granted access to them. Also, since any user with access to a given experiment type can add cases to any experiment of that type, scientists can continue each other's work. All the other functionalities accessible from the web interface show the implementation of this feature as well, since all users with access to a certain experiment type can operate on experiments of that type.

5.5 Restricted Access to Underlying System Commands

Many of the freely available portlet based solutions, as well as many of the ones which do not use portlets, ask the end user for low level data, such as the path to the executable to run, the arguments, or the path to output files. While this may be appropriate for the purpose of those tools, in a scenario like the one described in this work it has a number of disadvantages: it could pose a security risk, since the end user is able to decide the executable to run; prevents from being able to easily relate and keep together all runs of a given executable; makes it more cumbersome to repeatedly execute the same executable, since in each run the same information has to be entered again; does not allow to grant access to users based on the executable to run and the user wishing to run it. The need to restrict access to underlying system information is mentioned in work such as [17] as well as in discussion forums on the subject.

Among the tools analyzed, the portal corresponding to the Condor tools at [3] avoids asking the end user the path of the executable to run. The Experiment Manager considers that end users should not be able to decide this low level aspects, in particular the command to execute.

The functionality of experiment type definition is central to the implementation of this distinctive feature. By introducing the concept of an experiment

type, the decision to access an underlying application – that supports experiments of the corresponding type – and the actual execution of the application can be separated into two distinct stages. In this way, an administrative user decides that a certain application should be accessible and defines an experiment type that represents the kind of studies that application can conduct. End users granted access to an experiment type are not informed nor can they decide on the command that will be executed in the native system.

5.6 Enhanced Access to Results and Aid in Interpreting them

Many of the portlet based tools allow to execute a job on the underlying system, but the ability to get the results from the portal is – if existing – limited. The ability to transfer input and output files is many times restricted to the standard input and the standard output.

An application that supports conducting certain experiments could produce results in files apart from the standard output, or even generate several files as the result of a given experiment. The Experiment Manager considers this need and consequently allows to retrieve from the portal all files produced when executing an experiment. Furthermore, it includes an experimental feature that allows to plug additional actions at certain extension points – called hooks — in the application. This allows to automatically act on the results, even creating a feedback loop by which actions on the Experiment Manager itself can be triggered. None of the existing solutions found include such a feature, which is very innovative.

This distinctive feature is implemented by the functionality of experiment result retrieval. In the first place, all files generated by the underlying application are retrieved – not just the standard output. Secondly, providing results in the form of a zip file, with a directory for each case, is convenient in keeping results organized. Finally, the incorporation of the hooks feature allows to automatically act on results, even triggering actions in the very same Experiment Manager. The hooks feature has been implemented for the event of an experiment case result becoming available. The automatic monitoring functionality also helps in implementing this feature, since it allows to perform the actions associated with a given hook as soon as the results for a relevant experiment case are detected.

5.7 Incorporation of Web 2.0 Concepts

The so called 'Web 2.0' technologies have become increasingly popular, as it has the interest in using them in scientific portals. This was the main topic of interest in the Grid Computing Environments (GCE) workshop in 2007 [18]. These technologies, and Ajax in particular, can improve the user experience significantly. Despite this, few solutions for cluster and grid usage make use of this concepts. For instance, some tools allow to query the status of the experiments, but the query has to be manually repeated in order to update the information. The Experiment Manager uses Ajax technology to monitor the general status of

experiments, thus reducing the amount of manual queries – and page reloads – needed.

The functionality of querying of experiment status shows the incorporation of Ajax technology applied to the status of an experiment. Ajax is also internally used to retrieve the latest available results for a given experiment.

6 Sample Usage

6.1 Environment Configuration

Some configuration is required before starting to use the Experiment Manager. In completing the corresponding settings, the flexibility of the application is clearly shown. This section describes the essential configuration.

The application uses a 'local store' – which is ultimately space in permanent storage – to save data related to running the experiments. Therefore, a directory for permanent data (such as experiment case inputs and outputs) and a directory for temporary data (such as the current result for an experiment, comprised of the available case results) are needed. Both are indicated in a properties file, called wexpman.properties, in the form wexpman.permanentDataDirectory = path_to_dir and wexpman.tempDataDirectory = path_to_dir.

The Experiment Manager already implements the necessary code to use a native system managed by Condor. The relevant settings are indicated in a properties file, called nativeSystem.properties. If Condor is used, the submit node is indicated as an initialization parameter to the corresponding gateway class, in the form nativeSystem.gatewayInitParam.0=http://host:port.

A database is used to store experiment information, including information to reconnect to jobs in the native system should the application be down for some time (e.g. because of maintenance tasks). The relevant settings are indicated in a properties file, called dbconn.properties. An implementation for Db4o is already provided, and if used the path to the database file is indicated in the form dbconn.databaseFile=path_to_file in the properties file.

6.2 Sample Experiment

In order to design a sample experiment to show the capabilities of the Experiment Manager, let the problem of modeling a cache be considered. An experiment can then be defined to analyze how certain characteristics are affected when specific parameters vary.

This example employs the CACTI [19] tool which, among other features, estimates cache access times based on parameters such as cache size, cache line size, associativity, feature size and number of banks. Among the several versions of CACTI which are available, version 3.2 was used in this case; the example can be nevertheless extrapolated to newer versions. The choice of version obeys to practical reasons: version 6 has not been yet released at the time of this writing; version 5 takes 36 parameters, which would make the example more complex;

and while version 4 takes the same amount of parameters as version 3 – as little as 5 parameters in its simple usage – some sample outputs were available for version 3, which allowed to better check the results obtained. It is noteworthy that version 6 will support specifying cache parameters as an input file to the CACTI tool, instead of doing so using arguments in the command line.

The source code for CACTI 3.2 was obtained from [19] and then compiled both using the `make` command and the `condor_compile` command combined with `gcc` (`make` could also have been used to condense several statements into one, this depends on the Condor installation). In this way, two executables were obtained, the former being a regular one, the latter being linked with Condor libraries (which allows to take advantage of features such as checkpointing and remote system calls). In this example, the parameters proposed in [20] are used. These parameters vary the associativity and the cache line size, while keeping the cache size, feature size and number of banks constant.[1]

6.3 Actions Performed by Administrative Users

Administrative users are responsible for enabling access to applications in the native system by defining experiment types associated with them. Experiment type information is stored in a properties file, called wexpman.properties. The information included for each experiment type, along with values that will be used in this example (between parenthesis) follows: a type identifier (cacti); a friendly name (CACTI); the job type (discussed below); the path to the executable in the native system; the arguments to the executable (${wexpmanInputFile}); extra files needed for execution (discussed below); the fully qualified name of the class containing the additional actions to carry out as part of the hooks functionality.

Some of the aforementioned information can use further clarification. The acceptable values for job type depend on which software is used to manage the native system; in this case Condor is used and the job type corresponds to the Condor universe (VANILLA for regular executables and STANDARD for executables which were linked with Condor libraries). Therefore, both versions of the CACTI tool that were previously compiled can be used – one will take advantage of the extra features of the STANDARD universe while the other will behave as a regular executable. The arguments to the executable are specified as a string which should contain a customizable placeholder that will be replaced by the name of the file containing information for the corresponding experiment case during execution, in this case the arguments are just the name of that file. Some tools require extra files to be used (and transferred when applicable) along with the main executable file; these are specified using their full path in the native system, separated by commas. Classes that perform customized actions can be developed and plugged at defined extension points, by means of the hook configuration.

[1] While care has been taken in presenting a real-world, sensible use case, the research presented here concerns experiment management – no conclusions should be drawn with respect to the subject of cache modeling.

The Experiment Manager provides information corresponding to each experiment case in a file for that case. This is suitable for many tools and is indeed forthcoming in CACTI 6. Prior versions of CACTI require, however, that the parameters be passed directly in the command line. The Experiment Manager allows to accommodate this requirement by combining the extra files capability with a small wrapper script (Listing 1.1) which is used as the executable and feeds the contents of the experiment case information file to the original executable, which is then indicated as an extra file for the experiment type.

Listing 1.1. Wrapper Script Example

```
#!/bin/sh
args=`cat $1`
./cacti $args
exit 0
```

As it can be seen from the preceding explanation, the Experiment Manager is very flexible in allowing to adapt to different situations, configurations, tools, scenarios, and requirements.

In order to simplify the definition of experiment types, a supplementary command line utility – which administrative users could access remotely – is provided. In addition to including the new experiment type information in the properties file, this utility updates the portlet descriptor file so that portlets associated with the new experiment type are added to the portal interface. Administrative users can access this utility by executing `java -jar wexpmanAdmin.jar -configfile properties_path -portletfile descriptor_path`.

6.4 Actions Performed by End Users

End users define experiments, add experiment cases and get the results for both. They also need to be able to consult the status (of experiments and their cases) and query information. They may decide to stop one or many experiment cases. They expect that information will be available at a later stage. And, since a portlet based interface is already included, they can access all this functionality in a friendly way, such as from a web portal.

This section illustrates many aspects of how a scientist can use the Experiment Manager. In this example, the CACTI tool is run with varying parameters. Cache size is fixed at 65536 bytes, feature size is set to 45 nm and 2 banks are specified. The cache line size is set to 32 bytes in half the tests, and 16 bytes in the other half. The associativity parameter takes values of 1 (direct mapped), 2, 4 and 8 for each cache line size.

Since each experiment case must be associated with a file containing information for that case, the following files are created for this example, and named after the parameter values they contain for simplicity: 65size16blk1a45f2sub; 65size16blk2a45f2sub; 65size16blk4a45f2sub; 65size16blk8a45f2sub; and four similarly named files for 32blk. Furthermore, they are zipped up in a file named cactiInput.zip, so that they can be added all at once as a batch of cases.

The following screenshots show how an end user would define a new experiment, add 8 cases in a batch, and get the results of the experiment. Some other

interesting capabilities are pointed out in the screenshots as well. While both a cacti regular executable and an executable linked with Condor libraries were produced for this example, the screenshots are presented once. The reason is that both the procedure an end user must follow and the results which were obtained with each executable are exactly the same (for the case of the executable linked with Condor libraries, information about checkpointing and remote system calls was written to the standard error, but this does not alter the results). This means that usage of the application is decoupled from the characteristics of the executable in the native system.

In order to perform the actions described, end users will find two portlets for each experiment type they have access to. One of the portlets – named with the experiment type name followed by the text 'Edit Experiments Portlet' – focuses on editing actions, while the other – named with the experiment type name followed by the text 'Monitor Experiments Portlet' – concentrates on monitoring needs.

In the screenshot (Fig. 2) it can be seen how an end user can define a new experiment, using the Edit Experiments Portlet. The image also shows some existing experiments in the Monitor Experiments Portlet.

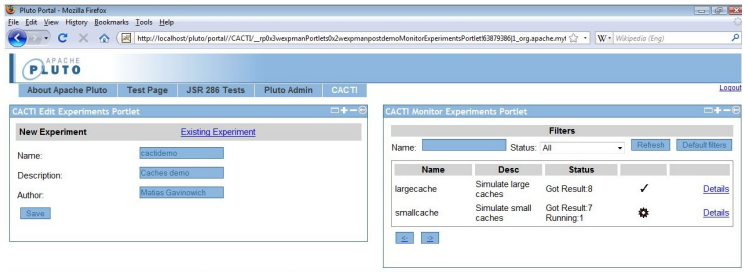

Fig. 2. Experiment definition (left) and existing experiments (right)

An end user can add many cases to an experiment in a single action. The screenshot (Fig. 3) shows how the Edit Experiments Portlet is used for that purpose. A file selection dialog – used to indicate the zip file containing the information for the cases to be added – is also shown in the image. If there had already existed any experiment cases, even if they were defined by another user with access to the CACTI experiment type, they would have appeared in the detailed view of the experiment (shown in the screenshot).

The status of the experiments can be monitored by means of the Monitor Experiments Portlet. A summarized view of all experiments of the CACTI type is available as shown in the image (Fig. 2), and the detailed status for a single experiment can be obtained as illustrated by the screenshot (Fig. 4). It is worth pointing out that the detailed status information is showing that several cases are running at the same time, possibly in different hosts in the native system.

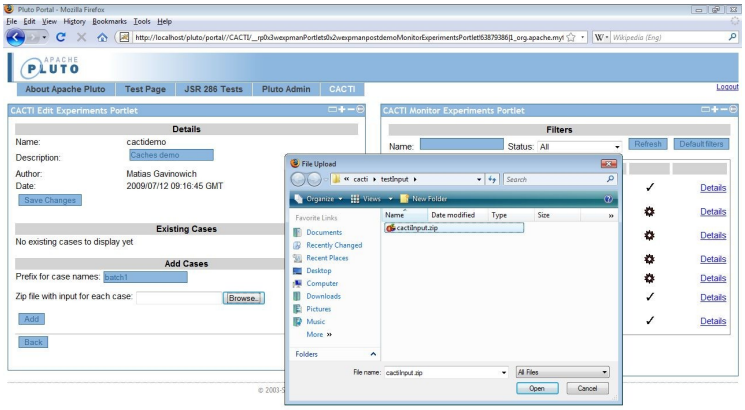

Fig. 3. Experiment case definition

From the same view that presents the detailed status, end users can stop a single case or all cases that belong to the experiment and are currently executing. The input used in any of the experiment cases can be retrieved from the aforementioned view as well. Since the application automatically stores experiments, end users can work with experiments and their cases even long after they have been defined. The screenshot (Fig. 4) also shows the interface included in the Edit Experiments Portlet to search experiments by their metadata; a search which returned the same experiment shown in the Monitor Experiments Portlet has been performed.

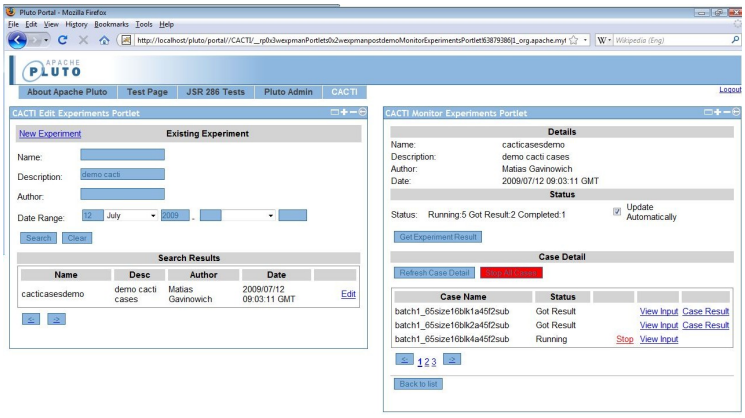

Fig. 4. Query of experiment status (right) and experiment search interface (left)

At any time, end users can request the result of an experiment. They can also retrieve the result of any single case that has its result available (the automatic monitoring functionality periodically checks for results). The screenshot (Fig. 5) shows an experiment with results for all its cases. Experiment results are provided as zip files containing a directory for each experiment case. All output produced by each case is retrieved – the CACTI experiment has written its results to the standard output, but this is not a requirement. Experiment case results are also packaged as zip files and their contents comprise just the files corresponding to the case.

In the image (Fig. 5), the zip file containing the results for the experiment shown in the portlet has been opened and its directories, as well as the contents of one of the directories, can be seen. The output of one of the cases has been retrieved from the zip file and is also shown.

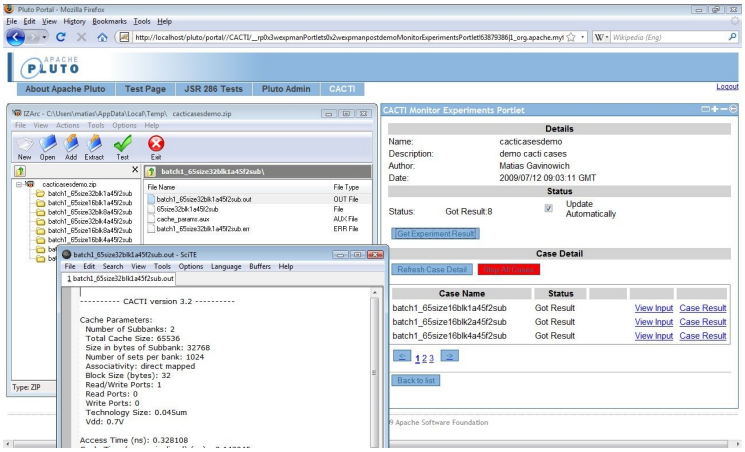

Fig. 5. Experiment result retrieval

In the output shown in the screenshot (Fig. 5), for a direct mapped cache with a cache size of 65536 bytes, a cache line size of 32 bytes, a feature size of 45 nm, and 2 banks, an access time of 0.328108 ns has been estimated by CACTI – other information, not shown in the image, has been produced by the tool as well. For other values of the parameters, different estimations have been obtained, and they can all be consulted in the experiment results.

7 Conclusion

In defining the functionality included in the Experiment Manager, the features of existing tools were carefully considered. The goal has been to produce a single product that would include all of the improvements suggested when analyzing the available solutions. In doing so, a number of ideas which could be useful

to the community are presented, and a description of how each of them was addressed is provided. The application is also very flexible, maintainable, and extensible from the technical point of view.

A very promising subject for future work is the hooks functionality. If the possibility to automatically respond to certain events already makes the application more useful and saves scientists valuable time, then the possibility to create a feedback loop and act on the application itself has a potential which limits are yet to be found.

Acknowledgments

The work described in this paper was supervised by Rosita Wachenchauzer. My deepest thanks to Rosita for her ideas and guidance.

References

1. von Laszewski, G., Zimny, P., Trieu, T., Angulo, D.: The Java CoG Kit Experiment Manager (2005)
2. von Laszewski, G., Foster, I., Gawor, J., Lane, P.: A Java Commodity Grid Kit. Concurrency and Computation: Practice and Experience **13**(8-9) (2001) 645–662
3. Condor Project: Condor Project Homepage
4. Fox, G., Pierce, M., Gannon, D., Thomas, M.: Overview of Grid Computing Environments (2002)
5. Casanova, H., Obertelli, G., Berman, F., Wolski, R.: The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid (2000)
6. Hepper, S., Fischer, P., Hesmer, S., Jacob, R., Taylor, D.S., McCallister, M.: Portlets and Apache Portals. Manning (2005)
7. Pierce, M.: Building Web Portals as Science Gateways
8. Fowler, M.: Patterns of Enterprise Application Architecture. Addison-Wesley (2002)
9. Sun Microsystems: Core J2EE Patterns: Patterns index page
10. Java Community Process: JSRs: Java Specification Requests
11. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley (1994)
12. OGCE: Main Page
13. GridSphere: GridSphere and Grid Portlets Documentation and API
14. Abramson, D., Giddy, J., Kotler, L.: High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid? In: International Parallel and Distributed Processing Symposium (IPDPS), Cancún, Mexico (May 2000) 520–528
15. National University of Singapore: NUSgrid
16. National Library of Medicine: BLAST: Basic Local Alignment and Search Tool
17. Beckles, B.: Building a secure Condor pool in an open academic environment. In: Proceedings of the UK e-Science All Hands Meeting 2005. (2005)
18. Grid Computing Environments: GCE workshop 2007
19. HP Labs: CACTI: An integrated cache and memory access time, cycle time, area, leakage, and dynamic power model
20. Grid Appliance Wiki: Archer: Hands-on tutorial