Mississippi Computational Web Portal

Tomasz Haupt•, Purushotham Bangalore, Gregory Henley
Engineering Research Center at Mississippi State University
P.O.Box 9627, Mississippi State, MS 39762, USA
haupt@erc.msstate.edu, puri@erc.msstate.edu, henley@erc.msstate.edu

SUMMARY
This paper describes design and implementation of an open, extensible object-oriented framework that allows integrating new and legacy components into a single user-friendly Grid Computing Environment. This way we extend the researcher's desktop by providing seamless access to remote resources (that is, hardware, software and data), and thereby simplifying currently difficult to comprehend and changing interfaces and emerging protocols. The user, through the familiar Web Browser interface is able to compose complex computational tasks represented as a collection of middle-tier objects serving as proxies for services rendered by the back-end. The proxies through a grid resource broker use the grid services, as defined by the Global Grid Forum, to access remote computational resources. The middle-tier objects are persistent, and therefore once configured simulation can be reused, shared between users, or transition into operational or educational use.

KEY WORDS: Web Portals, Grid Computing Environments, Access to Remote Resources, Problem Solving Environments, Enterprise Java

1. Introduction

Computational approaches to problem solving have proven their worth in almost every field of human endeavor. Computers are used for modeling and simulating complex scientific and engineering problems, diagnosing medical conditions, controlling industrial equipment, forecasting weather, managing stock portfolios, and for many other purposes. Yet, although there are certainly challenging problems that exceed our ability to solve them, computers are still used less extensively then they should be. To pick just one example, university researchers make extensive use of computers when studying the impact of land use on biodiversity, but city planners selecting routes for new roads or planning new zoning ordinances do not. Yet it is these local dimensions that, ultimately, shape our future.

Computational power constantly is opening new opportunities for numerical simulations, in turn opening opportunities for new science. This computational power is expected to be a low cost alternative for design and validation, boosting efficiency of manufacturing, and be a reliable source of forecasts (e.g., weather, earthquake damage, the stock market, to name just a few domains), as well as all the other claimed and realized advantages for academic computing. As the power of computers has increased and the cost of hardware has decreased, the use of the computer as a tool in complex problem solving has become routine. Moreover, advances in networking technology and computational infrastructure make it possible to construct large scale high-performance distributed computing environments, or "Computational Grids" [1] that provide

• Correspondence to: Tomasz Haupt, ERC at Mississippi State University, P.O. Box 9627, Mississippi State, MS 39762, USA. Tel: (662) 325 4524; Fax: (662) 325 7692

dependable, consistent, and pervasive access to remote high-end computational resources, data repositories and databases, as well as data acquisition systems and instruments. These Computational Grids have potential to fundamentally change the way we think about computing, as our ability to compute will be no longer limited by the resources we currently have on hand. In addition, the access to remote resources promotes sharing of resources and collaboration, and can revolutionize the way the software is distributed and transitioned from researchers and developers to operational or educational use.

The transition from the traditional programming model to a distributed, grid-based programming model is, however, very difficult to achieve. The grid environment is inherently complex, and may intimidate the end user. This is why, despite the fact that projects such as Globus [2] or Legion [3] that build a grid-like environment have reached a certain level of maturity, the actual use of the computational grid is embarrassingly small, as it is hindered by:

- The need of software investment protection, that is, reuse of legacy applications.
- Complexity of the grid environment that typically offers only a command line–based interface that may intimidate the end user accustomed to the Windows environment.
- Complexity of the distributed, heterogeneous back end systems, as the constant demand for faster and faster compute servers drives vendors to introduce more and more sophisticated, scalable architectures including scalable interconnects, where even the end user (consumer rather than code developer) is exposed to all of the nitty-gritty details of the system on which applications are run. In such circumstances, coupling several codes into a single complex application is in most cases prohibitively hard.
- Lack of expertise in modern software engineering by the application domain specialists.

In this paper we describe our design and implementation of an open, extensible object-oriented framework that allows integrating new and legacy components into a single user-friendly problem solving environment. The framework forms the foundation of a Grid Computing Environment [4], which we refer to as the Mississippi Computational Web Portal (MCWP). This way we extend the researcher's desktop by providing seamless access to remote resources (that is, hardware, software and data), and thereby simplifying currently difficult to comprehend and changing interfaces and emerging protocols. The user, through the familiar Web Browser interface is able to compose complex computational tasks represented as a collection of middle-tier objects serving as proxies for services rendered by the back-end. The proxies through a grid resource broker use the grid services, as defined by the Global Grid Forum, to access remote computational resources. The middle-tier objects are persistent, and therefore once configured, a simulation can be reused, shared between users, or transitioned into operational or educational use. This work builds on our previous experience with developing Web-based systems providing Web access to remote resources [5,6].

2. MCWP Driving Applications

Distributed Marine Environment Forecast System (DMEFS)

In a long-term vision, the Distributed Marine Environment Forecast System [7] (DMEFS) is an open framework to simulate the littoral environments across many temporal and spatial scales that will accelerate the evolution of timely and accurate forecasting. This is to be achieved by adaptation of distributed scalable computational technology into oceanic and meteorological predictions (Climate-Weather-Ocean models), and incorporating the latest advances in solution schemes, grid generation, and scientific visualizations. DMEFS is expected to provide a means

for substantially reducing the time to develop, prototype, test, validate, and transition to operations of simulation models as well as support a genuine, synergistic collaboration among the scientists, the software engineers, and the operational users. In other words, the resulting system must provide an environment for:

- model development, including model coupling
- model validation, and data analysis in general
- routine runs of a suite of forecasts
- decision support.

The model developers are expected to be computer savvy domain specialists. On the other hand, operational users who routinely run the simulations to produce daily forecasts have only a limited knowledge on how the simulations actually work, while the decision support is typically interested only in accessing the end results. As the domain expertise level varies from one user category to another, so does their equipment: from the high-end development environment to a portable personal computer.

Distributed Simulation Framework for Seismic Performance for Urban Regions (SPUR)

The long-term objective of the SPUR project [8] is to advance the state-of-art in simulating the effects of a major earthquake on an urban region by the integration of earthquake ground motion modeling with modeling of structural and infrastructure systems using advanced computational and visualization methods. A distributed interactive simulation framework is being created to facilitate investigation of the performance of urban regions resulting from a major earthquake and for education of future earthquake engineers. The goal is to provide damage estimates based on best available information ultimately leading to earthquake related risk analysis enabling policy-makers and emergency response agencies to plan for remediation and emergency response through what-if scenarios.

3. Requirements Analysis

Both projects, DMEFS and SPUR, require support for different kind of users: developers, operators, customers, and administrators. The developer is assumed to be a computer savvy domain specialist whose role is to develop, configure, run, test, and validate models. The operator runs validated codes, either as a part of his or her regular research activities, or to produce routine forecasts. It is critical to hide complexity of both grid environment and applications from the operator. The customer does not run models at all; instead he or she is provided an access to results. The customer does not know, or care, how the data are produced, but he or she must be given tools to identify, localize and access the data of interest. Finally, the administrator is responsible for portal configuration and management of users and resources.

Furthermore, this system is expected to provide an integrated environment where applications can be validated, shared between users, and transitioned for operational use. In other words, the definitions of the user's computational tasks, their configurations and results must be persistent, so they can be reused at later time, shared and/or compared with other results and observational data. This leads to a concept of the user space, referred to as the user context, where the task descriptors are stored.

Finally, the system is expected to hide complexity of the heterogeneous, distributed back end resources. This can be achieved by introducing resource descriptors (machines, batch systems, applications) that provide all necessary information necessary to build the application, configure it, stage the input data, submit it, and retrieve results.

The requested functionality of the system is shown in Figure 1, in form of a high-level use case diagram. On the left side of the diagram the system users (actors) are shown. The administrator role is to manage users and resources. The developer activities involve management of his or her context (creating, cloning and destroying projects and tasks), composing computational tasks, running them, and analyzing the results. To compose a task, the user creates or selects a task context, adds applications descriptors to it, and specifies relationships between the constituent applications. The user may use existing application descriptors in his or her user context, he or she may create a new one by registering a new application (and optionally publish it so others can use it), or import a descriptor published by someone else. Before the task can be submitted, each constituent application has to be configured: the user must specify input files, input parameters and options, final destination of the output files, as well as to specify the target machine on which the application is to be run. A graphical user interface assists the user with the application configuration process. Optionally, the user may publish the task (list of configured applications and their relationships). This is a mechanism for transition of computational tasks from R&D to operations. The operator typically imports a published task, reconfigures it as needed, and runs it. In addition, the operator may schedule the task for routine runs, say, everyday at 10.00 pm. The customer selects the data of interest and analyses them.

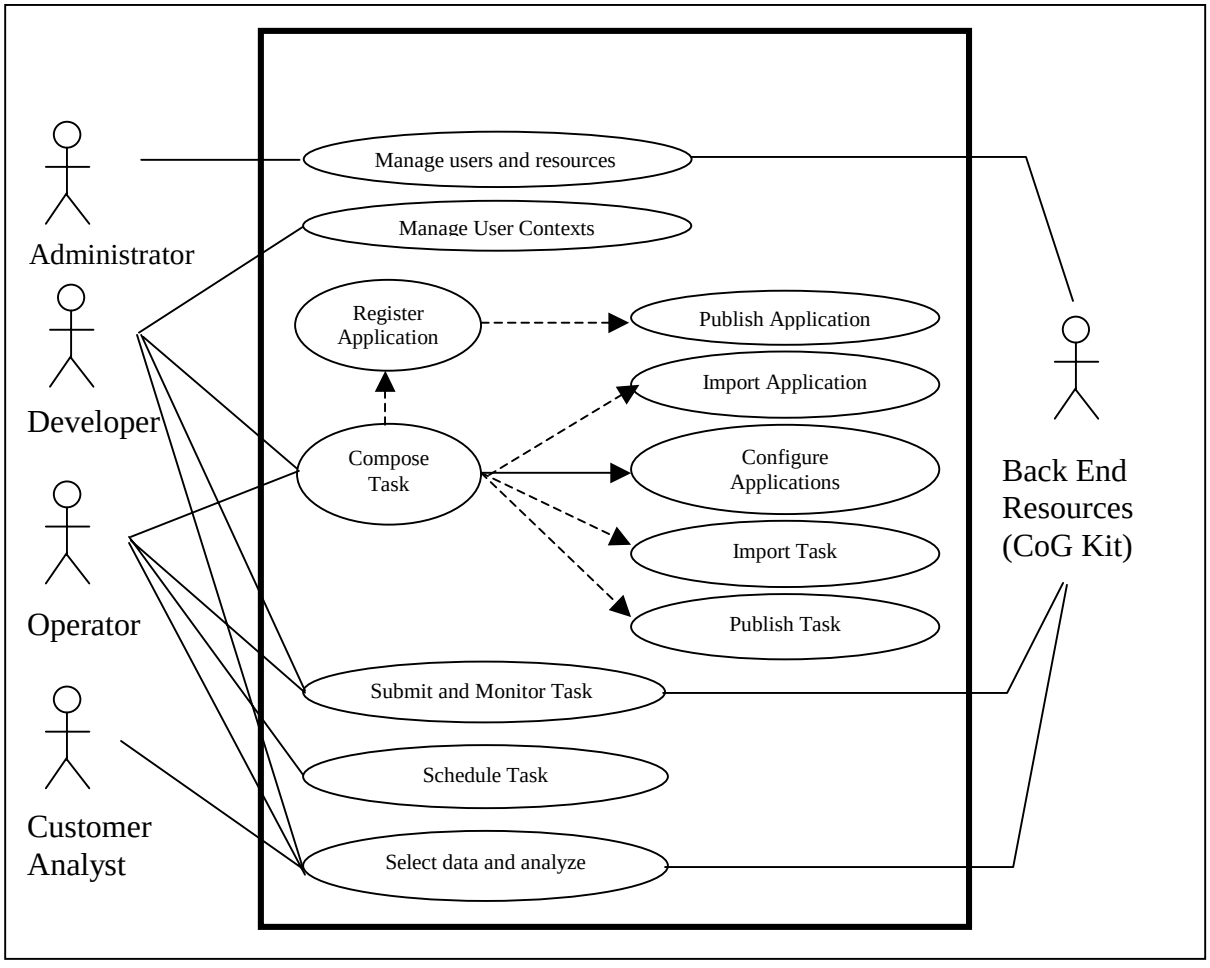

Figure 1: A high-level use case diagram for the Mississippi Computational Web Portal (see text for discussion). [Non standard UML notation: a dashed arrow symbolizes <<extend>> relationship, while solid arrow symbolizes <<use>> relationship]

Note that we treat the back-end systems as a (non-human) actor, that is, we take it as an external system with respect to the computational portal. We communicate with the back end through the grid interface, implemented by Java CoG [9], serving as a client for Globus services. Currently we use CoG for job submission and monitoring (GRAM), for file transfer (GridFTP), and for access to Globus Information services (MDS). In addition, we are using myProxy server[10] to maintain users credentials.

The user contexts, application and task descriptors are entities internal to the portal, and as such they do not belong to the grid environment. The application descriptors relate to the grid environment through resource descriptors (machines, batch systems, etc.) maintained by the portal administrator.

4. Portal Design

For the sake of clarity, extensibility and reusability of the design, we split the whole system into four subsystems: user interface, user contexts, metadata, and resource broker.

User Interface

The front end of the system, that is, the user interface, exposes the functionality of the system to the user. For both our applications, DMEFS and SPUR, it presents a vertically integrated environment for a particular application area (Climate-Weather-Ocean, and Seismic Performance, respectively). Consequently, the MCWP front-end does not exposure directly the grid interface (CoG). The grid interface is deliberately hidden from the end user, and it is used exclusively by the resource broker subsystem.

Both projects mandate a Web browser-based implementation. However, since it is designed as a separate subsystem, the choice of implementation does not affect the other subsystems, as the subsystems interact with each other exclusively through well-defined subsystem interfaces. The other implementation choices include client-side applications and peer-to-peer mechanisms such as email or JXTA[11] . Peer-to-peer mechanisms can be used for portal-to-portal interactions.

The Web browser-based front end comprises a set of dynamically generated Web pages and HTML forms, enhanced with JavaScript for client-side error checking and Java Applets, which implement complex application and data visualization interactive user interfaces. We use Java Server Pages (JSP) technology, including custom tags and JavaBeans, to generate dynamic, context sensitive web contents.

The challenge in design of the front-end is to maximize responsiveness of the system by minimizing the user effort (in terms of number of mouse clicks) to get things done. To this end, we have to maximize the amount of information simultaneously displayed on the screen while keeping the interface intuitive and minimizing the page refresh time. To achieve that we use HTML frames (see Figure 2). The upper frame confirms the DMEFS user role (developer in this case) and provides access to a set of common portal services: remote file browser, status of the user jobs, status of machines, user log, help, change role and sign-out of a DMEFS session. The bottom frame provides an automatically refreshing short status of the machines accessible thorough the portal (in this case two SUN clusters at ERC/MSU), and a placeholder for system messages (none in this particular screen dump). The middle frame is split into three: the left one

exposes the interface of the user context subsystem, the right one the interface of the metadata subsystem, and finally the center one the interface of the resource broker subsystem. The user creates or selects a task context using the context interface, creates or selects application descriptors using the metadata interface, and composes and configures a computational task using the resource broker interface. Other snapshots of the DMEFS and SPUR front-ends are given in Figures 3, 4, 5, and 6.

User Contexts

MCWP is designed as a problem solving environment. Therefore we introduce a notion of persistence to the interface. This is important for many reasons. Typically, the most difficult part of numerical simulations is configuration of the job (or a set a jobs) for scalable execution. This configuration effort may involve accessing and preprocessing input data (including, for example, grid generation) and data post processing such as quick evaluation of the quality of the result (e.g., "has the solver converged to a solution?") and/or visualizations. In many cases, several jobs in sequence (with data-flow type dependencies) or concurrently must be executed to obtain final results. For example, one code may generate boundary conditions for another (such as in littoral waters simulations) or the solution is to be obtained by applying precise but computationally expensive methods to an approximate solution resulting from crude but fast methods (e.g, molecule geometry optimization). Such requires both system and application specific knowledge

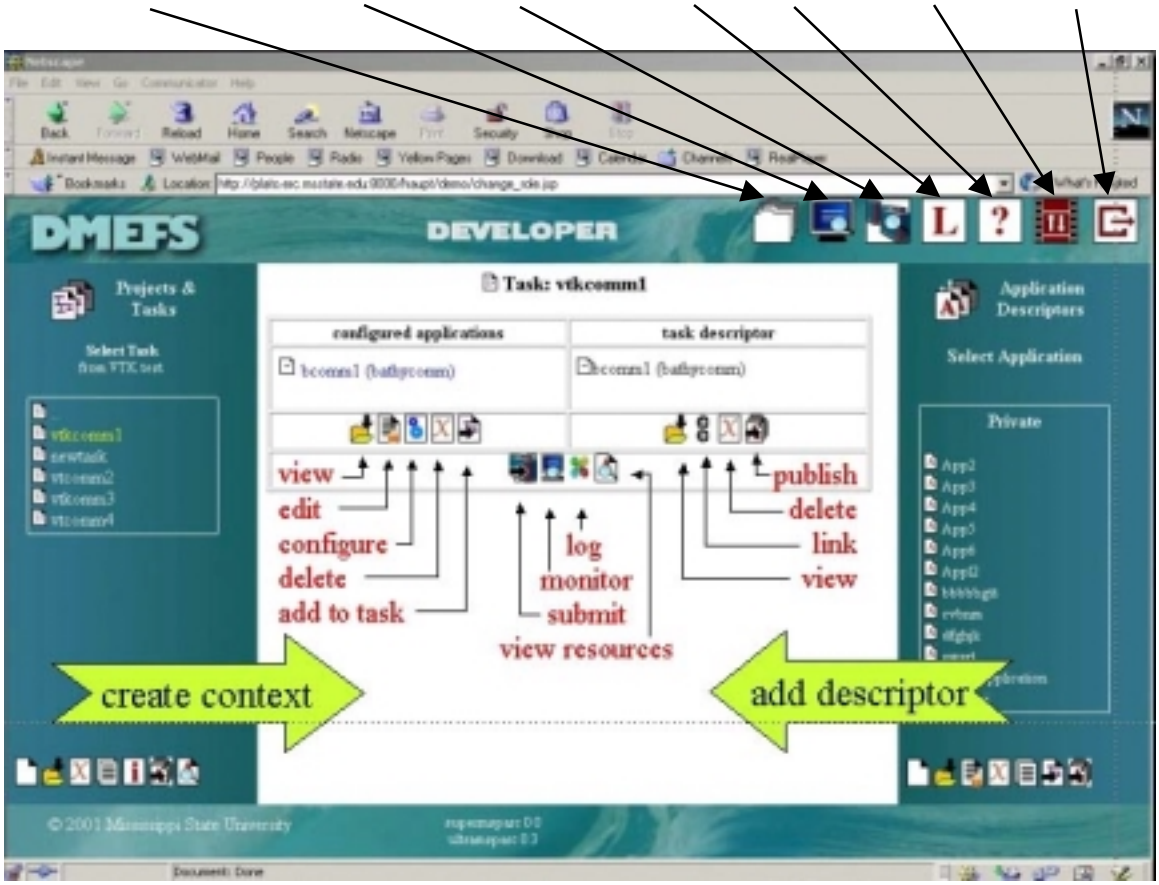

Figure 2: A screen dump of the DMEFS front end (see text for discussion). The screen is divided into 5 frames to hide latency of web access to data, while minimizing the user effort (number of mouse clicks) to perform tasks.

to configure complex applications, and it is labor intensive, requiring highly skilled user labor. Therefore, it is crucial to introduce a framework where configurations can be saved for later reuse and shared between users, thereby also providing validity in the same sense as does a well-designed, repeatable scientific experiment.

Figure 3 demonstrates illustrates another advantage of the user context. Acting as a DMEFS analyst, the user can search his or her user space for files generated by a particular job, or all files generated within a particular task or project to visualize them.

The user context comprises a hierarchy of entities. The user entity (also referred to as a user context) represents the user name, preferences and credentials (necessary for allocation of remote resources such as PKI certificate or Kerberos ticket) and also contains one or more project contexts, each of which in turn contains task contexts that comprise application contexts. In this way, the user can organize his or her work into projects, and within a project he or she can construct multi-step computational tasks by adding applications to the task context and defining the data-flow type relationship between them. The user context objects are implemented as entity Enterprise JavaBeans (EJB) that guarantees their persistence, and operations on them as session Enterprise JavaBeans. The session EJB interfaces constitute the interface of the user contexts subsystem.

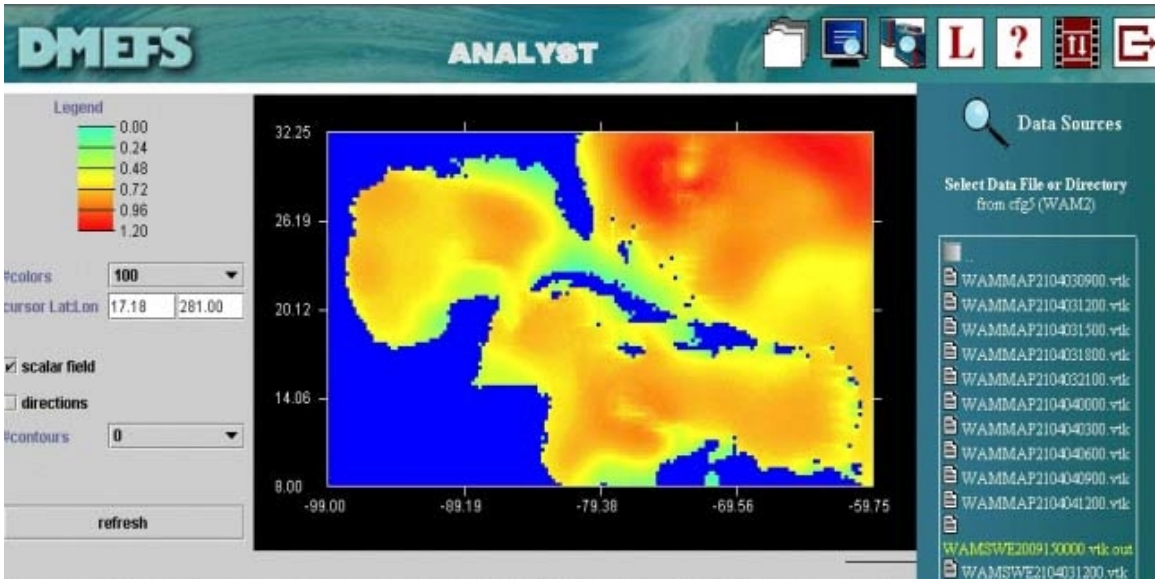

Figure 3. Another snapshot of the DMEFS front end, showing a fragment of the analyst interface. The right frame lists all output files generated within cfg5 task of WAM2 project. The content of the highlighted file (postprocessed by a back-end VTK module) is displayed using Java2D-based applet. The data represent a forecast for the swell height in the Gulf of Mexico.

Metadata

Within MCWP, the computational task defined by the user is expressed in a platform independent way. The separation of the task specification from the resource allocation simplifies the user interface by hiding the details of the heterogeneous back end and automating the process of task definition. The resource broker is then responsible for mapping application specific user requests onto a generic grid interface. The intelligence of the resource broker is built on top of portal information services, i.e., metadata. In our system, the metadata comprise application, task, and machine descriptors, all implemented as XML files, and in addition, a job descriptor implemented as an entity EJB bean.

The **application descriptor** provides all information needed to configure and run the application on a selected machine, and, if feasible, it provides also information how to build it. In addition, it contains a description of the output files that can be used for visualizations, coupling and archiving. The application descriptor can be thought of as a "recipe" of how to submit an application, thus hiding all complexity of it from the user. This is important as the recipe can vary from platform to platform. The application descriptor also solves the problem of software upgrades: all changes are entered to the descriptor, again transparently to the user.

The **task descriptor** represents a computational task hierarchically composed of "atomic" tasks, that is, configured applications, described by the application descriptors. The task descriptor consists of the list of applications, and if the list contains more than one application, it also describes the relationship between constituent applications. In the simplest case of a workflow, completion of one application triggers submission of another. In such a case, the resource broker examines the corresponding application descriptors to determine necessary file transfers, if any.

Each task can be submitted multiple times (with or without modification of its configuration), which results in submission of all its constituent applications prescribed in the task descriptor order. Each instance of a submitted application is referred to as a job. Since a job is a transient object (it disappears from the system after it completes), we introduce a persistent **job descriptor**, an entity EJB that maintains information about the job: time when it was submitted and completed, its exit code, and its configuration including pointers to input and output files (c.f. Figure 3).

Finally, the **machine descriptor** provides information on machines accessible through the portal including the batch system. This information is used by the resource broker to actually submit jobs and to monitor their status. This information is needed only for back-end systems where the Globus metacomputing toolkit is not installed.

The application descriptors are entered to the system through a registration process (e.g., an HTML form). The developer creates an experimental descriptor in his or her private area, optionally including a GUI. Once the application is mature enough, the developer can publish its descriptor so other users can use it. Since the descriptor provides information on how to configure the application, it is much easier for a new user to start using it. Moreover, the descriptor may contain a default or example configuration further simplifying making test runs of a new application. The task descriptor is generated as the response to the user interaction with the portal (Figure 1). The job descriptors are created automatically by the resource broker, and finally the machine descriptors are created by the portal administrator during the machine registration process.

Resource Broker

The primary responsibility of the resource broker is mapping of the user task onto resource allocation requests. Before submitting, the task must be resolved, that is, each atomic task must be assigned the target machine, either explicitly by the user, or implicitly by a resource broker. In addition, the location of the input files, final destination of the output files, as well as values of arguments and switches must be provided, either through the user interface or using defaults (the latter in an operational environment). The resolved task is then submitted: for each atomic task the input files are staged, the executable is submitted, and as soon as the execution completes, the output files are transferred to specified destination, if any.

Figure 4: Automatically generated GUI for the WAM model by applying XSL transformation to the application descriptor which defines five input parameters: restartflag, regioncode, mainregion, dtg and subregion

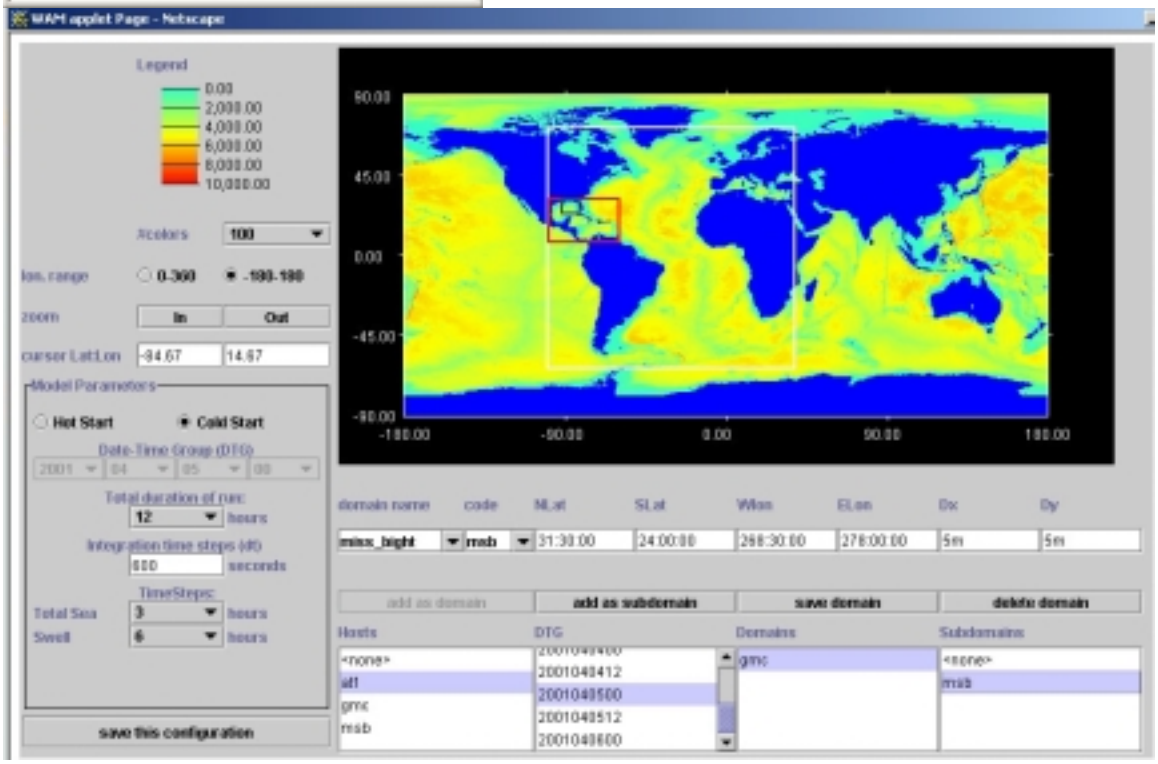

Figure 5: WAM model's GUI provided by the application developer to select all five WAM input parameters: restartflag (Hot or Cold Start), DTG, mainregion, regioncode, and subregion (shown as a nested rubber bands on top of the map showing ETOPO-5 bathymetry). This particular snapshot shows the configuration to run regional forecast for Gulf of Mexico (gmc) for April 5th, 2001, using boundary conditions computed earlier for the Atlantic basin (at1). This run will save boundary conditions for sub-regional, fine-resolution forecast for Mississippi Bight region (msb). This applet examines job descriptors

within the task context to produce a pick list of regions for which boundary conditions have been computed for the selected time; in this example, the Atlantic basin data (column Hosts) are available every 12 hours between April 4th and April 6th (column DTG), and the boundary values for Gulf of Mexico are available (column Domains).

In the future we envision the resource broker to optimize and automate selection of resources to further simplify the use of the grid environment for the end user. In our current implementation, the user picks an application from a list. The portal responds with list of machines where the application can be run (or built). The user selects the target system, and the portal responds with an application GUI that allows the user to configure it as needed (by specifying input files, options, parameters, etc.). The GUI (an html form or an applet) is generated automatically through XSLT from the application description, shown in Figure 4, or is provided by an application developer, as shown in Figure 5. Figure 6 shows an example of GUI of the SPUR portal.

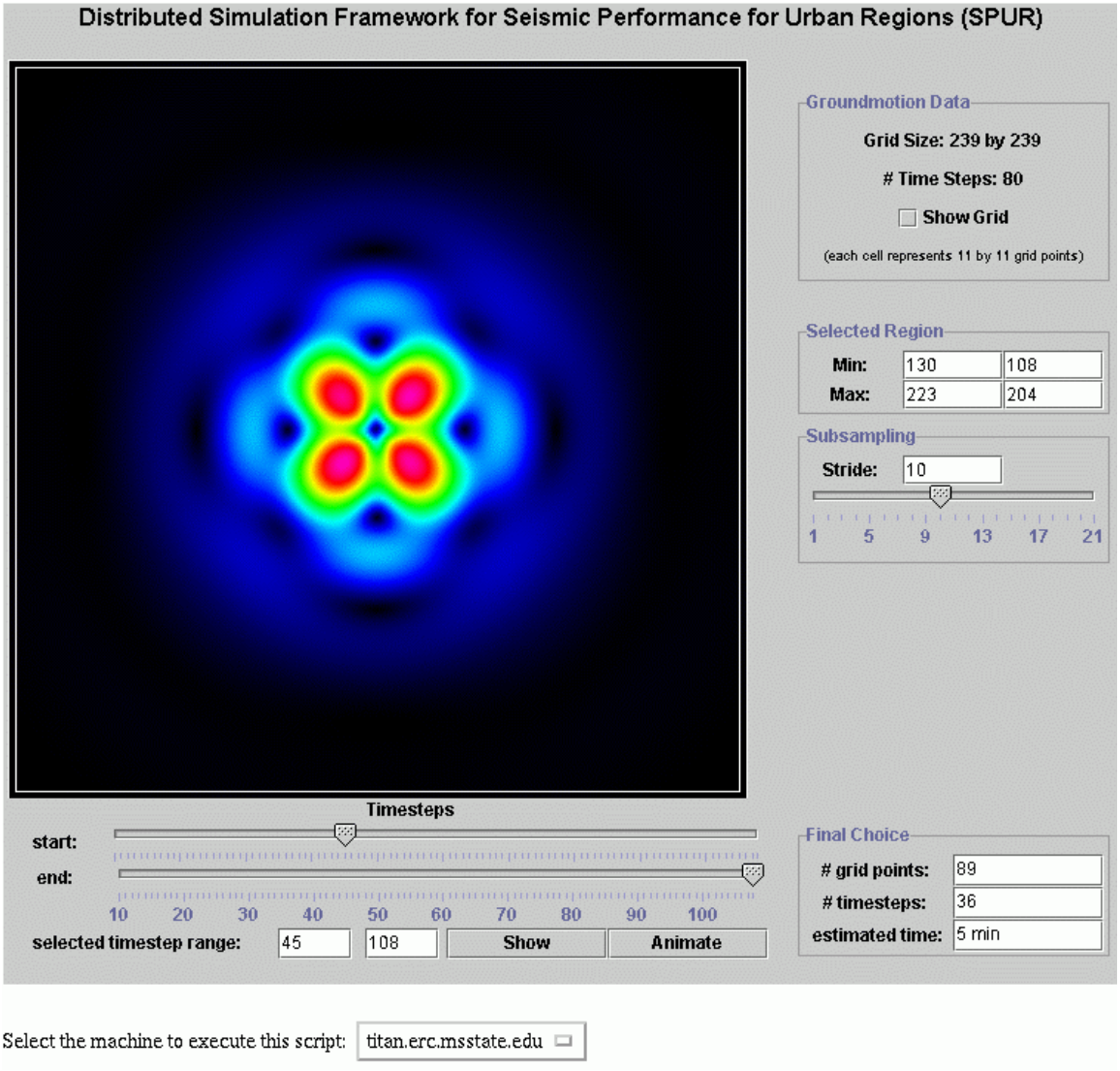

Figure 6: An applet to visualize the ground motion data and selection of a subregion to feed the structure response simulation code.

5. MCWP architecture

The current version of MCWP is realized as a modern multi-tier system, as shown in Figure 7. The Front End is a Web-based interface that allows the user to specify the computational problem, allocate the resources, monitor progress, and analyze results. The Middle Tier is split into the Web Tier (Web Server) and EJB Tier (Application Server). The Web Tier accepts user requests and generates responses through interactions with the application server, which in turn implements the business logic of the portal. It follows that the Web Tier is responsible for the look and feel of the Front End, while the EJB tier is application independent. The Application Server interacts with the back end systems through the Java CoG kit.

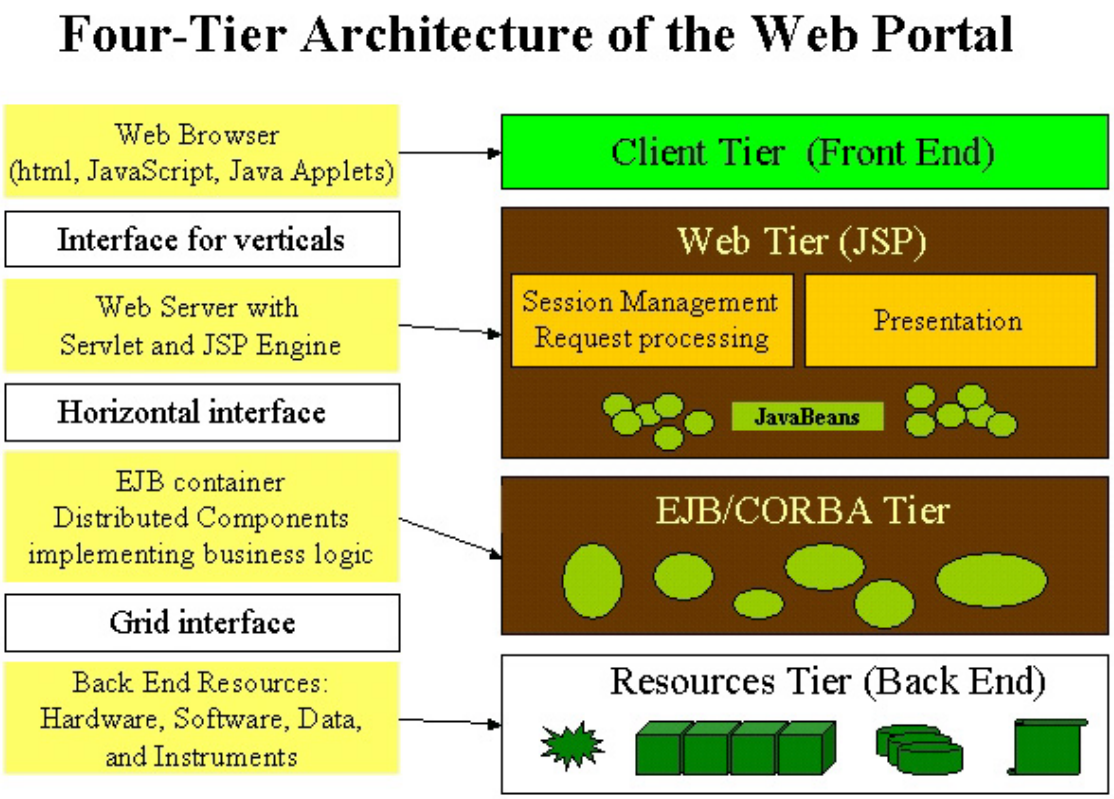

Figure 7: Architecture of the Mississippi Web Portal

6. Summary

The Mississippi Computational Web Portal is realized as a modern multi-tier system where the user can state complex multi-step problems, allocate all resources needed to solve them, and analyze results. A wizard type user interface will help user learn the system and train users new to simulation methodology on the process without encumbering the advanced user, while automating and hiding the unnecessary details. In this environment, definitions of problems, methods of solving them, and their solutions are persistently stored and, consequently, viewed and reused at later time, shared between researchers and engineers, and transitioned for operational or educational use. The Portal is also an environment that extends the user desktop by providing a seamless access to remote computational resources (hardware, software, and data), hiding from the user complexity of a heterogeneous, distributed, high performance back end.

The middle-tier is split into Java Server Pages based Web tier responsible for request processing and dynamic generation of responses, and Enterprise Java Beans based application server. The EJB container is populated by a hierarchy of entity beans representing the state of the system or acting as proxies of services rendered by the back end. Operations on the entity beans are implemented as session beans. This design makes it possible to separate user requests (in terms of application independent task specification) from the back end resource allocation (through platform independent resource specification). The middle-tier provides a transparent mapping

between task and resource specification, hiding complexity of the heterogeneous back-end system from the user. Persistency of the middle-tier object allows for reuse once configured complex tasks and thus transition them into operational use.

The project is in its initial phase. Having built the framework, we plan to extend its functionality by adding support for remote visualizations, grid generation and other tools. In particular, we are interested in a better integration of our system with the desktop resources. In addition, we are interested in extending our framework to support event driven simulations.

References

1. Ian Foster, Carl Kesselman (Editors) The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann Publishers; ISBN: 1558604758, November 1998. See also home page of the Global Grid Forum: http://www.gridforum.org
2. I. Foster, C. Kesselman, Globus: A Metacomputing Infrastructure Toolkit, Intl Journal of Supercomputer Applications 1997, **11**:115-128, see also http://www.globus.org
3. Legion Worldwide Virtual Computer, http://legion.viginia.edu
4. Grid Computing Environments Working Group, http://www.computingportals.org
5. Tomasz Haupt, Erol Akarsu, Geoffrey Fox, WebFlow: a Framework for Web Based Metacomputing, Future Generation Computer Systems 2000, **16**:445-451.
6. Tomasz Haupt, Erol Akarsu, Geoffrey Fox and Choon-Han Youn, The Gateway System: Uniform Web-Based Access to Remote Resources, Concurrency: Practice and Experience 2000, **12**:629-642.
7. DMEFS project, http://www.erc.msstate.edu/~haupt/DMEFS
8. SPUR project, http://www.erc.msstate.edu/~jmeyer/SPUR
9. G. von Laszewski, I. Foster, J. Gawor, W. Smith, and S. Tuecke, CoG Kits: A Bridge between Commodity Distributed Computing and High-Performance Grids, ACM 2000 Java Grande Conference, 2000
10. J. Novotny, V. Welch, MyProxy, http://dast.nlanr.net/Projects/MyProxy/
11. JXTA, http://www.jxta.org