5-10-2003

An Open Framework for Developing Distributed Computing Environments for Multidisciplinary Computational Simulations

Purushotham Venkataramaiah Bangalore

AN OPEN FRAMEWORK FOR DEVELOPING DISTRIBUTED COMPUTING ENVIRONMENTS FOR MULTIDISCIPLINARY COMPUTATIONAL SIMULATIONS

By

Purushotham Venkataramaiah Bangalore

A Dissertation
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy
in Computational Engineering
in the College of Engineering

Mississippi State, Mississippi

May 2003

AN OPEN FRAMEWORK FOR DEVELOPING DISTRIBUTED COMPUTING ENVIRONMENTS FOR MULTIDISCIPLINARY COMPUTATIONAL SIMULATIONS

By

Purushotham Venkataramaiah Bangalore

Approved:

Anthony Skjellum Associate Professor of Computer Science and Engineering (Major Professor)	Donna S. Reese Associate Professor of Computer Science and Engineering (Committee Member)
W. Roger Briley Professor of Mechanical Engineering (Committee Member)	Ratnasingham Shivaji Professor of Mathematics (Committee Member)
Tomasz Haupt Adjunct Professor of Computer Science and Engineering (Committee Member)	Boyd Gatlin Associate Professor of Aerospace Engineering (Graduate Coordinator)
A. Wayne Bennett Dean of Engineering	

Name: Purushotham Venkataramaiah Bangalore

Date of Degree: May 10, 2003

Institution: Mississippi State University

Major Field: Computational Engineering

Major Professor: Dr. Anthony Skjellum

Title of Study: AN OPEN FRAMEWORK FOR DEVELOPING DISTRIBUTED COMPUTING ENVIRONMENTS FOR MULTIDISCIPLINARY COMPUTATIONAL SIMULATIONS

Pages in Study: 181

Candidate for Degree of Doctor of Philosophy

Multidisciplinary computational simulations involve interactions between distributed applications, datasets, products, resources, and users. Because the very nature of the simulation software emphasizes a single-computer, small-usership and audience, the kinds of applications that have been developed often are unfriendly to incorporation into a distributed model. However, advances in networking infrastructure, and the natural tendency for information to be geographically distributed place strong requirements on integration of single-computer codes with distributed information sources, as well as multiple computer codes that are geographically distributed in their execution. The hypothesis of this dissertation is that it is possible, via novel integration of Internet, Distributed Computing, and Grid technologies, to create a distributed computational simulation systems that satisfies the requirements of modern multidisciplinary computational simulation systems without compromising functionality, performance, or security of existing applications. Furthermore, such a system would integrate disparate applications, resources, and users and would improve the productivity of users by providing new functionality not currently available.

The hypothesis is proved constructively by first prototyping the Enterprise Computational Services framework based on a multi-tier architecture using the Java 2 Enterprise Edition platform and Web Services and then two distributed systems, the Distributed Marine Environment Forecast System and Distributed Simulation System for Seismic Performance of Urban Regions, are prototyped using this enabling framework. Several interfaces to the framework are prototyped to illustrate that the same framework can be used to develop multiple front-end clients required to support different types of users within a given computational domain. The two domain specific distributed environments prototyped using the framework illustrate that the framework provides a reusable common infrastructure irrespective of the computational domain. The effectiveness and utility of the distributed system and the framework are demonstrated by using a representative collection of computational simulations. Additional benefits provided by the distributed systems in terms of new functionality provided are evaluated to determine the impact on user productivity. The key contribution of this dissertation is a reusable infrastructure that could evolve to meet the requirements of next-generation hardware and software architectures while supporting interaction between a diverse set of users and distributed computational resources and multidisciplinary applications.

DEDICATION

I would like to dedicate this dissertation to my grandparents.

ACKNOWLEDGMENTS

It has been a long journey to get to this point and it would not have been possible without the cooperation, advice, and help of numerous people. I would like to take this opportunity to thank everyone who has contributed towards reaching this destination.

My advisor, Dr. Anthony Skjellum, has been the motivation and guiding force for me to take up this path. Without his encouragement and guidance this work would not have started. I thank Dr. Skjellum for not only being my advisor but also for being my *Guru* and guiding me through this journey and making sure that I am on the right track. I am also grateful for the financial support provided by Dr. Skjellum during the first part of my Ph.D., first as a graduate assistant and then as a research assistant in the High Performance Computing Laboratory in the Department of Computer Science.

I would like to thank my committee members Dr. W. Roger Briley, Dr. Tomasz Haupt, Dr. Donna Reese, and Dr. Ratnasingham Shivaji for serving on my committee and providing valuable feedback and suggestions. I also thank Dr. Oppenheimer for serving on my committee while Dr. Shivaji was on travel.

The computational engineering program at the Engineering Research Center (ERC) at Mississippi State University (MSU) is a unique multidisciplinary program and I am grateful to MSU for providing an opportunity to be a student in this program. I thank all my professors and instructors during the Ph.D. program for providing excellent multidisciplinary education. I would like to acknowledge the financial support and facilities provided by the ERC during the course of this work.

This work was carried out while I was working as a Research Associate in the ECS Group lead by Dr. Tomasz Haupt at the ERC. Without the encouragement and assistance of Dr. Haupt I would not have undertaken my dissertation work in the area of Grid Computing. I am grateful to Dr. Haupt for providing me an opportunity to work under his supervision in the fascinating area of Grid Computing and allowing me to work on the design and development of the Enterprise Computational Services (ECS) framework.

This work would not be possible without active collaboration and interaction between the researchers and students of the ECS Group as well as researchers at the GeoResources Institute at Stennis. In particular, I would like to thank Mr. Jim Corbin, Dr. Avichal Mehra, Mr. Sachin Bhate, and Dr. Pat Fitzpatrick at Stennis for their inputs, suggestions, and discussions during this work. I would like to extend my sincere thanks to Gregory Henley and Sheikh Ghafoor for always being there whenever there was a question or a problem and suggesting viable alternatives. I would also like to thank members of the ECS Group Anand Kumar Kalyanasundaram, Biju Raman, Kaustubh Kulkarni, Fazal Saiyed, Satyendra Kallur, Shravan Kumar Durvasula, and Maxim Khoutornenko for their contributions and assistance throughout this work.

Thanks to Dr. Don Trotter, Dr. Brad Carter, and Dr. Joe McCaffrey for encouraging me to work on the Distributed Marine Environment Forecast System (DMEFS) that funded part of this work. Special thanks Dr. Stokes and Dr. Haupt at ERC and the multi-institution team including Dr. Gregory Fenves, Dr. Bozidar Stojadinovic, and Mr. Jaesung Park from the Civil Engineering Department at University of California, Berkeley, Dr. Jacobo Bielak and Mr. Antonio Fernandez from the Civil Engineering Department at Carnegie Mellon University, Pittsburgh, PA, and Dr. Joerg Meyer and Mr. Prashant Chopra from the Electrical and Computer Engineering Department at University of California, Irvine for the collaboration on the Seismic Performance of Urban Regions (SPUR) project.

This work was supported in part by the Office of Naval Research under grant N00014-00-1-0886, the National Science Foundation under grant EEC-121989, EPS-0132618, and EIA-0103594, and the National Institutes of Health under grant 1-P20-ES11278-01. I would like to thank all these funding agencies for their support.

I would like to thank all members of the HPC lab in the Department of Computer Science and friends and colleagues at the ERC for their friendship and support. Special thanks to system administration staff and the front office administration staff at the ERC for all their support.

Without the encouragement and sacrifices of my parents and brothers I could not have pursued a Ph.D. I would like to thank them for teaching the importance of education and for thier patience and support through all these years. I would like thank my wife Arathi and son Nikhil for their love, endurance, and understanding throughout this journey.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

CHAPTER Page

APPENDIX

LIST OF FIGURES

FIGURE Page

FIGURE Page

NOMENCLATURE

Internet Related:

ASP	Active Server Pages
B2B	Business-to-Business
CGI	Common Gateway Interface
EJB	Enterprise Java Beans
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol over SSL
J2EE	Java 2 Enterprise Edition
JDBC	Java DataBase Connectivity
JMS	Java Message Service
JNDI	Java Naming Directory Interface
JSP	Java Server Pages
LDAP	Lightweight Directory Access Protocol
RMI	Remote Method Invocation
P2P	Peer-to-Peer
PKI	Public Key Infrastructure
SOAP	Simple Object access Protocol
SSL	Secure Socket Layer
UDDI	Universal Description, Discovery, and Integration
WSDL	Web Services Definition Language

XML	eXtensible Markup Language

Grid Related:

CoG	Commodity Grid Kits
GCE	Grid Computing Environment
GGF	Global Grid Forum
GIS	Grid Information Services
GPDK	Grid Portal Development Kit
GRAM	Globus Resource Allocation Manager
GSI	Grid Security Infrastructure
MDS	Monitoring and Discovery Service
OGSA	Open Grid Services Architecture
OGSI	Open Grid Services Infrastructure
PBS	Portable Batch System
RSL	Resource Specification Language
SRB	Storage Resource Broker

Other Category:

API	Application Programming Interface
DCSE	Distributed Computational Simulation Environment
DCSS	Distributed Computational Simulation System
ECS	Enterprise Computational Services
OLAP	On-Line Analytical Processing
PSE	Problem Solving Environment

CHAPTER I

INTRODUCTION

Advances in numerical techniques and computational methods have resulted in widespread use of computational modeling and simulations to perform complex tasks like weather forecasting, aircraft design, analysis of aircraft and submarine maneuverability, and so on. Over the last two decades scientists and engineers have focused on developing computational models to effectively solve these individual problems separately using modern high performance computing resources. Advances in networking infrastructure and the natural tendency for information to be geographically distributed place strong requirements on integration of these individual applications with distributed information sources. For example, high fidelity ocean modeling requires tight coupling with atmospheric simulations as well as real-time riverine data. The ocean simulation may in turn provide input for subsequent derivative simulations such as spills of possibly chemically reacting liquids. As another example, simulations of the seismic performance of an urban area require the integration of ground motion simulations with the structural response. Furthermore, simulation of a manufacturing system involves dynamical configuration of a tree of distributed simulation processes. Each of these processes is controlled by a different organization: a supplier of a particular subsystem, and a subsystem, which may in turn depend on its suppliers and each supplier may use different software to simulate its particular manufacturing process. One can identify similar examples that require multidisciplinary integration in many application domains.

These simulations have been developed as stand alone applications meant to exploit local resources required to perform the inherent computations. Complex applications

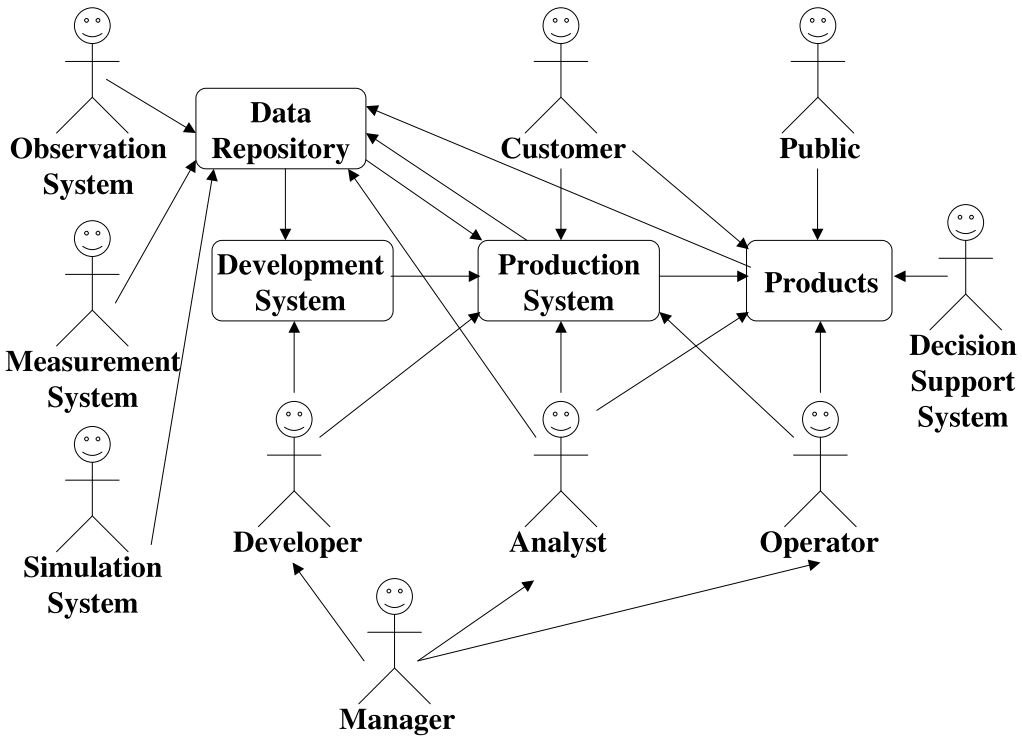
are developed by composing/integrating several stand alone applications that are geographically distributed and execute on heterogeneous platforms. For example, a typical Computational Field Dynamics (CFD) simulation requires importing and preprocessing of a CAD geometry, creation of surface curves, surface grids, and volume grids, assembling these grids into a global coordinate system, creating flow solver inputs, running the solver, and, finally, post-processing the output data. Many of these steps are currently accomplished manually and they require specific training and certain domain expertise (understanding the sequence of steps, data formats and tools for conversions, running applications on different platforms, and validation of the results).

These applications are developed and maintained by different users belonging to different organizations and are not designed to interoperate with other applications, hence making it more difficult to establish meaningful collaboration between such applications. Creating and configuring complex, distributed, multi-step applications is a labor-intensive process and such a process often requires application-specific knowledge available only to expert users. Currently, the user is responsible for managing the execution of each step as well as the flow of information through the different stages of simulation. Data is staged and unstaged manually by the users using command-line tools like *ftp* or *scp*. Consequently, the whole procedure is both complex and error prone even in the simplest case when all data are available locally.

In practice, however, many real-life simulations must be performed on remote high performance systems while the datasets are available from diverse sources: remote file systems, remote databases, remote mass storage systems, web repositories, online instruments and acquisition systems (microscopes, telescopes, medical instruments, remote sensors, and so on). Specific expertise is required to use these datasets and typically involves transferring data from the remote data source to the appropriate compute resource and then applying necessary format conversions or filters on the copied data.

Earlier computational applications were developed, maintained, and used by scientists and engineers who were experts in their domain and hence were experts in using the applications effectively. The interfaces of these applications often proved complex, and therefore difficult to comprehend and master by newcomers. The complexity of the interfaces arose not only from the complexity of the problem to be solved, but also from the fact that these applications were designed to be used only by the developers themselves and then evolved into multi-user applications. In contrast, most commercial applications used for industrial simulations have well-designed interfaces, and come with comprehensive documentation as well as support. However, they are typically based on proprietary implementations and the engineer operating them has little, if any, control over numerical algorithms and methods used, nor the detailed heuristics that go along with many such systems.

Modern computational simulations involve different types of users with diverse expertise. Figure 1.1 illustrates prototypical interaction between users and applications that are distributed in a typical scientific and/or engineering computational simulation. The combination of all the different distributed applications required to perform a computational simulation is referred to as a distributed computational simulation system (DCSS). The complete ecosystem comprised of distributed applications, computational resources, and users is called as a distributed computational simulation environment (DCSE). Applications depend on input data providers like observation systems, measurement systems, and other applications. The developer of the application has to develop, test, and validate the application using these inputs. The developer may be also involved in transition of the validated applications for operational use. An operator would execute the preconfigured application and generate products using the validated application. An analyst could execute the application and/or compare the results produced with another similar application or observation/measurement data. The final products produced by the application could be archived and accessed by a



A distributed computational simulation system requires interaction between different users and subsystems. The diagram shows interaction between users in different roles interacting with specific set of subsystems. For example, user in the "operator" role requires access to the production system and final products of the simulation. The production system uses data from the data repository to generate the desired products while the data repository obtains data from several data sources including other simulations.

Figure 1.1: A distributed computational simulation system with different types of users and subsystems

consumer such as a customer with a defined community or the general public. There might also be a decision support system that would enable users to search and retrieve products of interest. It follows that different users may assume different roles (*e.g.,* developer, integrator, operator, analyst, customer), and it is not unusual that the same user may want to assume different roles at different times.

1.1 Drawbacks/shortcomings of present simulation systems

Modern multidisciplinary computational systems suffer from several shortcomings that prevent them from being effectively used by new users as well as users who are not domain experts. Some of the key shortcomings are as follows:

- User intervention, coordination, and expertise is required to develop, deploy, and transition computational simulations. The current approach of using scripts to manage this problem is unsuitable for end-users such as a customer or the general public who may not have any expertise in using computational simulations. All the minute details about the application, hardware architecture, data sources, data formats, and so on are presently exposed to the end-user. Furthermore, for successful simulations, human interaction, coordination, and steering are also required to make sure that particular inputs/outputs are available in specific locations at specific times. If such inputs/outputs are not available, other fallback approaches must also be devised. Thus, to improve the simulation process user intervention must be minimized and routine tasks must be automated.
- Creating and configuring complex, distributed, multi-step applications is a labor-intensive process and often requires application-specific knowledge available only to expert users. Thus there is a need for saving configurations for later reuse and sharing between users (persistence) thereby also providing repeatability (a requirement for valid experimental procedure based on good scientific methodology).

- While compute resources, data sources, and final products are geographically distributed, the individual applications are stand alone and mainly exploit local resources. To better exploit global resources these applications must be modified to create distributed applications.
- Computational simulations require secure access to distributed heterogeneous resources belonging to different organizations for computation, storage, and data access needs. Users manually log in to each of the remote resource, modify the source code, compile the application, stage datasets, execute the application, monitor the status, and unstage final output results to local workstation or desktop for visualization. This process can be simplified if the user could prepare the applications for execution, monitor the status, stage and unstage data from his or her workstation through a single sign-on without explicitly logging in to each remote resource. Such a setup would also improve the productivity of the user by reducing the time spent in performing these routine tasks. Furthermore, a task stream could be set up and saved on the user's local workstation and save time spent in preparing the task stream.
- While a large number of datasets are online, from the researcher's point of view, there are significant impediments, which often make finding, acquiring, and using these online data practically difficult. The storage format of data is often specific to a particular system and most data archives have their own data management systems with specialized interfaces for navigating their data resources. Virtually none of these data systems interoperate with each other, making it necessary for a user to visit many systems and "learn" multiple interfaces in order to acquire data. Once data are transferred, the data must be converted into the format that the application requires or alternatively modify the application to read the specific format. Often simple format conversions will be insufficient and special interpolation or extrapolation operations will be required.

- Final results or products generated as a result of the simulations do not contain any information about what they represent, what was the source of the datasets used, which input datasets and parameters were used to generate them, which applications were used, and so on. Thus there is a need to annotate the results produced by simulations so that these metadata can be stored in a metadata repository and used to perform search and query operations. Also when complex, multi-step, multi-site applications are executed, a global version control mechanism is required to keep track of different version and source of the applications and datasets used. This information can be also stored as part of the metadata.
- Applications, datasets, and products are typically shared through anonymous ftp sites or web sites. Even though it is possible to protect unauthorized access through password protection and monitor who is downloading from these sites, selectively sharing this information with few users is difficult to achieve using password protection. Thus there is a need to provide access control mechanism through which users can selectively share information with other trusted users. This can also lead to the development of trusted user communities and provide a forum for users to contribute and share information with other members of such a user community. It is also possible to support dynamic and short lived user communities while sharing time sensitive information.
- In figure 1.1, it was observed that there are different types of users involved in a typical simulation system. Also, the user requirements and skill-levels of each user differ. For example, user in the "developer" role would require functionality to prepare a task, access distributed resources, submit the task, monitor the task, and post-process the output produced. An user in the "customer" role just needs to access the final finished product and does not require other simulation capabilities. Thus, it is necessary to provide functionality and interfaces suitable for individual user requirement and skill-level.

1.2 Requirements for Modern Simulation Systems

To improve the capabilities of modern DCSE it is not only necessary to fix the shortcomings mentioned in section 1.1 but also to provide a common reusable infrastructure that can incorporate new functionality and capability required to support next generation hardware and software architectures. Such a common reusable infrastructure that can evolve with new requirements is known as a *framework* and this section describes some of the requirements for such a framework. This list of requirements is by no means complete and new requirements will be added to satisfy the needs of next generation hardware and software architectures.

The excessive user intervention and coordination required to perform routine simulation tasks must be minimized and many of these tasks must be automated and made reusable. Simple interfaces that can be used by novice users or users who do not possess domain expertise must be supported to reduce the time required to train new users. Furthermore, next generation simulation systems would be used by diverse set of users and a single interface may not be sufficient to support all these users. Thus, it is necessary to provide specialized or customized interfaces for different types of users based on their requirements.

Modern scientific and engineering applications evidently require distributed computing to exploit distributed resources (applications, data sources, products, users, and computational resources). Distributed computing techniques provide a natural way of solving complex problems which involve distributed data, resources, and people while addressing issues like performance, scalability, high availability, resource sharing, and fault tolerance. With the increasing popularity of the Internet as well as network enabled languages like Java, distributed applications are widely developed and deployed in the enterprise world whereas they have not yet become popular among scientific and engineering applications that have performance as their primary goal. However, developing distributed applications is not a easy task since such applications must

account for heterogeneity in hardware architectures and software platforms as well as synchronization between the different processes involved in a distributed application.

Developing distributed scientific and engineering applications is even harder since these applications have not only to consider performance but also portability across hardware architectures, software platforms, and parallel programming paradigms (message passing, shared memory, and so on). Thus, it is necessary to explore different options commonly used to develop distributed applications and evaluate their applicability to high performance computational simulations in terms of computer science expertise required, complexity of usage, code modifications necessary, and performance implications on existing applications. Since the majority of these applications are developed and maintained by scientists and engineers who may not have the necessary expertise and background to develop distributed applications a simpler programming paradigm is required. Such a model should hide the details and complexities of computer science issues (such as distributed computing, software maintenance, and security) from the scientists and engineers while allowing them to concentrate on solving the scientific/engineering problem on hand.

Users often have access to several remote high performance computing resource and have to decide where they will execute an application based on the machine load and user's allocation. Typically, this is a manual process that would involve connecting to each machine, determine the queue status, checking if required resources are available, checking if the user has enough allocations left, and then deciding where to submit. One could maintain a directory or repository of compute resources for which the user has allocations and the repository can provide up-to-date information about the machine status, system load, allocations available, and so on. A resource broker can use this information to automate the process of resource selection by determining the available resources that best match user requirements, and then provide the user with a set of possible choices or the best choice that meets the requirements. Users specify "what"

resources and "when" they are needed and the resource broker determines "where" the corresponding resources are available. Thus, one could provide on-demand access to distributed compute resources based on user requirements (assuming that the application is portable across available platforms).

Since multidisciplinary simulations are comprised of several stand alone applications developed and maintained by different users belonging to different organizations, it is often difficult to locate applications that generate a particular product of interest. Even if one finds an application that produces the desired product, domain specific expertise will be required in order to determine different inputs required, format of the inputs, numerical methods used, type of output produced, how to execute the application, and so on. To facilitate locating applications, retrieving metadata about an application, and composing complex multi-step applications one has to provide an application metadata repository. Such a repository can allow users to register applications, share them with selected users, browse available applications based on categories, compose complex multi-step applications, and support search and query operations based on specific application properties or attributes.

Data required to perform a computational simulation come from different sources (instruments, data acquisition systems, simulations, and so on) and are in different formats. Similarly, data produced by these simulations also in different forms and formats (raw data, plots, images, animation, and so on) so that they may be used by different types of users. These datasets do not contain information about how they were obtained, when and where they were obtained, what inputs were used to generate a particular result or what instruments were used to collect these datasets. In most of the scientific applications such information is not documented or is not easily available to the consumer.

If such information about the data can be somehow captured, cataloged, and a repository of such information is maintained, users will be able to make effective use of

these datasets. Such a metadata repository can be used to provide users with capabilities to publish, catalog, search, and retrieve different datasets or products generated by different applications and users. Users will be able to perform search and query operations in order to request specific datasets or products based on data attributes, retrieve selected products, publish their results, and share it with other users. The metadata repository also enables users to easily validate simulation results by comparing and analyzing simulation results with observation data or other similar simulations.

Typically users perform the tasks of pre-processing, submitting jobs, monitoring status, and post-processing by explicitly connecting to remote resources. Instead, users could exploit domain specific environments often referred to as problem solving environments (PSE) [1] to define and describe such operations, submit the configured task, and monitor the task status. A given PSE can provide functionality to compose complex multi-step tasks from individual distributed applications, save task configurations, load earlier configurations, share configurations with other users, select different compute resource, upload/download files from remote resources, and so on. Furthermore, a PSE can also provide a single sign-on capability to access different distributed resources and automate the tasks of staging, submission, monitoring, and unstaging. Such a PSE could also hide the details about different distributed resources, support reuse and sharing of configurations, and provide history of all operations performed by the user. Such persistent history of operations, referred to as pedigree, could be used to reproduce earlier results, localize a particular result, monitor progress, determine what input files and parameters were used, what output files were produced, and when (date and time) and where (hostname) the results were produced. Furthermore, a PSE can also be customized to meet the requirements of different users in terms of the services provided as well as access to these services.

Typically datasets are not available in a format that could be directly used as input to a simulation and often requires format conversions and/or special filters. In order to

perform preprocessing, datasets are copied (using *ftp* or *scp*) to a target remote computer where the simulation is to be executed and new data files required for the simulation are created. This process involves copying data files, executing a converter or filter, and creating new data files. If one is dealing with large-scale datasets, this procedure takes time and significant disk space and leads to duplication of data. Since, one will not require the initial dataset once the new data files are created, copying the data could be eliminated if capability to execute a filter or converter on the data at the source itself is provided. This would require capability to execute a remote procedure call at the data source not just transfer data (*i.e.,* when a dataset is selected in addition to the copy function, filter or converter functions must be provided). If the new data files contain only a subset of the information present in the original file, this approach will reduce the data transfer time and the disk space used.

The same approach can also be extended to deal with large-scale datasets produced by simulations. In fact, this approach can be generalized so that instead of executing a simulation users could query a specific format of a final product based on the attributes of the data. If the product is not available in the format requested, a filter or conversion is executed on the data at the source and the final results are returned to the user. If the data are not at all present then using metadata associated with the dataset the corresponding application that generated the data is executed on the fly or scheduled to execute as a precondition for progress. When the simulation completes the final product is returned to the user and if the simulation could not be completed in real time the user could be informed when the final result is ready.

One can further extend the remote filtering approach to create a knowledge base that could be used to query and retrieve products based on certain attributes and when data is not available appropriate filters or applications are executed to produce the required data. This approach can potentially reduce duplication of simulation executions and the

results of a single simulation could be used to provide different types of products based on user requirements.

Besides integrating multiple stand-alone applications, there is also a need for a collaborative environment for different kinds of users working concertedly in the solution of complex multidisciplinary simulations. Such an environment could reduce duplication of effort, maximize reuse, provide a global view of resource allocation and utilization, and provide accounting and statistics that can be used to estimate present resource utilization and try to predict future requirements based on present utilization. Such a distributed environment can be used to monitor data transfers between different resources and provide caching if multiple applications and users access the same datasets. This environment could also be used to transition scientific and engineering applications developed by researchers for operational use by users who are not domain experts while providing an environment to support further research and development transparent to the operational user.

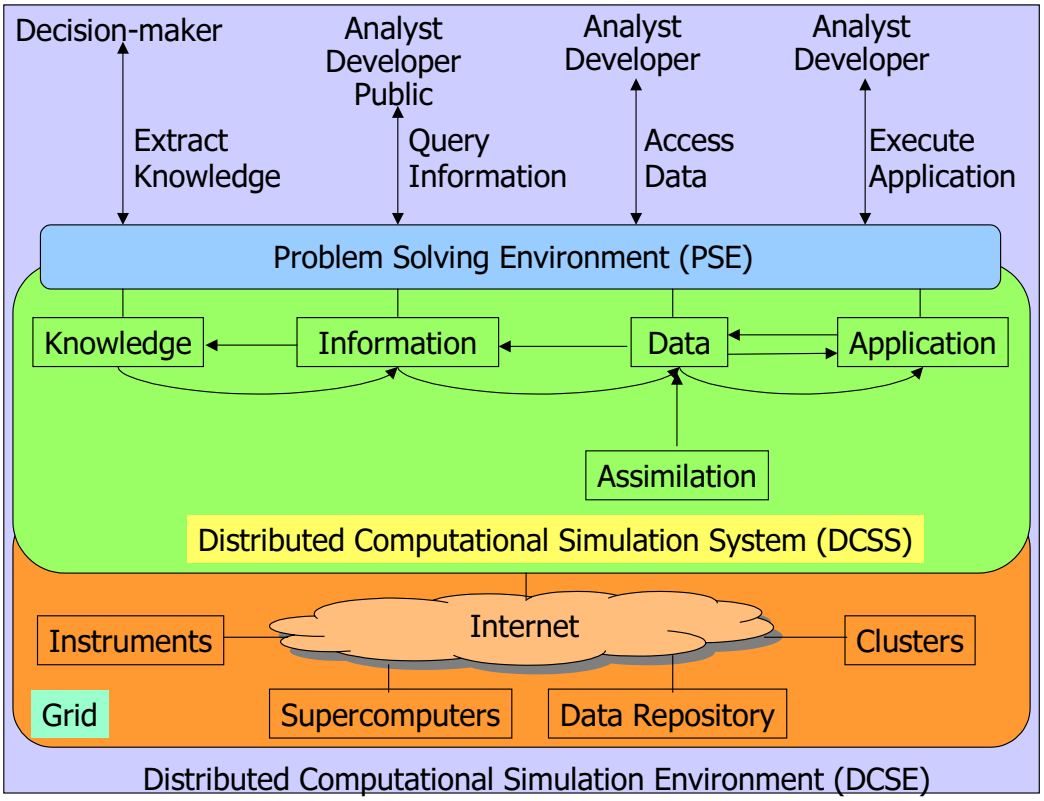
In addition, the environment could enable linking different classes of users over space and time; defining the classes of interactions, enabling allowed interactions, and disabling others. All these interactions form valuable collaborative modes of operation that go beyond data to include persistent program state, historical information about uses of an application, and also form the basis for potential communities of users with common detailed interests. Figure 1.2 provides a graphical representation of such a DCSE that connect diverse set of users, applications, and computational resources through a DCSS. A DCSS is layered on top of existing computational resources connected through Internet and users can access the services provided by a DCSS through user interfaces provided by the PSEs. When a user requests certain information the information repository is first queried and if the requested information is available, appropriate information is returned to the user. If requested information is not available then the data repository is searched to determine if data is available to generate the requested information. If

data is available then data is processed into information. Otherwise, the application repository is queried and applications that can generate the desired data are obtained. Appropriate applications are selected and the data repository is used to obtain any input files required to perform the simulations and the desired output data files are generated. These data files are further processed to create information and provided to the user. All these operations could happen with or without any user intervention and also at each step the DCSS could return results that are an approximate match to the requested information or data. Comparing this diagram with figure 1.1 one can observe that all interaction between users, applications, computational resources are now handled through the DCSS. Thus, a DCSS could hide the complexities of accessing and managing distributed computational resources from scientists and engineers.

1.3 Background

Some of the requirements mentioned in section 1.2 are begin addressed in the literature in different disciplines but there is no single system, of which the author is aware, that can support all these requirements within a common framework.

Distributed object and component technologies such as CORBA [2], DCOM [3], and EJB [4] are widely used to develop distributed enterprise applications. These technologies cannot be directly used to develop high performance scientific and engineering applications since it would require extensive modifications to existing code and expertise in developing distributed applications. Also, these distributed computing technologies are not supported on traditional HPC platforms. Thus developers of these applications rely on hand-coded scripts and user intervention to manage all the distributed computing needs. However, distributed computing technologies could be exploited by a middleware solution to encapsulate and manage the complexities involved in accessing distributed resources.



A distributed computational simulation system (DCSS) provides services to connect distributed applications, computational resources, and users. A problem solving environment (PSE) layered on top of a DCSS provides user interfaces to access the DCSS services. Different users require different functionality and the PSE processes user requests and uses appropriate DCSS services to perform the actions requested by the user. The DCSS maintains knowledge bases, information repositories, data repositories, and application repositories to satisfy different user requirements.

Figure 1.2: Distributed users, computational resources, and applications connected together through a DCSS to form a DCSE

Several key Internet technologies such as Java 2 Enterprise Edition (J2EE) [5], .NET [6], and Web Services [7] have enabled rapid development and deployment of highly available and reliable enterprise applications. These technologies hide the complexities involved in performing a desired operation and the corresponding implementation details from the consumer by providing easy-to-use and customizable interfaces. Users can perform advanced transactions (such as stock trading, banking, shopping, auction) using a web browser without being concerned about how these transactions are performed. Web Services support platform independent and programming language neutral application development. Thus, Internet technologies could be used as a mechanism to deliver the different services required by a DCSS through a collaborative environment with specialized interfaces to support specific needs of different users involved.

The emerging Computational [8] and Data Grids [9] provide infrastructure that allows variety of computational resources, data warehouses, and instruments distributed across multiple administrative domains to create a virtual computer that can be accessed on demand. While computational grids aim to provide virtualization of compute resources, data grids seek to provide virtualization of data access when data is distributed across different storage systems and data access environments. Data grid provides infrastructure for efficient access to large-scale data repositories and archives by creating replicas and replica catalogs. Grid technologies could be used to access the distributed, heterogeneous resources required by a DCSS.

PSEs provide domain specific environments to define a problem, select specific algorithms, create a solution, and visualize the results. Earlier generation PSEs were implemented as standalone applications and were used as a tool for rapid prototyping [10, 11, 12, 13]. Present day PSEs are implemented as typically implemented as client-server applications or computational portals (a web portal for computational requirements) [14] using a three-tier architecture [15]. The evolution of PSEs from

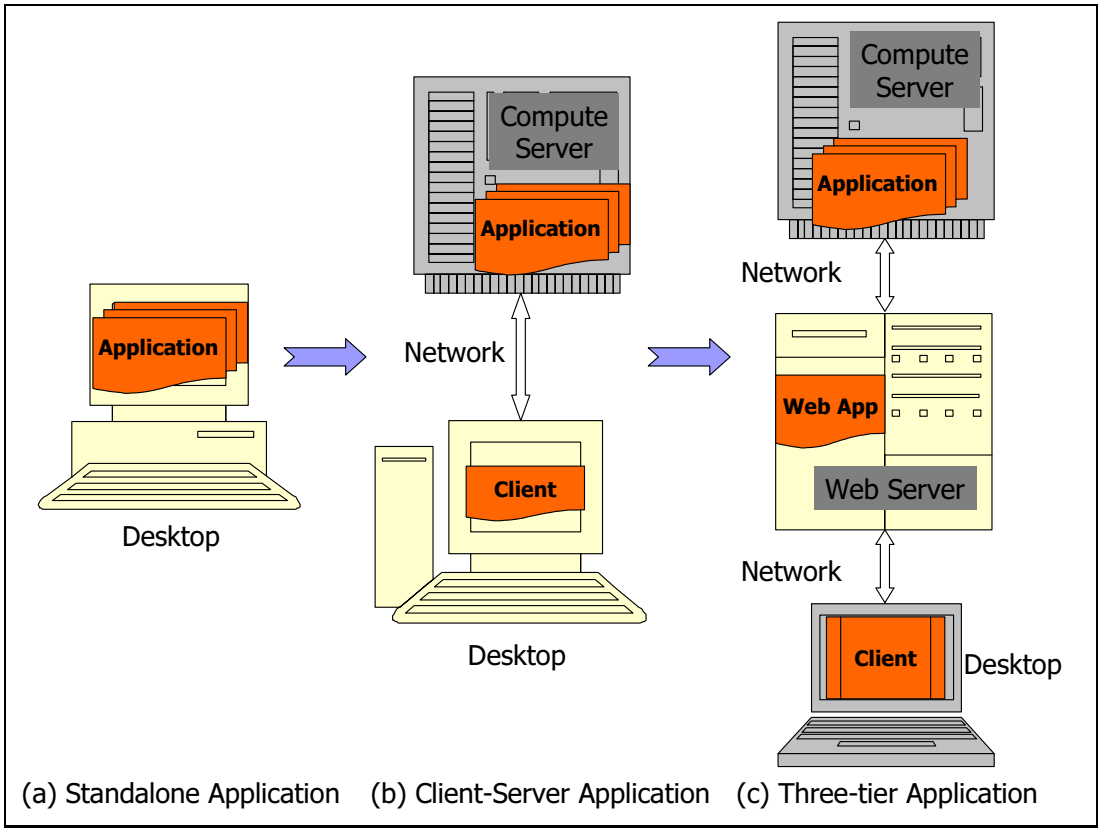


Figure 1.3: Pictorial representation of changes in the architecture of problem solving environments from standalone applications to three-tier applications

standalone applications to applications based on three-tier architecture is illustrated pictorially in figure 1.3. Typical three-tier architectures include a front-end with easy-to-use user-interfaces, a middle-tier that performs request processing and response generation, and a back-end where the user requested operations are performed. The middle-tier performs requested transactions at the back-end on behalf of the user and strives to hide the complexities of the back-end processing from the user. The front-end clients can be web-based browsers or desktop applications written in programming languages like Java, C++, or Visual Basic. Browser-based clients provide ubiquitous access without requiring installation and maintenance of special software and are often referred to as thin-clients. With browser-based clients user interactions are processed by a web server and responses are sent back to the browser for display. Desktop applications provide rich graphical capabilities and faster response time since user interactions are processed on the client side. But desktop applications, also known as thick clients, require software to be installed and upgraded on users desktop. In this dissertation web browser-based PSEs are referred to as computational portals while desktop-based clients are called as PSEs. Thus PSEs and Computational Portals based on multi-tier architectures could be used to hide the complexities of accessing and managing distributed resources from the end-user.

A DCSS that can satisfy all the requirements of modern computational simulations requires a combination of distributed computing, Internet, and grid technologies as illustrated in figure 1.4. Distributed computing technologies could be used to connect standalone applications into a single user-friendly system by combining the Internet and Grid technologies. Internet technologies could provide ubiquitous access, user-friendly interfaces, and collaborative problem solving environments while Grid technologies could form the basis for accessing geographically distributed computational resources. The focus of this dissertation is to develop a reusable infrastructure (framework) that can evolve to meet the requirements of next-generation hardware and software architectures

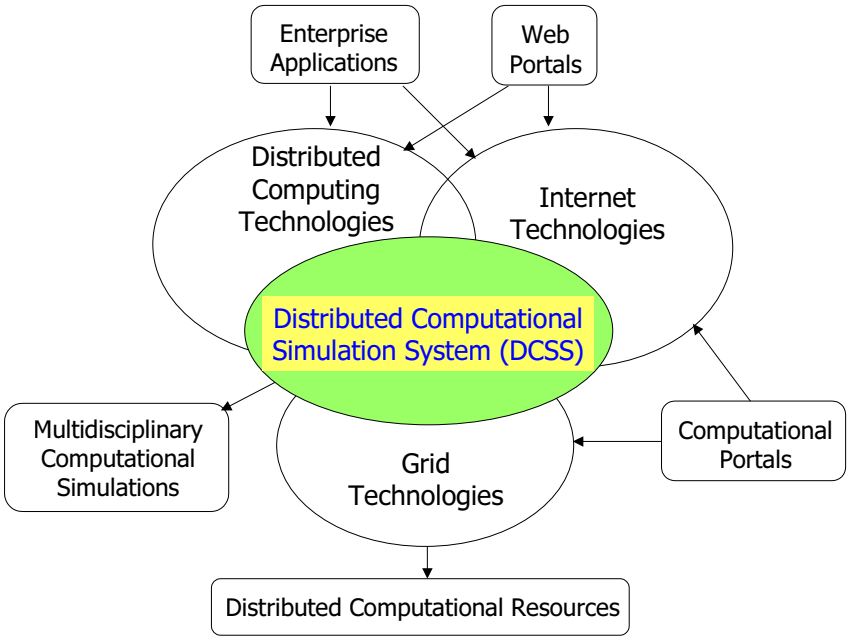


Figure 1.4: Graphical representation of different technologies available to support DCSS

(open) while supporting interaction between diverse set of users and distributed computational resources and applications (distributed computing environment).

1.4 Hypothesis and Approach

The hypothesis of this dissertation is that it is possible, via novel integration of Internet, Distributed Computing, and Grid technologies, to create a distributed computational simulation systems that satisfy the requirements of modern multi-disciplinary computational simulation systems (as described and defined in section 1.2) without compromising functionality, performance, or security of existing applications. Furthermore, such a system would integrate disparate applications, resources, and users and would improve the productivity of users by providing new functionality not currently available.

Since it is not practical to create a single distributed system that is suitable for all computational domains, the common features required for developing DCSS

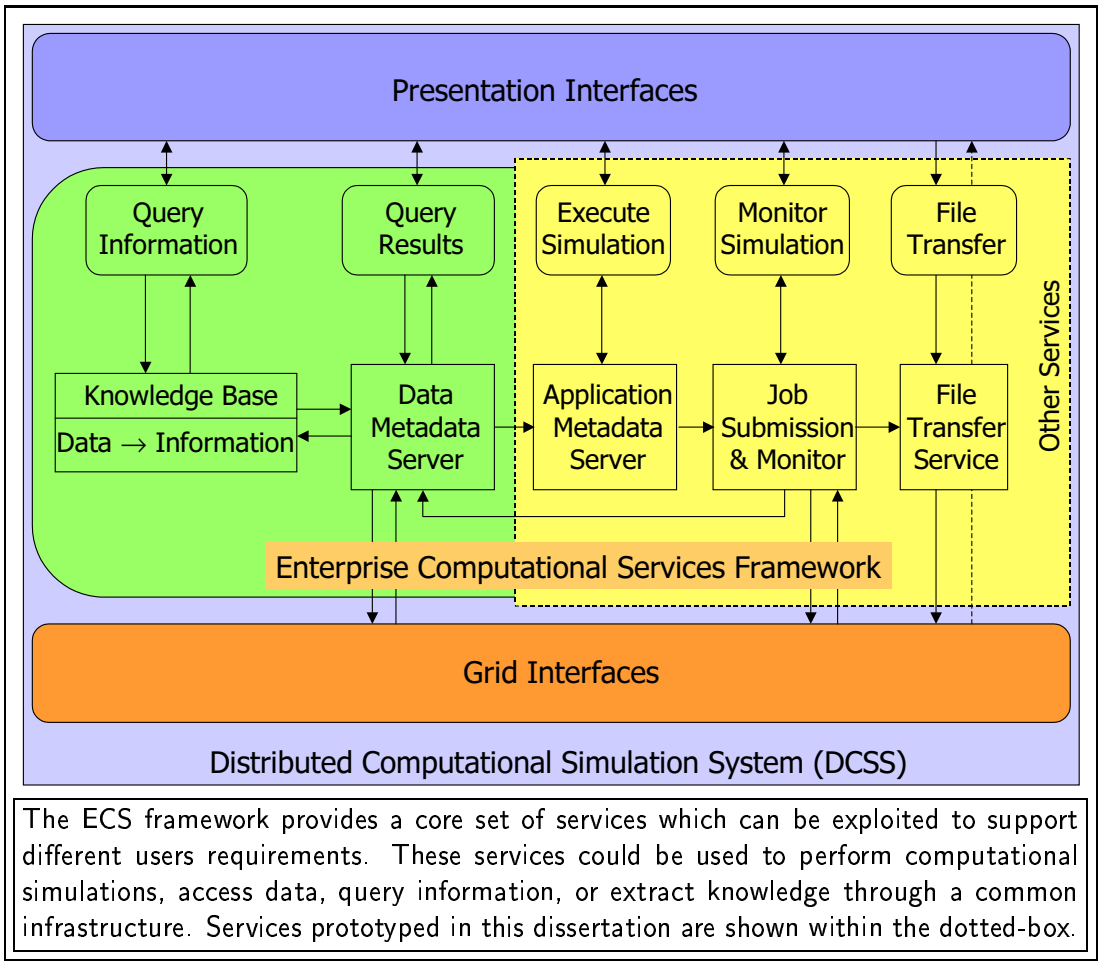


Figure 1.5: Overview of the services provided by the Enterprise Computational Services (ECS) framework

are abstracted here and a components-based open framework, called the Enterprise Computational Services (ECS) Framework, is created to develop different domain specific DCSEs. A DCSS based on the ECS framework shown in figure 1.5 illustrates the interaction between services provided by the ECS framework and the presentation interface. Figure 1.5 provides a close-up view of figure 1.2 with the DCSS enlarged to illustrate how services provided by the ECS framework are used to execute user requests. In this dissertation the services in the dotted-box are prototyped to show the feasibility of realizing a DCSS based on the ECS framework. The focus here is not to implement all the functionality discussed in section 1.2 but to design a common framework that can incorporate such a broad range of functionality required to support multi-disciplinary computational simulations. Such a framework could enable the development of DCSEs suitable for high performance scientific and engineering applications by reducing the complexities involved in accessing and using distributed resources, supporting multiple users with different background and expertise, and improving productivity of users by automating some of the routine tasks.

ECS exploits existing technologies such as Commodity Grid Kits [16] to provide basic services for authentication and authorization, job submission and monitoring, and file transfer required to access the distributed computational resources. In addition to these basic services ECS provides higher-level user services and persistence services. User services such as user workspace, task composer, workflow manager, scheduler, and resource broker are required to create comprehensive problem solving environments (PSEs). Persistence services such as the application metadata repository, job repository, resource repository, and pedigree services provide access to historic information and support reuse and sharing of information among different users of a DCSE. Persistence services are unique contribution of this dissertation and these services could be used irrespective of the mechanism used to access the distributed computational resources. The application metadata repository is used to store information required to configure

and execute an application and this information is used for automatic creation of proxy objects. These proxy objects support creation of complex multistep applications out of standalone applications distributed across multiple organizations without modifying existing applications, thereby providing a framework for creating distributed applications using standalone applications. The ECS framework relying extensively on metadata about the application to automate several tasks that would otherwise require human intervention, thereby improving user productivity.

Even though services provided by the ECS framework are classified differently, all services could be are directly accessible through the ECS interfaces as shown in figure 1.6. User level services typically use the persistence and remote access services to provide high-level functionality required to create PSEs while remote access services exploit the persistence services to automate the tasks of job submission, job monitoring, and file staging and unstaging. To utilize the ECS framework it is not required to use all the services provided, DCSS developers could choose only the required services and provide their own implementation for some of the services or use other third-party services.

The hypothesis is proved constructively by first prototyping the ECS framework based on a multi-tier architecture using the Java 2 Enterprise Edition (J2EE) platform and Web Services and then two distributed systems, the Distributed Marine Environment Forecast System (DMEFS) [17] and Distributed Simulation System for Seismic Performance of Urban Regions (SPUR) [18], are prototyped using this enabling framework. Several interfaces to the framework are prototyped to illustrate that the same framework can be used to develop multiple front-end clients required to support different types of users within a given computational domain. The two domain specific DCSEs prototyped using the framework illustrate that the framework provides a reusable common infrastructure irrespective of the computational domain. The effectiveness and utility of the distributed system and the framework are demonstrated by using representative collection of computational simulations. Additional benefits provided by

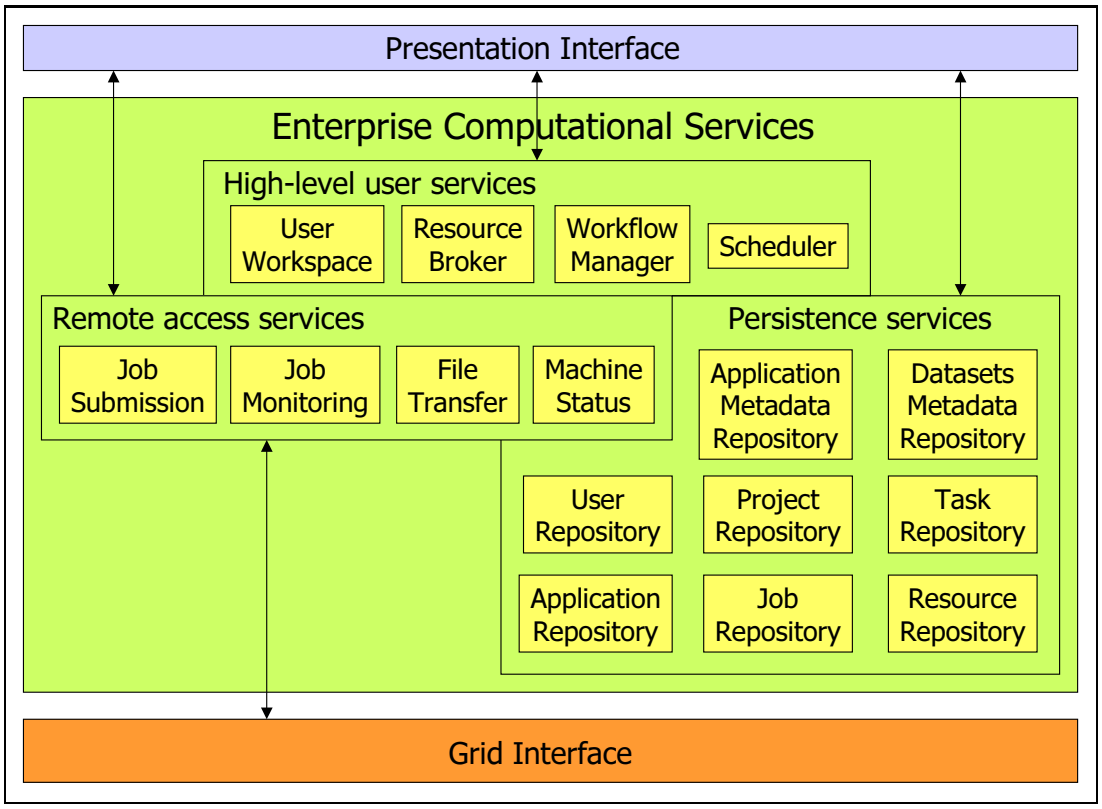


Figure 1.6: Pictorial representation of different types of services provided by the Enterprise Computational Services (ECS) framework, the key contribution of this dissertation

the distributed system in terms of new functionality provided are evaluated to determine the impact on user productivity.

1.5 Contributions

In addition to providing a framework for developing next generation DCSS, this dissertation also makes the following contributions to the area of multi-disciplinary computational simulation systems:

- provides a framework for integrating disparate applications, resources, and users;
- offers novel approach to develop client-server or distributed HPC applications;
- provides novel metadata-based application repository that allows users to register new applications, share applications with other users, browse available applications based on categories, and search and query operations based on specific application properties or attributes;
- provides a persistent workspace with pedigree information where different applications can be composed, configured, submitted for execution, have job status monitored, and configured tasks are shared with other users;
- makes computational simulations accessible to non-experts by hiding complexity and automating routine tasks;
- supports rapid transition of applications from development mode to production mode;
- provides an open architecture to incorporate new services or provide selective access to existing services; and,
- provides an architecture to support the emerging on-demand computing paradigm, virtual simulation facilities, and application service providers for computation oriented software systems.

1.6 Acknowledgements

This work was carried out by the author while working as a member of the research staff in the ECS Group lead by Dr. Tomasz Haupt at the Engineering Research Center, Mississippi State University. This work would not be possible without the vision, leadership, and direction provided by Dr. Haupt as well as interaction and collaboration with other members of the ECS Group [17, 18, 19, 20, 21]. While the author is primarily responsible for designing, prototyping, and testing the Enterprise Java Beans and Javabean interfaces, the key components of the ECS framework shown in figure 1.6, several members of the ECS group have contributed to the development of front-end user interfaces. The distributed environments, DMEFS and SPUR, are collaborative efforts undertaken by the ECS Group at the Engineering Research Center at Mississippi State University that exploits the ECS framework.

1.7 Organization

This dissertation describes the design and prototyping of a DCSE using an open framework to support the requirements of modern multi-disciplinary DCSS. Chapter II briefly describes the various technologies available for developing distributed applications and distributed environments as well as other related efforts to develop high performance distributed environments. Chapter III considers the various research issues involved, and analyzes how other efforts discussed in Chapter II have addressed these issues. Then, the overall architecture and approach to the DCSS and the multiple front-end clients to be supported by the framework are described in turn. Implementation details of the ECS framework are provided in Chapter IV. Chapter V motivates the constructive proof of the hypothesis; it provides the description of distributed systems, DMEFS and SPUR, prototyped using this framework. Chapter V describes how the ECS framework is used to meet the requirements of these systems. Details of representative applications

used to determine the value added by the framework and implications of using the distributed system on the functionality, performance, and security of these applications are also discussed. Chapter VI provides the summary and conclusions while future work is discussed in Chapter VII.

CHAPTER II

LITERATURE SURVEY

In Chapter I various requirements of modern multidisciplinary computational simulation systems were described. In this chapter other efforts in the literature designed to satisfy some of these requirements are discussed.

2.1 Distributed Computing

Multidisciplinary computational simulations require distributed computing technologies to access and utilize distributed resources effectively. Distributed object and component technologies such as CORBA [2], Microsoft COM/DCOM [22, 3], Java RMI [23], Java Beans [24], Enterprise Java Beans [4], and others provide viable options for developing distributed computing applications. These technologies are used extensively in the development of commercial applications by the software industry (these applications are often called enterprise applications).

2.1.1 Microsoft Millennium

The Microsoft Millennium project [25] focused on building the next generation of operating systems, which are widely distributed, highly abstracted, flexible, fault-tolerant, and adaptive. The major goals of this project are as follows: transparent distribution, single-system image, dynamic re-configuration, and heterogeneous integration. The Millennium project addresses meta-computing issues by defining aggressive abstraction and adaptation mechanisms for the application software and the

operating system themselves with a top-down approach to solving the lower layer issues of hardware heterogeneity and distribution.

2.1.2 High Level Architecture (HLA)

The High Level Architecture (HLA) [26] developed by Defense Modeling and Simulation Office (DMSO) is an IEEE standard and part of the Object Management Group (OMG) Distributed Simulation Systems 1.0, which provides a general architecture to support reuse and interoperability of applications developed and maintained by the Department of Defense (DoD). The HLA is realized using the Runtime Infrastructure (RTI) [27] provided by DMSO and several vendors are providing products and services based on this architecture. The main purpose of HLA is to couple simulations together to create a composite simulation.

2.1.3 Common Component Architecture (CCA)

The Common Component Architecture (CCA) [28] is trying to address some of the shortcomings of existing distributed computing techniques by using a component based framework to provide high-performance scientific components while supporting higher level abstractions, reusability, and interoperability between various interdisciplinary applications developed by different groups across different organizations. CCA is comprised of components and frameworks, where components form the software building blocks that are composed together to create applications and different components are created within the framework which provides basic services for different components to use and communicate with one another. CCA provides a set of rules and interfaces for creating components and their interaction with other components. When a CCA component is created it must follow these rules and implement standard component interfaces.

The Scientific Interface Definition Language (SIDL) [28] as part of CCA (similar to CORBA IDL) is used to describe a component and the framework interfaces in a language-independent interface. SIDL provides additional capabilities such as complex numbers, dynamic multi-dimension arrays, multiple interface inheritance, and single implementation inheritance, etc. necessary to support high performance scientific computing. Interaction between different components are described by the CCA ports the communication model. Several Department of Energy (DOE) laboratories and collaborating universities are involved in this effort and several projects based on the CCA are in progress.

2.1.4 CORBA-based Wrapping and Coupling Environment (CWCE)

A CORBA-based distributed environment [29] is developed at the NASA Glenn Research Center to assist migration of legacy FORTRAN applications to client/server architectures by providing an C++ wrapper library that encapsulates details of CORBA and IDL interfaces. The CORBA-based Wrapping and Coupling Environment (CWCE) also supports tighter coupling of distributed legacy applications and provides efficient data exchange between different application components.

2.2 Connecting Distributed Resources

There are many efforts to connect distributed resources shared by multiple organizations in order to provide ubiquitous computing and this is often referred to as a "computational grid [8, 15]." There are several tools and services (commonly known as "grid technologies") developed by academic projects such as Legion [30], Globus [31], UNICORE [32], that support effective utilization of heterogeneous high performance compute resources distributed geographically. This section provides discussion of the various grid technologies and services they provide.

The main goals of these Grid Technologies are as follows:

- create a worldwide virtual computer by sharing compute resources distributed across the globe belonging to different organizations that can be accessed from anywhere;
- provide secure access to these distributed resources while providing security for site administrators as well as users;
- provide single sign-on to all these distributed resources while preserving individual site autonomy;
- hide complexities of the hardware environments heterogeneity and interoperability;
- provide resource management, transparent job scheduling, monitoring, and information services; and,
- support the notion of computer resources as a utility that can be accessed from anywhere.

The Global Grid Forum (GGF) [33], an international organization of researchers, practitioners, and vendors, oversees the development of the underlying technologies, and serves as a standard body in the area of Grid Technologies. The forum addresses a wide variety of issues from security and trust relationships to resource description, discovery, reservation and (co)allocation to debugging, monitoring, and event notification, and many more. Several vendors (*e.g.,* Avaki, IBM, Platform Computing) have started to support these Grid technologies. Computational Grids based on these technologies are feasible now, and many grid prototypes have been already in place sponsored by the National Science Foundation (NSF) (*e.g.,* Grid Physics Network (GriPhyN)), United States Department of Energy (DoE) (*e.g.,* Earth Systems Grid (ESG)), National Aeronautics and Space Administration (NASA) (*e.g.,* Information Power Grid (IPG)), as well as by the European Union (*e.g.,* European Datagrid).

2.2.1 Globus

In this section the most popular implementation of grid technologies, the Globus Toolkit [31], is briefly discussed. Globus Toolkit has been adopted by twelve leading computer and software vendors as the de-facto standard for grid computing. The Globus Toolkit provides software tools and services to enable the development of computational grids in the areas of security, information services, data management, and resource management.

2.2.1.1 Grid Security Infrastructure

The Grid Security Infrastructure (GSI) [34] is used by the Globus Toolkit for authentication and secure communication. The GSI is implemented using public key encryption, X.509 certificates, the secure sockets layer (SSL) communication protocol, and extensions to incorporate single sign-on and delegation. Thus GSI enables secure communication across multiple organizations with a single sign-on using digital certificates.

2.2.1.2 Monitoring and Discovery Service

Information about various grid system components are available through the Monitoring and Discovery Service (MDS) [35]. MDS can be used to store and access various system specific information like architecture type, operating system version, amount of memory, and so on. In addition to this, MDS can be used to discover, publish, and access both static and dynamic information about the status of different resources in the computational grid. MDS uses the Lightweight Directory Access Protocol (LDAP) [36] to access such information about the different grid components and provides a unified view of the disparate grid resources.

2.2.1.3 Globus Resource Allocation Manager

The Globus Toolkit provides the Globus Resource Allocation Manager (GRAM) [37] for allocation and management of resources on the computational grid using a resource specification language (RSL) [37] to request resources. In addition to resource allocation and management the GRAM also updates the MDS with information about the current availability of computational resources. The GRAM API can be used to submit a job, query the status of a job, and cancel a job. A GRAM service will be running on each computer that is part of the computational grid and is responsible for interfacing with the local site-specific resource management system like PBS [38] or Condor [39].

2.2.1.4 Data Management Services

Data can be accessed and managed using the data management services of the Globus Toolkit. Earlier versions of the Globus Toolkit used the Globus Access to Secondary Storage (GASS) [40] service to access secondary storage on a remote host in the computational grid. Newer versions of the Globus Toolkit use GridFTP [41] - a secure, high-performance, and robust data transfer mechanism to accessing remote data. In addition to GridFTP, Globus Toolkit provides Globus Replica Catalog to maintain a catalog of dataset replicas so that instead of duplicating large datasets only necessary pieces of the datasets are stored on local hosts and the Replica Catalog will keep track of replicated files [9]. The Globus Replica Management software provides the replica management capabilities for data grid by integrating the Replica Catalog and the GridFTP.

2.2.1.5 Commodity Grid Kits

Globus services described above are system level services that can be accessed through a set of client-side command-line utilities. There is no direct support at the application level for application developers to exploit the benefits of Grid computing

using traditional distributed computing techniques. To enable the development of applications to utilize grid services effectively and integrate grid services into existing application development platforms the Commodity Grid Kit (CoG) [16] is provided as part of the Globus Toolkit. CoG provides client-side APIs for Grid services through language bindings and implementations in Java [42], Python [43], CORBA [44], and others. CoG implementations communicate with the Grid services executing on the server similar to command-line clients. CoG provides a mapping between Grid services and the higher level frameworks and environments developed on top of the Grid services. The CoG API consists of mappings to the low-level grid services (*e.g.,* GSI, GRAM, MDS) and low-level utility components (*e.g.,* RSL).

2.2.1.6 MyProxy

Globus Toolkit uses GSI-based digital certificates to access distributed computational resources. To ensure security typically these certificates are stored as files on a file system and this file system is accessed through a secure manner (for example, using *ssh* [45]). Instead of using these certificates directly a proxy certificate with a short lifetime is normally created and this proxy certificate is used to access grid resources. If the user is away from their primary system or logged in from a system that does not have grid clients to create a proxy certificate then user will not be able to access the required computational resource. In order to allow users to access the distributed resources from any network connected computer without requiring access to user's long-term (permanent) credentials the MyProxy online credential repository [46] provide a credentials repository to store and retrieve proxy credentials. Using the MyProxy server users can access their credentials from anywhere without depending on any grid software to create proxy certificates and without accessing their permanent certificates. MyProxy server also enables browser based Grid Portals to access user credentials and access the grid resources on users behalf. MyProxy provides command-line tools as well as Java

and Perl APIs to store and retrieve credentials from the credential repository. Typically proxy credentials are encrypted using a passphrase and stored by the MyProxy server and users must authenticate with the server to retrieve credentials.

2.2.2 Legion

Legion [30] introduced the concepts of meta-computing and virtual computers, which reflect the ideas of presenting a logical view of a single computer out of many computers, which are potentially heterogeneous, physically distributed, connected with heterogeneous networks, and logically belonging to multiple administrative domains.

Legion supports applications written in the Mentat Programming Language (MPL) [47] (a parallel version of C++), Fortran, and Java as well as legacy applications through object wrappers. Legion provides a set of tools that can be used to authenticate and access distributed resources, register applications, submit registered applications for execution on remote hosts, transfer files, and so on. Legion is used in the commercial implementation of Avaki's Grid Software.

While both Legion and Globus systems aim to achieve similar goals and objectives they differ in both their architecture and implementation. While the Globus Toolkit is based on a bag of services architecture, the Legion project uses an object-oriented metasystem where each component is an object. Unlike the layered architecture of Globus, Legion can be viewed as a "Virtual Operating System" that sits on top of local operating systems and provides both the low-level services and high-level abstractions for end-users. Globus Toolkit can be used to implement the low-level services provided by the Legion core objects.

2.2.3 Open Grid Services Architecture

The emerging Open Grid Services Architecture (OGSA) [48] seeks to provide an interface based on web services for managing grid service instances. The OGSA

specification provides the interfaces and behaviors of a *grid service* and the Open Grid Services Infrastructure (OGSI) [49] provides the corresponding implementation for these interfaces. OGSA provides a standard mechanism for creating, naming, registering, and discovering instances of grid services as well as bindings for multiple protocols and implementations. OGSA specification is defined by WSDL [50] interfaces and the grid services can be delivered through multiple hosting environments such as J2EE [5] and .NET [6].

2.2.4 Other Related Work

Other client-server applications such as Netsolve [51] and Network based Information Library (Ninf) [52] provide easy-to-use programming interfaces and mechanisms for accessing distributed computational resources. Users can create distributed applications by calling the appropriate optimized remote libraries supported by these systems. All the aspects of distributed computing and parallel computing are hidden from the user by the Netsolve agent or the Ninf metaserver. The main difference between these two systems is that Ninf uses Ninf IDL whereas Netsolve provides interactive interfaces (MATLAB [10] interface and shell interface) and programming interfaces for C and FORTRAN. Unlike Globus Toolkit, Netsolve and Ninf provide application level support through APIs to access distributed software libraries.

2.3 Efficient Access to Large Scale Datasets

As mentioned in Chapter I, large data collections are emerging as important resources for both industrial and scientific users. In domains as diverse as weather, climate and ocean modeling, high energy physics, astronomy, and computational genomics, the volume of interesting data is already measured in terabytes and will soon total petabytes of storage. The combination of large dataset size, geographic distribution of users and resources, and computationally intensive analysis results in complex and

stringent performance demands. Efficient and reliable queries (data discovery, selection and access) require advanced techniques that collectively maximize use of scarce storage, networking, and computing resources. In this section various efforts underway to address these issues are described.

2.3.1 Storage Resource Broker

Storage Resource Broker (SRB) [53] is client-server middleware that provides a uniform interface for connecting to heterogeneous data resources over a network and accessing distributed data by organizing data as logical collections. SRB uses the metadata catalog to provide location transparency by accessing data based on attributes instead of file names or physical locations. SRB also provides capabilities to store replicas of data, access replicated data efficiently, user authentication, access control of datasets, auditing, and fault tolerance.

2.3.2 DataCutter

DataCutter [54] provides a framework for exploring and analyzing scientific datasets in the Grid environment. DataCutter uses the filter-stream programming model where logical units of computation (filters) are applied on remote data and data is exchanged between different filters using unidirectional buffer pipes (streams) of fixed size. DataCutter supports data filtering with user-defined operations, sub-setting, spatial query and indexing, and data aggregation using filter groups. Using the services provided by the DataCutter framework, application developers can create filters, specify the connectivity between different filters, and integrate these components within an application. DataCutter is already integrated with SRB and there are plans to integrate with Globus. Currently DataCutter is deployed at National Partnership for Advanced Computational Infrastructure (NPACI) as part of the NSF funded distributed teragrid facility (DTF).

2.3.3 Data Grid

While Computational Grid seek to provide "virtualization" of computational resources, Data Grid attempts to provide "virtualization" of data access for data distributed among different administrative domains, storage systems, and data access environments [9]. Data Grid is designed as a specialization and extension of the Computational Grid. Since applications must frequently operate in wide-area, multi-institutional, heterogeneous environments, the Data Grid architecture is designed to be independent of the low-level mechanisms used to query, store and transfer data, and to be integrated with the Computational Grid infrastructure that provides basic services as authentication, resource management, and information.

The two basic services provided by the Data Grid are data access and metadata access [9]. The data access service provides mechanisms for accessing, managing, and initiating third-party transfers of data stored in storage systems. The metadata access service provides mechanisms for accessing and managing information about data stored in storage systems, as well as the Data Grid itself. A basic Data Grid component is a logical storage system that represents any physical storage system implemented by any storage technology that can support the required access functions, such as file systems, database management systems, web servers, and hierarchical storage systems. The storage system provides functions for creating, destroying, reading, writing, and manipulating file instances. The file instance is the basic unit of information in a storage system, and may reside in a file system, database or other storage system. In other words, the Data Grid allows for defining units of information in a platform, storage technology, and physical location independent way. The Data Grid introduces an integrating architecture, building on top of, and complementing existing elements of the infrastructure such as SRB.

2.3.4 Data Warehouses and On-Line Analytical Processing

Section 1.2 described how large scale scientific and engineering are organized and managed. However, data required for enterprise applications are typically stored and managed in large relational databases often referred to as Data warehouses. For example, data warehouses store information that can be used to answer "who?" and "what?" questions about past actions. On-Line Analytical Processing (OLAP) systems [55], on the other hand, provide the ability to answer "what if?" and "why?" questions (in addition to answering the "who?" and "what?" questions) to support decision-making about future events. OLAP transforms data stored in data warehouses into strategic information by using a multidimensional view of aggregate data. OLAP supports simple navigation and browsing ("slice and dice"), simple computations, advanced analysis, and complex modeling. The main goal of OLAP systems is to transform raw data to information and then to knowledge that can support effective decision-making for different types of users. OLAP systems generate "just-in-time" information by computing data using complex relationships on the fly instead of just returning a piece of raw data.

2.4 Internet Computing

The wide use of the Internet has seen the development of new technologies and services over the last few years. These technologies change rapidly and newer technologies are introduced every month. With the introduction of the Web browser in the early 90's, static web pages started to pop up everywhere, soon their functionality was extended to provide some processing capability by the use of Common Gateway Interface (CGI) and Perl scripts. Even though several websites still use CGI/Perl, they have several drawbacks. CGI/Perl based processing is platform specific and is not scalable since a process is created for each invocation of the script. Also these sites are difficult to

maintain since content and presentation logic are embedded in the same script. Since the basic Hypertext Transfer Protocol (HTTP) [56] does not provide any sessions and each invocation of the CGI/Perl script invokes a new process, user sessions must be managed manually.

2.4.1 Servlets and Java Server Pages

Servlets [57], Java Server Pages (JSPs) [58], and Active Server Pages (ASPs) [59] support dynamic content generation and customization of web pages. Servlets provide a platform-independent and Web-server-independent server-side-processing capability in Java and there is no need for the browser to support Java (often referred to as server-side applets or *servlets*). Servlets can be invoked several times once loaded in memory and there is no need to create a separate process for each invocation (unlike CGI/Perl scripts), thus servlets also offer a scalable solution. Furthermore, user requests can be easily accessed through a request object and state information can be maintained in a session object provided by the servlets. While content generation and display are combined with a servlet, the JSPs separates content development and presentation and promotes code reuse through a component-based architecture. JSPs use JavaBeans [24] and customized JSP tags [58] to generate the content and provides special tags to insert content in a HTML [60] document.

2.4.2 Web Portals

The abovementioned technologies have changed the static nature of web pages and bought about a new era of dynamic content generation with separation of presentation and content. As a result of which many Web portals such as Yahoo!, MSN, and Excite were enabled and have gained popularity. These user-customizable portals provide convenient access to a number of Web-based services, including e-mail, weather forecast, stock quotes, shopping online, news, and many others. Web Portals act as a gateway to

the vast resources available on the World Wide Web or Internet. The important features of a Web Portal are accessibility, customization, and dynamic content generation.

Web Portals are used to provide online services for almost every aspect of everyday life including banking services, travel reservations, stock trading, auction, shopping, and so on. Users authenticate with these portals, enter some data, then click submit, appropriate action is initiated at the server, and some information is returned back to the user. The user is unaware of where the server is physically located, what action was performed, how it was performed, and what technology was used. Thus portals hide the various complexities involved in performing required transaction from the end user transparently by providing simple and easy to use interfaces. It is the simplicity of the user interface, which hides all implementation details from the user, that has contributed to the unprecedented success of web portals.

2.4.3 Java 2 Enterprise Edition (J2EE)

The introduction of the Java 2 Enterprise Edition (J2EE) architecture by Sun Microsystems [5] and the .NET initiative by Microsoft [6] have provided a new wave of capabilities for building robust and scalable enterprise applications. These technologies have changed Web applications from the simple client-server model to a multi-tier architecture. The J2EE platform provides an architecture for developing, deploying, and maintaining highly available and reliable enterprise applications. The J2EE platform provides runtime environments for development and deployment of enterprise applications, called containers. The containers provide security, transactions, multi-threading, component reusability, distributed deployment and clustering, connection resource pooling, interoperability with other enterprise applications (including legacy applications), and so on. The various containers are as follows:

- Enterprise Java Beans (EJB) Container – Supports EJB components used to implement business logic, interfacing with legacy systems, accessing services

provided by third parties, and generation of content. The EJB container also provides persistence, transaction management, security, life cycle management, and database connection pooling [4].

- Web Container – Supports JSP and servlet-based web applications that implement the client interaction, presentation, and control logic for an enterprise application.
- Application-client Container – Provides the standard Java 2 development environment to support J2EE client applications.
- Applet Container – Standard runtime environment for Applets typically embedded in Web Browsers.

Figure 2.1 illustrates a typical application of the J2EE architecture. The different front-end clients form the client tier, JSP-based web-tier and the EJB-based application-tier form the middle-tier, while the legacy applications and databases form the back-end tier. The J2EE platform provides several benefits when compared with generic web-centric applications. Some of the key benefits are as follows [5]:

- provides a simplifies architecture for rapid development and deployment of distributed applications using the "Write Once, Run Anywhere" programming model while supporting separation of concerns between business logic, content generation, formatting, and display;
- supports multiple application scenarios including stand-alone clients, web-centric applications, multi-tier applications, and business-to-business applications;
- provides a scalable solutions since the containers can be distributed and configured to perform automatic load balancing based on application demands;
- provides a security model that can be customized during deployment by establishing a mapping between users with roles;
- allows integration with legacy applications by supporting several industry standard APIs (like Java Database Connectivity (JDBC) [61], Java Naming Directory Interface (JNDI) [5], and Java Message Service (JMS) [5]); and,

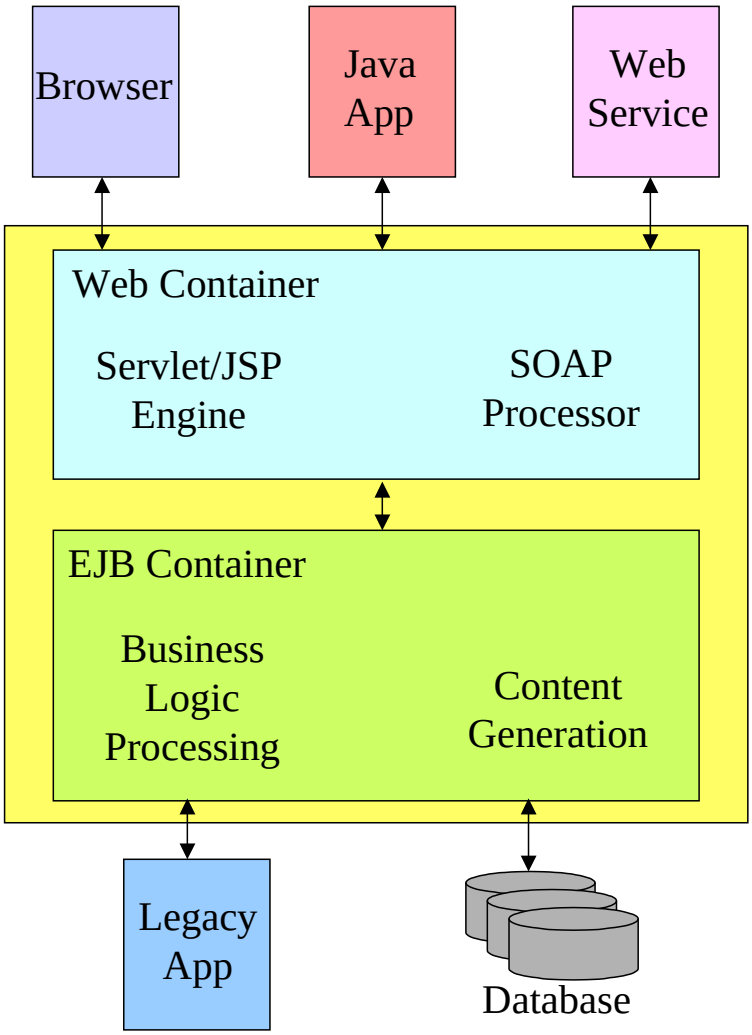


Figure 2.1: A typical application of the J2EE architecture

- a large number of J2EE complaint servers and tools are available to support development and deployment of distributed applications.

2.4.4 XML and SOAP

The need to interoperate and exchange information between businesses has resulted in the use of the eXtensible Markup Language (XML) [62] and the Simple Object Access Protocol (SOAP) [63], which uses XML over HTTP as a means of bypassing firewalls and access remote objects.

XML provides a set of rules for creating semantic tags used to describe data and does not specify any tag set or grammar. XML is a meta-language that can be used to define a language and provides a flexible means for electronic data exchange and delivery over the Internet. HTML is presentation-specific, instead of content-specific, and provides limited structure, reuse, interchange, and automation. XML does not replace HTML, but rather provides a portable and platform-independent tool for data storage, data exchange, and data sharing.

SOAP is an XML-based lightweight protocol for communication via Internet between applications using a platform-independent and language-independent protocol. SOAP is "simple" and "extensible" and solves the security problem of getting messages across firewalls and proxy servers. A typical SOAP message consists of an envelope that identifies the document as a SOAP message, header information, the message body that provides request and response information, and a fault element to process errors that occur while processing the message.

2.4.5 Jini and JavaSpaces

The Jini Network Technology [64] from SUN Microsystems supports creation of instantaneous, spontaneous networking between any two applications irrespective of their software and hardware. Services can be added and removed without any intervention and

these services can be software as well as devices. The Jini Technology supports discovery and lookup, leasing, and distributed transactions. JavaSpaces [65] is a Jini service that facilitates distributed application development based on the Linda coordination language [66]. Unlike traditional distributed computing techniques JavaSpaces use a space-based distributed computing model where applications coordinate with each other through the flow of objects through *spaces* instead of passing messages between two processes or performing a remote method invocation. JavaSpaces provide a simple programming model to develop client-server applications since only client programs need to be developed while common server program functionality is provided by the spaces for free.

2.4.6 Peer-to-Peer (P2P) Systems

Peer-to-peer (P2P) computing [67] is a form of distributed computing that enables different computer systems to directly interact with one another without the need for a centralized server. P2P is based on a decentralized distributed computing architecture where each entity in the network, called a peer, has equal status (a peer can act as a client or a server based on the context of an operation). These peers can be powerful servers as well as simple pagers or cell phones, it is not necessary for all peers to have equal capabilities and these peers can join the network anytime they choose and leave the network anytime as well. Thus a P2P system is decentralized, dynamic, and unpredictable system, where entities having varying capabilities are treated as equals and these entities can communicate with other entities without requiring a central server. These P2P systems are trying to put together the enormous amount of untapped resources distributed across the globe to support large scale computing and collaboration.

P2P systems differ from traditional client-server systems in many ways. A client-server system is based on a centralized architecture. Typically, clients request services and the servers provide the services. Whereas in a P2P system that is based on a

decentralized architecture, clients and servers have equal status and a peer can request a service or provide a service based on the context of operation. Because of the centralized nature of the client-server architecture, if the number of clients increase then the load on the server increases and the server becomes the central point of failure. If the server fails then clients cannot function on their own. In the case of a P2P system when more peers are added, the system becomes more powerful in its capabilities since the addition of more peers also means more resources are available. Thus a P2P system appears more scalable then a client-server system. A P2P system does not fail if one of the peer decides to leave or is shutdown or fails. As long as there is one peer alive a P2P system is operational at some level.

In the case of a P2P system, there is no need for a central registry to locate a service, other peers can be discovered either using the network model or the multicast model [67]. Using the network model a peer uses other peers in the network to locate another peer, there is no central registry. With the multicast model to locate a peer an IP multicast is used to send a message to all peers and the corresponding peer would extract the message and establish a TCP/IP connection with the requesting peer.

There are certain disadvantages to the P2P system as well. Since a client-server system is centralized it is easy to manage, administer, and enforce policies. But managing a P2P system is difficult since peers come and go and there is no central authority to regulate the actions of a peer. To aid the development of P2P systems there are no prevailing standards and there are different protocols, architectures, and implementations available whereas client-server architectures have well-accepted standards.

2.4.7 Web Services

Web-based applications require users to interact with the browser to perform any operation. A web service is trying to replace the human interaction with the web-based application so that another program can interact with the web-based application on

user's behalf. A web service sends a XML request and the web-application returns a XML response, the corresponding XML scheme defines what to do with the XML response. This allows different business applications to encapsulate the implementation details and provide a standard interface through which they can talk to other related applications without exposing the inner workings of the application.

A typical Web Service consists of three actors [7]: service provider, service registry, and service requestor and the operations that are defined on these actors are as follows: publish, find, and bind. A service provider deploys a service and provides a description of the service using the Web Services Definition Language (WSDL) [50] and publishes it to the service registry, normally the Universal Description, Discovery, and Integration Service (UDDI) [68], using the publish operation. The service registry is similar to yellow pages that contain all the published web services. Service providers can publish a service and service requestor can look for a particular service. A service requestor uses the find operation to locate and retrieve a service description and then uses the bind operation to bind the service description with the service provider and interact with the web service implementation.

2.5 Problem Solving Environments

Chapter I described that a problem solving environment (PSE) is required to assist the creation of complex computational simulations and described the various requirements of such a PSE. In this section some of the PSEs that address these requirements are briefly described.

2.5.1 Visual Programming Systems

Visual programming systems like Khoros [11] and Advanced Visual Systems (AVS) [12] provide graphical user interfaces to define icons representing application components and to connect these components directly or using control structures to

create data-flow models. Users can compose complex applications using graphical representations, save the compositions, execute the application, and reload saved compositions. Such visual systems are commonly used to create imaging and visualization systems where the individual components are predefined and provide common functionality required to create complete applications. The advantage of such systems is that users can create an application pictorially using existing components without writing new code. Such systems are easy to learn and use and also promotes reuse of predefined components.

2.5.2 // ELLPACK

// ELLPACK [13] is a problem solving and development environment for solving partial differential equation (PDE) based applications. // ELLPACK provides a GUI for defining the PDE problem, selecting a solution method, and analyzing the solution. The solution to the PDE problem is obtained on HPC platforms and results are returned to the GUI for further visualization and analysis. There is also a web based version known as Web // ELLPACK [69] that provides access to the PSE through a web browser.

2.5.3 Symphony

Symphony [70] provides a component-based framework for creating, saving, sharing, and executing meta-programs. Meta-programs are composed from existing applications by defining the dependencies using directed acyclic graphs. The front-end provides a graphical user interface to describe stand-alone applications and connect these applications. The back-end can be a grid-enabled resource where the applications are to be executed. This framework is suitable for creating complex meta-programs from legacy applications and does not require any changes to existing applications. Symphony supports incorporation of local application components like visualization tools into the meta-program. Also, Symphony is designed in such a way that it can be directly layered

on top of any grid infrastructure and does not depend on any specific server to be in place.

2.5.4 Computational Portals

The successful use of web portals to access vast amount of resources on the Internet has prompted researchers to develop analogous portals for addressing the requirements of computational simulations. Similar to a web portal acting as a gateway to the web, computational portals act as a gateway to distributed, heterogeneous HPC resources. These resources include not only computer hardware but also system software, third-party applications, user-created applications, and data in different form and formats. This transition of PSEs from standalone or client-server applications to three-tier, web-based computational portals is illustrated in figure 1.3. Numerous computational portals and problem solving environments have been successfully deployed and some of the popular ones are as follows:

- ASCI Grid Services [71],
- DISCOVER [72],
- Extensible Computational Chemistry Environment (ECCE) [73],
- Gateway Computational Web Portal [74],
- Grid Resource Broker Portal [75],
- Lattice Portal [76],
- Legion Grid Portal [77],
- Mississippi Computational Web Portal [19],
- NPACI Hotpages [78],
- Polder Computing Environment [79],
- SciRUN [80],
- TENT [81],
- UNICORE [32], and

- XCAT Science Portal [82].

There are many similarities in the functionality provided by the abovementioned portals, but no two portals are alike and each portal differ from one another either in terms of the methodology used to implement the portal and functionality provided to support different portal users. These Portals are classified as domain-specific application portals or PSEs and generic user portals [14]. Each portal provides a common set of functionality, as follows:

- provide secure access to distributed compute resources;
- provide details about the hardware use and utilization;
  - general information (processor type, number of processors, memory, network fabric);
  - Queuing-related (how many processors available, how many jobs in the queue are running or pending);
- provide a mechanism to access system software, third-party applications, user-created applications;
- provide means to describe an user-defined application, stage the various phases of the application, submit the application for execution on the HPC hardware, and show status of jobs running on the chosen hardware;
- provide access to the end-results (plots, visualization results) of the execution from the HPC platform to the user desktop;

Almost all of the above listed portals use the Commodity Grid (CoG) Kit in one form or other as the building block. Some of the portals [19, 32, 71, 74, 75, 76, 77, 78] support low-level services to securely access grid resources while some of portals [72, 73, 79, 80, 81, 82] support high-level features like computational steering, data manipulation, and visualization.

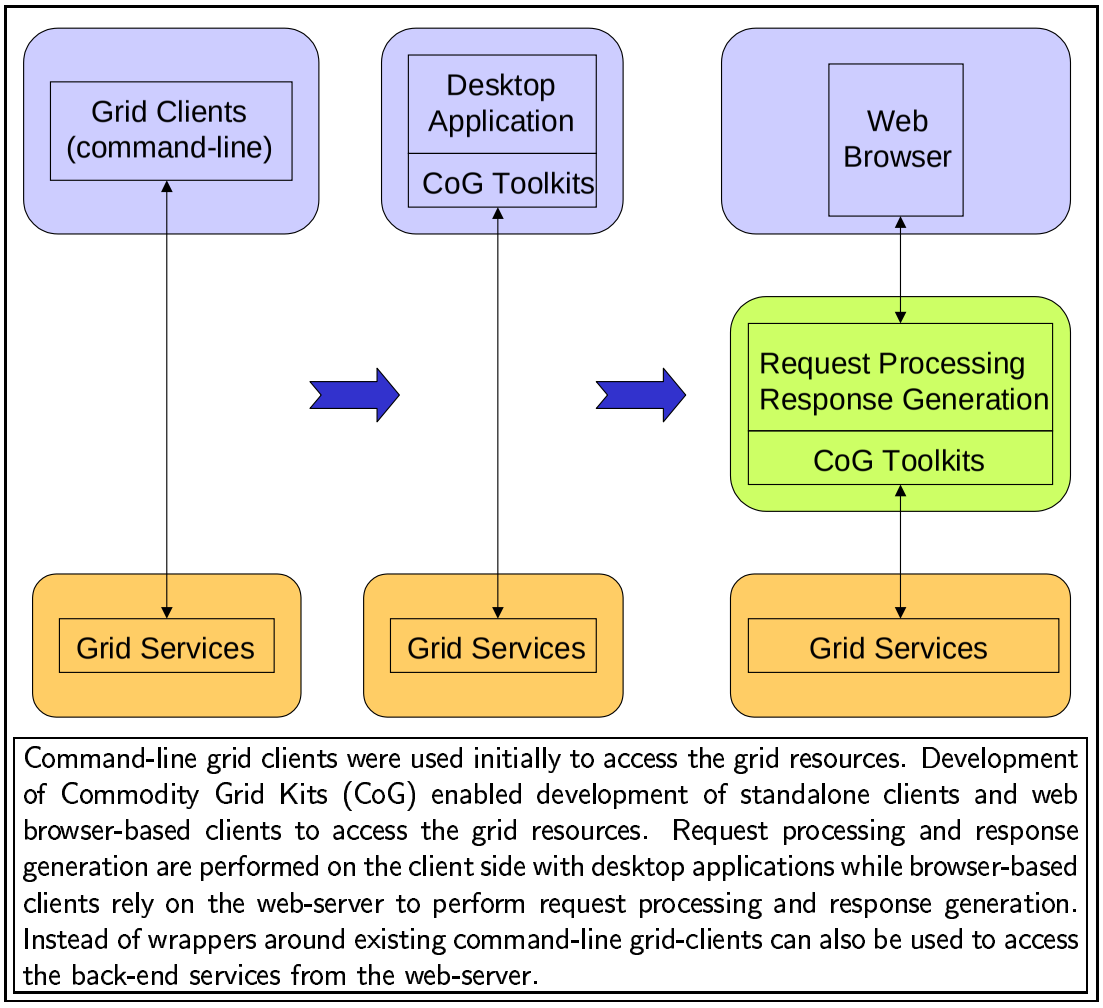


Figure 2.2: Illustration of evolution of grid computing environments

2.6 Grid Computing Environments and Frameworks

Grid Computing Environments (GCE) essentially a set of tools and technologies that provide user-friendly interfaces to access various grid resources and services while hiding the complexities of the Grid. An extensive list of grid computing environments can be found in [83]. The evolution of GCEs is illustrated in figure 2.2. Grid clients provide command-line clients to access the grid services. Commodity Grid toolkits (CoG) provide a programmatic interface to standalone clients written in programming languages such as Java and Python. To provide easy-to-use interfaces and make these clients accessible from a browser web-centric applications use CGI/Perl scripts to invoke grid clients or used CoG APIs to access the grid services directly from the web server. Typically a GCE is deployed as a computational portal built on a multi-tier architecture with simple user interfaces as shown in figure 2.3. Each GCE differs not just in the technology used to implement them but also in terms of the capabilities provided. This section discusses briefly some of the generic frameworks that can be used to develop domain specific application portals or generic user portals.

2.6.1 WebFlow

WebFlow [84] provides a framework for web based metacomputing by incorporating a distributed object-based, scalable, and reusable web server and object broker middleware as part of a three-tier architecture. The top tier provides a user-friendly visual programming tool and the bottom tier consists of distributed grid resources. The WebFlow middle tier provides the runtime environment that translates user requests to corresponding actions at the back-end. JavaBean components are wrapped on legacy code and these components are used to create a complex application using a commodity visual authoring tool. The web server in the middle tier is used to receive user requests and provide content for presentation to the front-end. CORBA RPC is used to communicate between the WebFlow server and the back-end resources. WebFlow

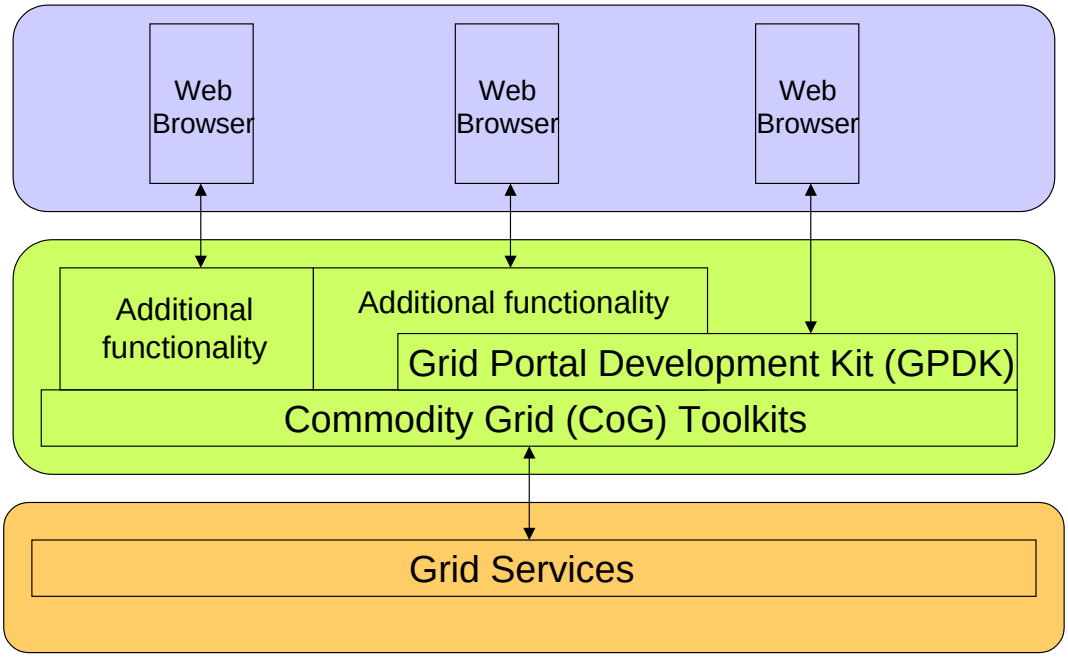


Figure 2.3: Typical three-tier architecture for computational portals

modules are CORBA objects implemented using Java beans. Instead of Java RMI, CORBA is used so that objects written in languages other than Java can be included. Gateway [74] is an example of a portal developed using WebFlow.

2.6.2 Grid Portal ToolKit (GridPort)

The Grid Portal Toolkit [85] provides a framework to develop web portals for computational grids. GridPort provides secure access to HPC resources and supports job submission, file manipulation, and account management. GridPort uses HTML to deliver the content and uses CGI/Perl to perform all the processing and generate the content for presentation. GridPort supports the notion of three different types of Portals - User, Service, and Application Portals. The service portal provides basic services like authentication, job submission, and account management. Application portals use the service portal services to develop application specific interfaces while the user

portal provides information and interactive services. The NPACI Genie Portal (formerly HotPages) [78] is one of the popular user portals developed using GridPort.

2.6.3 Grid Portal Development Kit (GPDK)

The Grid Portal Development Kit (GPDK) [86] is a framework that provides secure methods for submitting jobs, file transfer, and access to grid information resources. GPDK provides modular, reusable components for accessing common Grid services as well as providing a sample template portal that can be easily modified or extended to support additional services and create customized problem solving environments. GPDK uses Java Server Pages (JSP) and Java Beans to create a web portal that can run on any commercial web server, and provides authenticated connectivity to the grid using a web-based interface that allows customized portals development by end users. The GPDK has been used in several portal building projects such as Astrophysics Simulation Collaboratory (ASC) [87] and NASA IPG Launchpad User Portal [88].

2.6.4 Jini-based Portal Augmenting Grids

Jini-based Portal Augmenting Grids (JiPANG) [89] is a portal toolkit built on top of Jini technology to provide a uniform interface layer for different grid systems. JiPANG provides Java APIs and a browser to interact with grid systems such as Globus, Netsolve, and Ninf using a uniform high-level interface. To access the grid services using JiPANG there is no need to install any client programs. JiPANG also eliminates the need to maintain and update client programs in the dynamic grid environment using the Jini technology. JiPANG toolkit consists of three components: the service toolkit, the client toolkit, and a browser. A grid service can be registered using the service toolkit and client toolkit provides Java class libraries to access the grid services registered with JiPANG system. Instead of directly exposing the client toolkit APIs to the end-user a front-end application for the specific grid service can also be developed. The browser provides a

GUI to the JiPANG system and the browser is also registered as a grid service. Thus, users can launch this browser without downloading any software directly from a web browser.

2.6.5 Uniform Interface to Computing Resources

The main goal of Uniform Interface to Computing Resources (UNICORE) [32] is to provide secure access to distributed computing resources by hiding the system- and site-specific conventions of these systems from the user. UNICORE provides a PSE that supports job creation, submission, monitoring, and data transfer between local and HPC systems as well as data transfer between two HPC systems. UNICORE uses a three-tier architecture, the client-tier provides graphical user interfaces, the UNICORE gateway constitutes the second-tier, and the system specific UNICORE servers form the third-tier. The UNICORE client provides graphical user interfaces to create, submit, and control jobs from any desktop connected to the Internet. The client connects to a UNICORE gateway, which provide authentication and secure communication between the client and the gateway. The UNICORE server provides the system with specific Grid services by mapping the generic user requests to system and site specific resources.

2.7 Summary

This chapter provided a brief discussion on different technologies and approaches found in the literature to address some of the requirements of a DCSS presented in section 1.2. While traditional distributed computing techniques could be used to develop distributed applications, they are not widely accepted and used by the scientific and engineering applications developers since they do not address the high-performance requirements of these applications (especially support for efficient communication and suitable abstractions for high performance computing) and often require extensive modification to the existing applications (these applications are validated and used in

production mode). Most computational simulations are written in FORTRAN and must be retrofitted with object wrappers to create distributed applications using CORBA. Since there is no CORBA IDL for FORTRAN this would require manually writing C++ wrappers to create corresponding CORBA objects. The CWCE approach described in section 2.1.4 assists such migration by providing a C++ wrapper library that reduces the effort required to learn CORBA and manually generate C++ object wrappers. However, this approach still requires re-engineering legacy FORTRAN applications, including CWCE calls to wrap the application, and linking the CORBA/C++ library. Furthermore, CORBA services are not available on most of the traditional HPC platforms where computational simulations are typically executed. Thus, the majority of scientific and engineering applications that require distributed computing rely heavily on user intervention and hand-coded scripts to manage all the issues and complexities of distributed computing. Such approaches are ad-hoc and not based on any standard practices and do not make effective utilization of available distributed computing techniques. Hence, traditional distributed computing techniques are not suitable for connecting standalone computational simulations at the application level.

Internet technologies such as J2EE provide a multi-tier architecture for developing applications for the enterprise while technologies like web services provide a language and platform independent approach to connecting diverse applications and exchanging information. P2P systems provide an alternative to the traditional client-server model by allowing clients to communicate with one another without requiring a central server. These technologies can be used to deliver the services provided by DCSS.

Computational and data grids provide infrastructure that connects disparate computational resources, data warehouses, and instruments distributed across multiple administrative domains to create a virtual computer that can be accessed on demand. These technologies are currently accessible as system-level services and there is no direct application-level support to use these services. GCEs are addressing this issue by

providing high-level abstractions to the application developer to hide the complexities of the grid. GCEs are implemented as computational portals or desktop applications and do not support all the different types of users shown in diagram 1.1. Also interoperability between two different GCEs is not addressed by any of the GCEs.

CHAPTER III

ARCHITECTURE OF A DISTRIBUTED COMPUTATIONAL SIMULATION SYSTEM

It is evident that there is a need for a distributed computing simulation environment (DCSE) suitable for users of multi-disciplinary computational simulations to connect disparate applications, datasets, products, resources, and users into a single user-friendly system. Such a distributed computational simulation system (DCSS) must hide the complexities of distributed computing from the scientists and engineers and thereby improve user productivity. Users interact with such a DCSS by submitting requests and the distributed system processes the requests, performs the necessary action on behalf of the users, and then returns the corresponding response or result. Thus, all interactions between users, computational resources, applications, datasets, products, and so on are performed through the distributed system. Also since, each environment requires a different set of resources, applications, datasets, and products it is prudent to create a domain-specific environment instead of creating one distributed environment for all scientific and engineering problems. How these requirements are addressed by other systems was discussed in Chapter II and although some systems address some of these problems separately, no systems, of which the author is aware, solves the key problems jointly. In order to satisfy all the goals and requirements of a modern DCSE described in Chapter I a combination of several technologies is required. In this chapter a unique architecture to address these requirements by combining Grid, Internet, and Distributed Object/Component technologies is presented. This is the only architecture, the author is

aware of, that provides a framework to support such a comprehensive set of requirements as well as support different types of users.

The common features and capabilities required by a DCSS are extracted and a components-based *open framework*, called the Enterprise Computational Services (ECS) framework, that can be reused by different domain specific DCSE is developed. An *open framework* encapsulates changes made to the framework from the users of the distributed environment and the framework can evolve with the introduction of new technologies and take advantage of such technologies by incorporating them without affecting the overall distributed computing environment. Furthermore, the constituent components of the DCSS can be developed independently, possibly using different languages and programming techniques and be instantiated on different computing platforms. In Chapter II several complementary technologies like Grid Computing, Internet Computing, and Distributed Computing available to develop high performance DCSS were discussed. The distributed system that is presented here is based on a framework developed using a unique combination of these technologies. The ECS framework presented here utilizes three key enabling technologies: Grid, Internet, and Distributed computing to connect standalone software components or applications into a single user-friendly system while reducing the complexities of multi-site, multi-program, multi-resource, multi-organizational computing. Internet technologies are used to provide a user-friendly front-end, a problem solving environment (PSE), and discover and lookup service providers and services, while Grid technologies form the basis for accessing geographically distributed resources (this includes hardware, software, and data) securely. The distributed object/component technologies are used to develop a reusable distributed middleware, called Enterprise Computational Services (ECS), that provides high-level user services, metadata repositories, persistence, logging, and history.

3.1 Architecture

In order to support the development of domain-specific DCSE an open architecture based on the Java 2 Enterprise Edition (J2EE) [5] multi-tier architecture and Web Services [7] is used as the approach to proving the hypothesis. This architecture (illustrated in figure 3.1) is logically divided into three tiers: front-end, middle-tier, and back-end [5]. Geographically distributed, heterogeneous, HPC resources (hardware, software, and data) form the back-end tier. The front-end provides easy-to-use graphical user interfaces and programmatic interfaces to access the back-end resources. A reusable components-based middle-tier provides a common service layer to support these multiple front-end clients and diverse back-end resources. Some of the front-end clients can communicate directly with the back-end while other clients require translations or mappings. The middle-tier performs all the necessary translation of user requests to appropriate actions in the middle-tier in addition to providing a set of unique services not supported by existing transient Grid services. Each front-end client usually uses a different protocol to communicate with the middle-tier and similarly the middle-tier uses a different protocol to access different types of back-end resources. The multi-tier architecture encapsulates distributed computational resources and applications in the back-end and hides the implementation details of the middle-tier from the user by providing user-friendly front-end interfaces.

3.1.1 Front-end

The front-end tier accepts user requests, forwards the requests to appropriate services in the middle-tier, and displays the corresponding responses to user requests based on the results provided by the middle-tier. User requests are typically accepted through wizard-based Graphical User Interfaces (GUIs) and these interfaces hide all the complexities involved in processing the requests, performing necessary operations to satisfy the requests, and generating the desired results while automating some of the tasks that

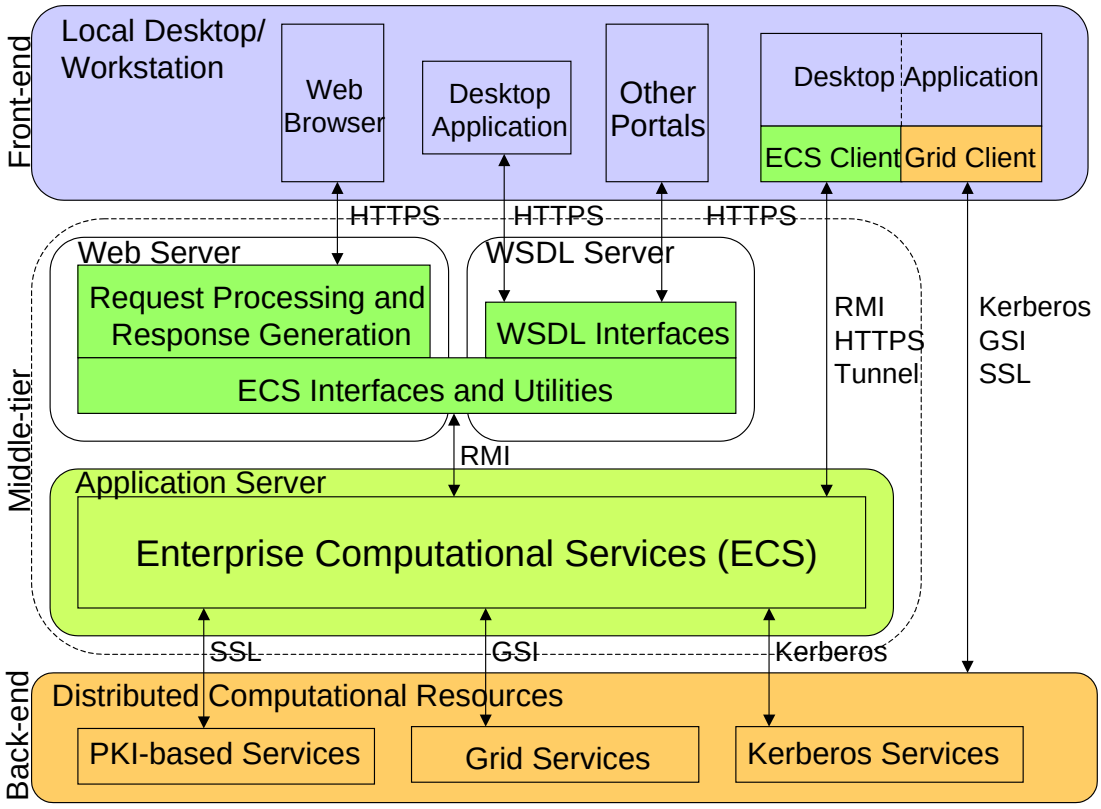


Figure 3.1: Overall architecture of a DCSE based on the ECS framework

require human intervention. Furthermore, these user-friendly interfaces help users to learn the system quickly and train new users without encumbering the advanced user. To create a domain specific DCSS only the front-end needs to be customized for that specific environment. The front-end client can be a web browser, standalone application (written in Java, C++, or Visual Basic), or a .NET application. Section 3.3 describes how the architecture described above can accommodate multiple clients appropriate for different types of users within a single distributed computational environment.

3.1.2 Middle-tier

The middle-tier is responsible for processing the user requests forwarded from the front-end, performing necessary operations including accessing back-end resources and middle-tier services, and then creating and formatting the content for presentation by the front-end clients. The middle-tier functionality can be further divided into two tiers: web-tier and application-tier. The web-tier is responsible for processing user requests and formatting content for presentation or display while the application-tier provides various services (business logic) used to perform operations requested by the user and generates content for the web-tier. The application-tier also acts as an Object Request Broker (ORB) for applications developed using CORBA or CORBA Component Model (CCM) [2]. A web-tier can be considered to be a client of the application-tier and it is involved in translating and mapping user requests to corresponding method invocations on application-tier objects.

The application-tier is a container for all the services provided by the middle-tier denoted here as ECS. The various services provided by ECS framework can be classified into user-level services, persistent services, and remote access services. These services are graphically illustrated in figure 3.2 and all these services are described in detail in section 3.2. ECS uses existing technologies like Commodity Grid Kits (CoG) [16] to provide basic services for authentication and authorization, job submission and

monitoring, and file transfer required to access the distributed back-end resources. In addition to these basic services ECS provides higher-level user services and persistence services. User services such as user workspace, task composer, workflow manager, scheduler, and resource broker are required to create comprehensive PSEs. Persistence services such as application metadata repository, logging, job status, and pedigree services provide access to historic information and support reuse and sharing of information among different users of a DCSE. Persistence services are unique to this architecture and these services can be used irrespective of the mechanism used to access the back-end and the type of front-end client used. Even though these services are classified differently they can be directly accessed through the ECS interfaces. User-level services typically use the persistence and basic remote access services to provide high-level functionality required to create PSEs. Remote access services also exploit the metadata associated with the applications to generate commands required for job submission and file transfer. ECS uses Grid clients to access the distributed Grid resources in the back-end. ECS uses specialized clients wrapped around command-line tools (*e.g., ssh*, *scp*, *krsh*) to access distributed resources that do not support Grid services and uses Kerberos or simple public key infrastructure (PKI) based mechanisms for authentication and authorization.

User-level services and remote access services are implemented as Enterprise Java Beans (EJB) [4] session beans while persistence services are implemented as EJB entity beans. The ECS services are accessed from the web-tier or front-end clients using ECS interfaces implemented as Java Beans [24]. These Java Bean interfaces are also used for generating WSDL interfaces [50] to support language and architecture neutral services. Since construction of WSDL and composition of SOAP messages [63] are not trivial and are both tedious and error prone, web service tools like Microsoft .NET [6] could be used to create WSDL interfaces from native service interfaces.

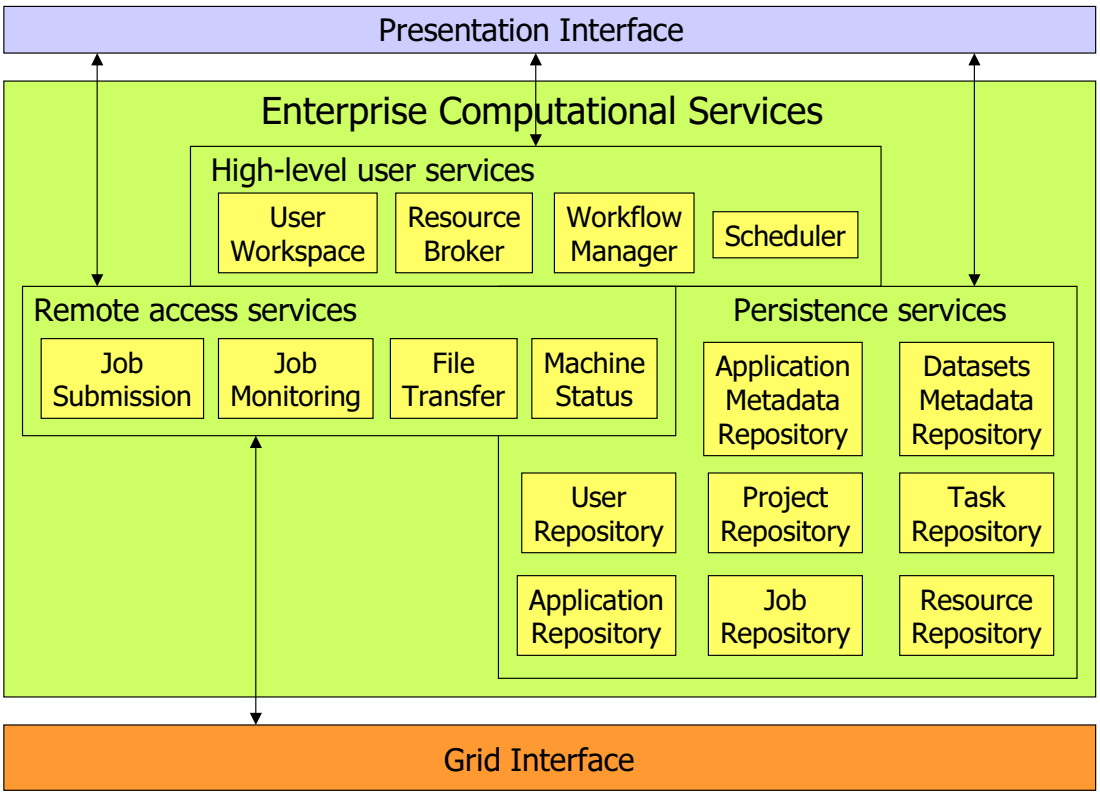


Figure 3.2: Graphical illustration of different services provided by the ECS framework, the key contribution of this dissertation

3.1.3 Back-end

All applications and datasets are distributed on these back-end resources and they are accessed from the middle-tier based on user requests. The back-end consists of both Grid-based on non-Grid based resources, including Storage Resource Brokers, Data warehouses and repositories, instrumentation and observation systems, and dedicated clusters within an intranet. Services for authentication and authorization, data transfer, job submission, and job monitoring are running on these back-end resources.

3.2 Enterprise Computational Services (ECS)

This section describes the different services provided by the ECS framework (illustrated in figure 3.2). Persistence services provide repositories for proxy objects created by ECS for application metadata, users, projects, tasks, applications, jobs, and resources. Remote access services exploit grid services to perform job submission and monitoring, file transfer, and obtain machine status while utilizing the persistence services to obtain information required for job submission, file transfer, and resource description. High-level users services like user workspace, workflow manager, scheduler, and resource broker use remote access services and persistence services to provide high-level abstractions to the user by hiding the details about the repositories and access mechanisms. All three groups of services provided by ECS can be accessed directly by the user through the ECS interfaces.

3.2.1 Application Metadata, Proxy Objects, and Repository

A standalone application residing on a remote machine is represented in the middle-tier by a proxy object and this proxy object is created by capturing all the necessary information required to execute the application. The application metadata consist of the following information about the application [19, 90]:

- Application signature (name, version, authors, keywords),
- Description of the application,
- Pointers to documentation and user manuals,
- Information about support and service,
- List and descriptions of configurable parameters,
- List and descriptions of parameter files,
- List and descriptions of input files required to execute the application,
- List and descriptions of output files that will be produced,
- Details about the source and destination of input files,
- Details about the source and destination of output files,
- Information about how to build the executable,
- List of machines where the application is installed or can be installed, and
- Information about different arguments and options required to execute the application on each host along with default values (number of processors, environment variables, etc.).

Since an application can be configured to execute on multiple hosts, application metadata are categorized into application-specific (machine-independent) and application-independent(machine-specific) information. Any given application can be installed and executed on multiple machines and the metadata contain links to all the different machines on which the application is installed or can be installed if the source code is available. For a given application executing on a specific host, users can change the application-specific parameters as well as machine-specific parameters. To execute an application using the same set of application-specific parameters on a different host, only the corresponding machine-specific parameters must be changed. Application metadata are stored in an XML [62] document to provide flexibility in updating and improving structure and content of the metadata. Sample application metadata is provided in Appendix A. Users can perform the following operations on the proxy objects:

- create a new proxy by providing the necessary information through the process of application registration;
- update or edit an existing proxy object;
- create a copy of an existing proxy object;
- delete a proxy object;
- export a proxy object to be shared with other users;
- import a proxy object shared by other users;
- configure and execute the application represented by the proxy, and
- compose a complex multistep meta-application using individual application proxy objects.

The application metadata stored in the metadata repository is defined as the *abstract* state of an application. In order to use these metadata and execute the corresponding application on the back-end, users must first select a machine and then various application-specific and machine-specific parameters must be provided. Once this configuration information is provided the application proxy is said to be in a *ready* state and can be submitted for execution. The proxy is said to be in *active* or *running* state while the application is executing and when the application completes execution it is in *complete* or *ghost* state. The various states and events that trigger these state transitions are illustrated in diagram 3.3. The state transition from *abstract* to *active* corresponds to an event when applications are submitted for execution on a specific machine directly using default values present in the metadata. When application metadata are selected and applications are composed and/or configured then there is a transition from *abstract* to *ready* state. Submission of a *ready* application(s) results in the transition to *active* state. Completion of executing applications results in a state transition from *active* to *complete*. Since applications in the completed state are stored persistently they can be resubmitted using the same set of parameters used in a earlier run, thereby providing repeatability similar to well-defined scientific experiments. Completed applications can

also be reconfigured and submitted to obtain obtain new products. Except for the *active* or *running* state, all other states are persistent, the *active* state (circle with a dotted line in figure 3.3) is the only transient state during the life cycle of a proxy object. Applications in the *active* state also update the job repository based on the notifications and events received from the back-end to support job monitoring.

3.2.1.1 Abstract State

Application metadata are gathered through the process of application registration wherein all information required to execute an application is captured as an XML file. Instead of manually creating this XML file, the front-end clients provide GUIs generated automatically using the XML schema used to represent the metadata. Application metadata are stored in the application metadata repository and users can perform any one of the following operations using the metadata:

- **Submit** Select an application and execute the corresponding application on one of the hosts using default parameter values causing the state change from *abstract* to *active*. Details about metadata-based submission is explained in section 3.2.9. In this case the application is executed as is assuming that default values were provided using registration.
- **Configure** Select an application and configure the application by providing specific values resulting in the state change from *abstract* to *ready*. The configured application in the *ready* state is stored in the user workspace so that users can reuse the configuration information as well as share configured applications with other users. Configuration details are provided in the following paragraph.
- **Compose** Select different applications registered on distributed machines, add these applications to the user workspace, and compose a multistep task (a collection of interdependent applications) by establishing the dependencies between the different applications. The compose operation is performed on multiple

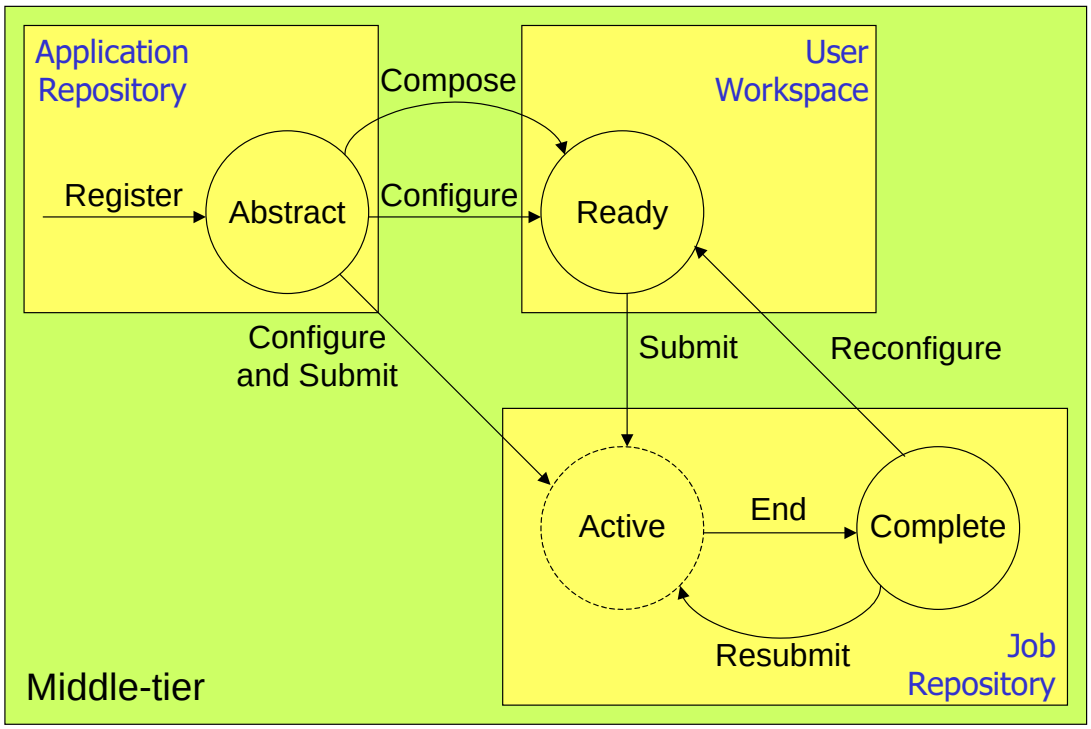


Figure 3.3: A state transition diagram illustrating the different states of an application proxy object

applications and involves configuration of each application. Each application proxy instantiation involves a state transition from *abstract* to *ready*. The configuration and composition information is also stored in the user workspace and section 3.2.2 describes the details about task composition while section 3.2.3 discusses the workflow manager and scheduler, indicating how multistep tasks are scheduled for execution.

In the *abstract* state, a proxy object contains default values for application-specific and machine-specific configuration information. The same application can execute on multiple machines and links to all these machines together with corresponding machine-specific information are stored as part of the application metadata. During configuration, the different application specific parameters required to setup an application are obtained through the front-end clients. These front-end clients can be generated automatically using the metadata or specialized GUIs can also be provided and registered as part of the metadata. Snapshots of sample registration interfaces are shown in Chapter V. If an application should be registered on multiple hosts then a resource broker can assist a user in selecting the host that would provide the shortest turn-around time or the best response time. Using the metadata a resource broker can also select a host that matches the application resource requirements and the user can then either select that host or provide new requirements and/or select a different host. Details about the resource broker are given in section 3.2.4. Furthermore, the task of generating host-specific batch scripts is performed transparently without user intervention. This approach can be further extended to create the notion of an application service provider (ASP) wherein the user does not even care where the application eventually executes.

3.2.1.2 Ready State

The proxy object in the *ready* state contains all the configuration information required to perform a run. Once a particular machine is selected from the metadata corresponding

to the *abstract* state, users can perform any one of the following operations using this persistent information:

- submit the ready application using the default values provided during registration of the application;
- modify application specific parameters;
- modify machine specific parameters;
- modify both application and machine specific parameters;
- reconfigure the application on a new machine.

The submit operation typically involves incorporating the user specified parameter values into the scripts or input files required to execute the application, copying appropriate parameters files and input files to the specific directory on the back-end, and then submitting the application for execution on the remote host. An application proxy object is created in the user workspace when the submit operation is performed and this proxy object contains application-specific values along with configuration information for the selected machine. When a task composed of multiple applications is submitted for execution then the workflow manager analyzes the dependencies between different application components and schedules individual components in the appropriate sequence. All dependencies on input/output files are handled transparently by the workflow manager by copying output files generated by one application to the desired input location of the next application. If specific machines are not selected during configuration, at the time of submission the resource broker can determine the suitable machine based on the application requirements described in the metadata and then submit the application on that machine. To configure an application on a new machine that was not in the metadata present in the *abstract* state, a user has to first select a machine from the list of machines he has access to using the resource repository and then provide machine-specific information required to execute the application on that machine.

3.2.1.3 Active State

An application executing on a remote resource is represented by its proxy object instance in the *active* state. A job proxy object is instantiated in the user workspace during the submission of the actual application on the back-end and this proxy object updates the job repository with status of the application. Once the application is submitted, the application may be queued on the batch system of the remote host, executing on the remote host, or waiting on the completion of another application (if it is part of a multistep task). Users can query the job repository to determine the job status using any one of the the many job attributed. Once the application completes execution successfully or terminates because of an error the proxy object is represented by the *complete* state. Since the proxy object is executing in the middle-tier, it facilitates handling callbacks or event notifications from the application executing on the back-end and update the corresponding job status in the middle-tier. This eliminates the need to have an active process running in the front-end to monitor the status of an application executing in the back-end.

3.2.1.4 Complete State

After an application has completed execution and is in the *complete* state, output files generated during the execution are copied to the predefined locations as specified during configuration. For example, the output may be copied to a mass storage facility, copied to a different host for performing advanced visualization, downloaded to the users desktop for further analysis, or copied to a online repository for dissemination of final results. Details about job start time, completion time, hostname, configuration information specific to this run, etc. are stored in the corresponding job proxy object. This persistent information can be used to describe how a particular final product was produced, to search and query specific products based on product attributes, and to maintain pedigrees. A completed application can be resubmitted for execution using the

earlier configuration information and this provides repeatability. One can also reconfigure the completed application with new parameters and submit the application for execution.

3.2.1.5 Application Metadata Repository

The application metadata repository provides a logical organization for storing, retrieving, and sharing various application metadata. Each user may wish to have a private set of proxy objects that are used for testing and validation purposes, while there may be some proxies that could be shared with other trusted users. An user may also wish to share a subset of the proxies with everyone. In order to facilitate such organization of application proxies, application metadata objects are organized into different groups. Even though any arbitrary number of groups are possible, three groups: private, public, and registered are supported in this work. Each user has a private space that can be used to register, configure, execute, and validate applications. Proxy objects are exported to the registered space so that they can be shared between specific users or group of users. If the application proxies are to be shared by all users then they are exported to the public space. The application metadata repository can be distributed among different servers and on each server the metadata repository can be organized into different groups. Users can search different metadata servers, browse shared metadata, and import selected proxy objects to private space. All of the above-mentioned operations are performed through front-end client interfaces (either graphical or programmatic). It should be noted that the application metadata repository contains only the abstract application states, the ready application states are found in the user workspace, and the completed application states are found in the job repository.

There is a one-to-many relationship between an abstract application descriptor and application proxy objects (different targets, different configurations) and a one-to-many relationship between the application proxy object and job objects (each submission creates a new instance of a job). Consequently, the application metadata are organized

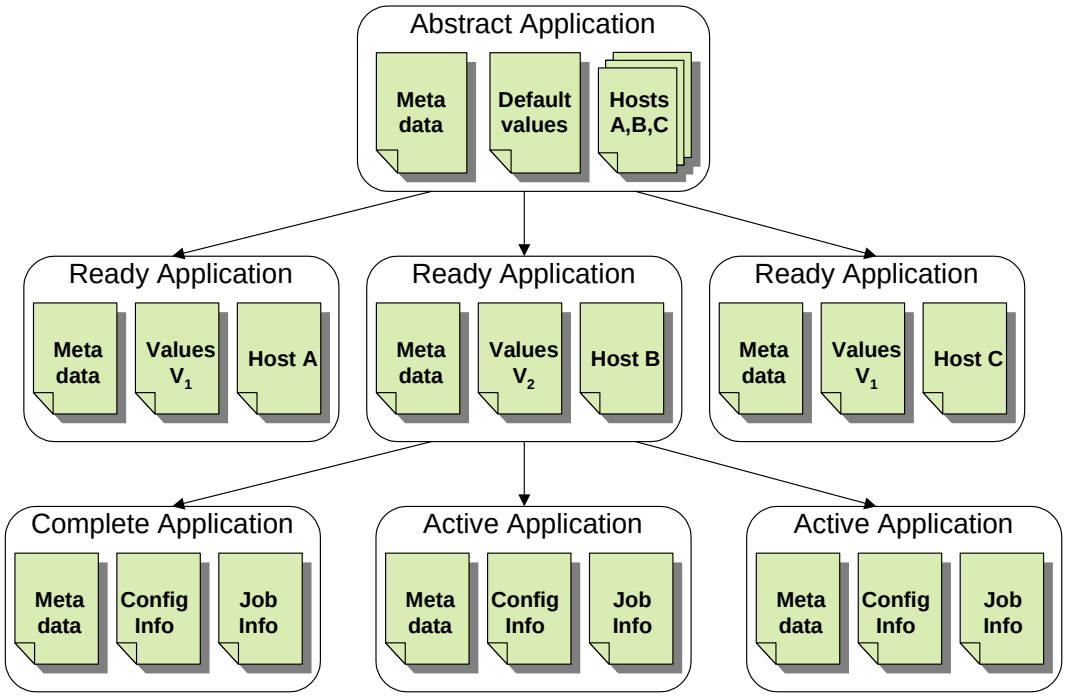


Figure 3.4: Representation of application metadata stored in different states

as a tree, as shown in figure 3.4. The root of the tree contains abstract information about the application along with default application-specific and machine-specific configuration information. Each branch of the tree splits into *ready* instances of the application with specific values for the application and machine dependent parts. User can exploit the same set of application specific values on different machines or provide different values for different machines. For each ready application there can be multiple instances of the *active* or *complete* application corresponding to each instance of a job.

The abstract part of the metadata are generated by the application developer through a registration process (for example, by filling in an HTML form). Each time the application is ported to a new platform, the user who deploys the application adds the new machine-specific parameters. This part of the metadata should be available to all potential users, so it is stored in the public metadata repository. A *ready* branch

is created by the user who configures the application for a submission. Typically, it is stored somewhere in private user space; however, the user may choose to publish his/her configuration to be used by others. This mechanism can be used for transitioning the application for operational use (there is no state transition here, only the ownership of the proxy objects are changed). The application is configured by the domain expert and then routinely run by an operator. The metadata at the *active* state are generated automatically by supplementing the *ready* state with runtime information and stored in the user space. Similarly, the transition from the *active* to the *ghost* state is made automatically following the user preferences and the system settings.

3.2.1.6 Benefits of Application Metadata

Key benefits of capturing the metadata associated with an application are as follows:

- Since application metadata are stored persistently it can be reused to perform simulations repeatedly. Also, an advanced user or domain expert can provide all the necessary configuration information, test and validate the application, and then share the application along with the configuration information with other users. A beginner or a novice user can either run the preconfigured and validated application by importing the proxy object to his or her private area. Once the user gains familiarity with the application one can experiment with different parameters and fine tune the parameters to meet his or her requirements. Thus, many of the steps involved in configuring and sharing of applications are automated by using the application metadata.
- Complex distributed applications can be composed from many standalone applications by defining the dependencies and relationships between these applications (*e.g.,* a dataflow model), thus we are able to achieve distributed computing using standalone applications that may be geographically distributed without requiring any code modifications.

- When the application is updated or bug fixes are performed, then the application developer can validate the application in his or her private space and then share the application proxy with appropriate users. There is no need to distribute a newer version of the software, only the new metadata need to be updated to reflect changes, thus this provides a transparent software upgrade path for the users of the application.
- Metadata and configured applications are stored persistently and this persistent information can be used to annotate the applications as well as provide pedigrees.
- Metadata about the completed job is also stored and this information can be used to describe the final product produced. Search and query functions can use the metadata associated with the final product to retrieve the appropriate result.
- Metadata can be used to automate some of the manual tasks associated with executing an application. Some examples of such manual tasks include: generation of batch scripts, generation of GUI's for obtaining configuration information, generation of RSL strings or appropriate commands for job submission, and data transfers before and after job submission. This is discussed in detail in section 3.2.9.
- Application metadata contain links to the metadata representing datasets used by an application. When datatypes defined in the datasets metadata are used to define input and output files registered with the application metadata, then this information can be used to identify semantic mismatches in datasets and trigger the execution of appropriate data filters or couplers to perform the required data conversion or interpolation when multistep tasks are created.

3.2.2 User Workspace

Application metadata including the default configuration information required to execute an application (abstract state) are stored in the application metadata repository. To execute an application. the metadata are selected and appropriate

configuration information is provided. When the application is executed multiple times the configuration information is overwritten and either the default value or the last set of values are saved in the metadata repository. In order to provide historic information about all the different instances of an application (particularly the ready and complete states) we need a place to save this information and this functionality is provided by the user workspace.

The user workspace provides persistent storage space for applications in *ready* and *complete* states, and also serves as a container for the user to organize different applications into related projects and tasks. For each user, the workspace is organized into a hierarchical structure of projects and tasks similar to how files can be organized into different folders. A project acts as a container for including different related tasks while a task is a collection of application proxy objects along with the details about the dependencies between different proxy objects. Projects and tasks are stored in the project and task repositories, respectively. Typically if the user wishes to preserve the ready and complete states on the application proxy object then a project and a task are first created. Then application proxy objects are added to the task by performing the following operations:

- select an application from the application metadata repository,
- select a specific host where the application will be executed, and
- provide other host specific and application specific information.

When multiple applications are selected to create a multistep task the dependencies between the individual applications are captured in the task metadata that is part of the task object. Sample task metadata shown in figure 3.5 represents dependencies between three standalone applications configured to execute on different machines that are geographically distributed. The front-end clients provide GUIs to add applications into the workspace and establish the dependencies. Each application proxy object thus created is now in the *ready* state and task composed of these application proxy objects in

```
<task>
   <application>
   <id>1001</id>
   <name>COAMPS</name>
   <input></input>
   <output>1002</output>
   </application>
   <application>
      <id>1002</id>
      <name>WAM</name>
      <input>1001</input>
      <output>1003</output>
   </application>
   <application>
      <id>1003</id>
      <name>NCOM</name>
      <input>1001</input>
      <input>1002</input>
      <output></output>
   </application>
</task>
```

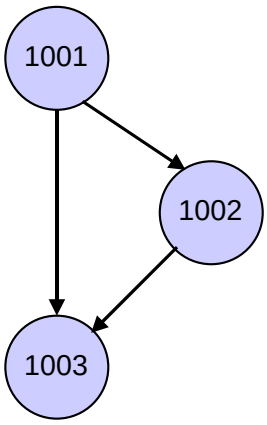


Figure 3.5: Sample task metadata that represent the dependencies between three geographically distributed standalone applications

the *ready* state can be submitted for execution on the appropriate machines. Thus, user workspace support task composition when a collection of interdependent applications are connected together to form a multistep task.

If a task is composed of a single application then the process of task submission is exactly the same as submitting an application as explained in section 3.2.9. When a task consists of multiple distributed applications with dependencies then the workflow manager uses the task metadata associated with that task and submits the appropriate application for execution as described in section 3.2.3. When an application is submitted for execution a corresponding job proxy object is created in the middle-tier. While the application is executing the job object corresponds to the active application state and when the application completes the job object is in the complete state as described earlier in section 3.2.1. The job object contains details about the product produced along with the configuration information. Using this information one can search for specific products

and then determine who created this product, when was this product created, and what datasets and input parameters were used to develop this product. Users can determine all jobs created on different machines for a specific project, task, or application.

The user workspace also promotes collaboration between different users of a distributed simulation system by allowing users to reuse and share tasks through the import and export operations. A develop or an advanced user can create a complex task involving multiple applications in the user's local workspace, validate the task, analyze the results, and share the task with other trusted users by exporting or publishing the task. A novice user can import the task to his or her local workspace and use the preconfigured task as is to understand the functionality of the task. Once the user is familiar with operating the task and its functionality then the user can experiment with different application-specific parameters or test the application with new datasets. If the user is simply interested in running these tasks on a routine basis by changing a few parameters, the user needs to only change these parameters and submit the task for execution. Thus software developed by researchers can be easily transitioned to operational use through the import and export operations defined on tasks. This reduces the time required to understand and learn how to configure and execute complex applications.

3.2.3 Workflow Manager and Scheduler

When a task consists of multiple applications with dependencies in their order of execution then the workflow manager is responsible for executing the appropriate application in the prescribed order. The workflow manager creates the appropriate job proxy objects in the middle-tier and provides necessary synchronization and communication between these proxy objects. The job proxy objects in turn submits the application to the scheduler to create the corresponding application in the back-end. The scheduler is responsible for performing the job submission from the middle-tier

and the scheduler can handle both time-based and event-based requests. The most common time-based request is to submit a job immediately while advanced time-based scheduling requests could include running an application at specific time-intervals (*e.g.*, every 12 hours, 10th minute of every hour). Event-based scheduling can be used to schedule individual applications that are part of a complex task involving input/output dependencies. The completion of one or more applications can trigger the execution of one or more applications.

The dependencies between individual applications in a task are described by the task metadata. The task metadata is an XML document with several application tags as illustrated in the figure 3.5. Each application tag defines the name of the application, application id, input ports, and output ports. When output from one application is required as input to another application then the output port of the first application contains the application id of the second application and the input port of the second application has the application id of the first application. Each application can have arbitrary number of input and output ports.

Some of the different types of dependencies that are common among HPC applications include sequential ordering, parallel ordering, complex ordering. A sequential ordering involves a collection of applications that should be executed only after its preceding one is complete while a parallel ordering involves a collection of application which can be executed simultaneously without waiting for any of them to complete. A complex ordering involves dependencies described through a directed acyclic graph wherein the output from multiple applications is required to start an application. All these dependencies are illustrated in figure 3.6.

The workflow manager has to create proxy objects for each application in the order described in the task metadata and the proxy objects in-turn will request the scheduler to submit the corresponding job on the back-end. The workflow manager submits the constituent applications of a task in the specified order using the scheduler, monitors

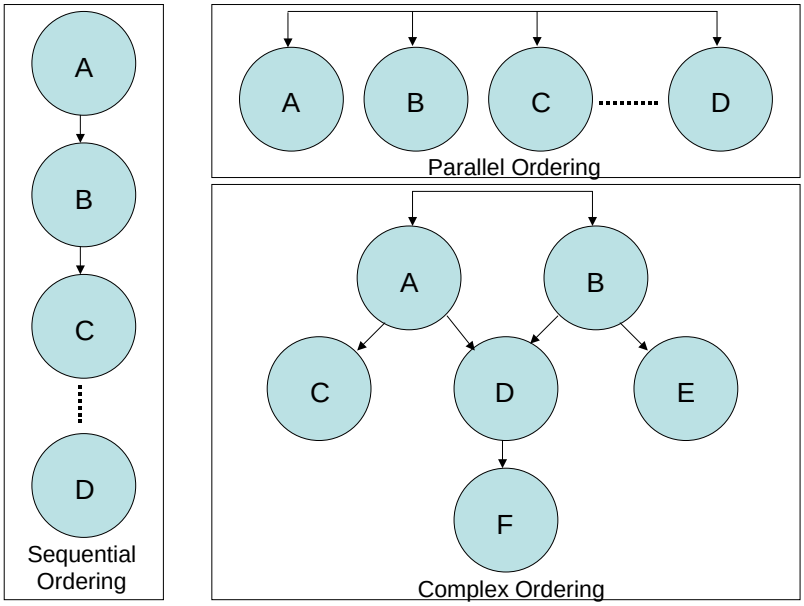


Figure 3.6: A graphical representation of common types of dependencies in the order of execution of standalone HPC applications

status of each submitted application and listens to events, as well as transparently moves input and output files across the distributed system, as needed. There are several options to implement the workflow manager and let us consider some of the commonly used approaches [91]. One approach would be to create all the proxy objects that does not have any dependencies as separate threads and wait for the corresponding applications on the back-end to complete before creating the next set of proxy objects that required the completion of the first set. An alternative approach would be to create all the applications as separate threads and exchange messages between different threads that have dependencies to start execution on the back-end. The first approach relies on explicit synchronization and can be implemented using semaphores or conditional variables whereas the second approach uses message-passing with implicit synchronization and can be implemented using message queues. The J2EE architecture supports Java Message Service (JMS) [5] as a means to provide asynchronous message passing capability. JMS provides both point-to-point and publisher-subscriber styles of

messaging and we use the publisher-subscriber messaging to implement the workflow manager. The algorithm for the implementation of the workflow is as follows:

1. read the task metadata,
2. for each application in the task metadata create the job proxy object as a separate thread,
3. each job proxy object subscribes to the messaging service,
4. the main thread publishes messages with the topic equal to the application id of all applications that do not have any dependencies (no input port entry), so that they all can start execution independently,
5. each proxy object waits until a message with the topic equal to its application id is published,
6. all the applications that received the message with the matching application id will request the scheduler to submit the corresponding application for execution on the back-end and wait until that application completes, and
7. once the application completes on the back-end the proxy object publishes a message with the topic equal to the output port address (if there is a output port entry) and then exits.

Requests for immediate scheduling and event-based scheduling typically involves performing a submit operation while scheduling periodic tasks requires a timer to perform the submit operations at the requested time. To support advanced scheduling a schedule table with information about the task to be submitted, when the task should be started, and how frequently the task should be submitted is maintained in the middle-tier. The scheduler process is typically started by the administrator of the system and it uses a timer object and the information from the schedule table to submit tasks at appropriate time-intervals. Users can select a task, provide scheduling information, and create a new schedule using the front-end clients. Similarly users can also modify existing schedules as well as delete them. A scheduled task can be paused, stopped, or cancelled.

3.2.4 Resource Broker

HPC users have access to several systems and on each system they have an allocation for the number of CPU hours alloted for their individual use or for their project. If the application is portable across all the platforms the user has access to then the user has to decide where to execute a particular application. There are several options available to the user and some of these options include selecting a machine that:

- has maximum allocation available,
- has least number of jobs waiting in the queue (so that there is a better turn-around time),
- has the fastest processors available,
- has maximum number of processors available, and
- has maximum amount of memory per node.

In order to decide where to submit an application users must perform the following tasks on each of the machine user has access: have to login to the machine explicitly, identify the different batch queues, determine the characteristics of each queue, select the appropriate queue(s), and using the machine specific batch commands obtain the current status of the selected queue(s). While grid services like the Grid Index Information Service (GIIS) [35] and Grid Resource Information Service (GRIS) [35] and service portals like Hotpages [78] provide information about status of the machines, the user has to still decide where to execute an application so that his requirements are satisfied.

Using the ECS framework, if an application is registered on multiple machines, the user has to manually select one of the machine based on some criteria during configuration or submission and then submit the application for execution. Instead this process can be automated by allowing the user to specific a set of requirements and the resource broker can determine the appropriate machine that meets the user's requirements. The resource broker could also determine the most suitable machine based on available user allocations,

machine characteristics at the time of submission, application and users requirements. If the resource broker should be able to find only one machine that matches the user requirements then the resource broker can submit the application without requiring any further user intervention. Similarly if the resource broker cannot find any machine that matches the requirements then the resource broker has to wait until the required resources are available or inform the user that the resource broker could not find a suitable match. The user can then either change the requirements or let the resource broker wait until such resources become available. If the resource broker determines that multiple machines can satisfy the user requirements, then the broker could select a default machine or the first in the list and submit the application. Instead the resource broker could also provide this list of machines to the user and let the user select the appropriate machine.

The application metadata have fields to specify such requirements that the resource broker can use to make the appropriate decisions without requiring any user intervention at the time of submission. In order to decide which machine to submit an application the resource broker requires access to the list of machines on which user has allocations, the current status of the machine with respect to current load average, allocations units remaining for the user, total number of processors on the system, number of jobs currently executing, waiting, queued, and so forth (in addition to the application metadata). The user profile has a list to machines the user has access to and the resource repository provides both static and dynamic information specific to a machine. Static information includes actual number of processors, different queues, authentication and authorization mechanisms. Dynamic information include machine load, number of jobs currently running, waiting, and queued.

It is important to automate this operation of selecting a specific machine for submission particularly when complex tasks are composed with individual applications that have dependencies in the order of execution. In such a situation, the user can select

the machines up-front during configuration or let the resource broker decide the suitable machine at runtime. Allowing the resource broker to select machine at runtime provides the user with better options to meet his requirements since the latest information about the machine status and user allocations will be used to make the decision. It is possible to in advance to specify quality of service (QoS) requirements (like a deadline, start time, end time, minimum-maximum number of processors) in the application metadata and let the resource broker determine the machine that satisfies such requirements. With the emerging on-demand computing and computational economies the resource broker's role will become even more important than it already is. In addition to traditional parameters like utilization, efficiency, throughput, and turnaround, new parameters like cost and availability will also likely play a role in determining resource optimality.

3.2.5 Resource Repository

The resource repository provides the list of computer resources, mass storage devices, and data repositories that are registered with the system. For each resource registered static information such as the machine name, IP address, operating system, number of processors, details about the various queuing systems (for batch systems), access mechanism to connect to the machine, and paths to common compilers and development utilities are stored. For each queuing system details about the queue including name of the queue, name of the queue server, number of processors and maximum memory available through this queue, queue specific commands for submission, monitoring, and deletion are also stored for each machine. The XML document for an entry in the resource repository is provided in appendix B. The resource repository is referenced in the application metadata and used during application registration, configuration, and submission. Job submission service uses the resource repository to obtain machine access type and queue details required to submit an application. A user profile contains links to the resources that the user has access to and the resource broker uses machine specific

information to determine if it satisfies user's requirements. To support the resource broker, the resource repository also contains dynamic machine specific information. These information can be obtained by connecting to Lightweight Directory Access protocol (LDAP) [36] servers like GIIS [35] or GRIS [35]. For non-grid resources, a separate process executes in the middle-tier and gathers all the necessary information periodically.

3.2.6 Job Repository

The job repository contains collection of job proxy objects that represents applications in the active and completed states. In addition to the information stored in the application proxy object corresponding to the ready state the job proxy object consists of current job status, start time, end time, hostname, queue id (for batch jobs), and user workspace details (project and task ids). The job proxy object is created when an application is scheduled for submission and updated either by the corresponding application proxy object running in the middle-tier (when the back-end provides a callback mechanism) or through a separate job monitoring service (when there is no callback mechanism while submitting to a batch system using command-line clients like *krsh* or *ssh*).

The job repository provides the current status of an application submitted for execution as well as historic information about different instances of an application's execution. Since configuration information is also stored along with pointers to inputs used and outputs generated, users can select a completed application and determine what products were generated, how they were generated (what input datasets and parameters were used), and when they were generated. Thus we are able to annotate the final results produced by a simulation. This information can also be used to perform search and query operations to locate a specific product based on the product attributes captured during the application registration phase. Additional search and query operations based on user,

project, task, application, machine, and queue can also be performed on the entries in the job repository. Users can view the status of all jobs submitted on different machines at the same time using the job repository, instead of connecting to each machine separately.

3.2.7 Credentials Repository

In the simple client-server architecture where command-line grid clients are used for job submission, monitoring, and file transfer the proxy credentials created on the client-side are used for authentication and authorization of the grid resources. In case of multi-tier architectures, the back-end grid resources are accessed from the middle-tier by a separate process that belongs to a different user. User credentials are created on a user's desktop and this credential must be transfered to the middle-tier so that the middle-tier can access the back-end on behalf of the user. This mechanism of copying the proxy credentials to the middle-tier works well as long as the user can create a proxy credential and upload it to the middle-tier. To ensure security, typically user's credentials are stored as files on a file system and this file system is accessed through a secure manner (for example, using ssh). If users are not physically near their primary system or logged in from a system that does not have grid clients to create a proxy credential, then they will not be able to access the back-end using the services provided by the middle-tier. In order to allow users to access the back-end from any network connected computer without requiring access to user's long-term (permanent) credentials, one has to provide a credentials repository to store and retrieve proxy credentials. Such a credentials repository called MyProxy [46] already exists and this effort uses the MyProxy server to create and store users proxy credentials and the middle-tier retrieves the credentials when users requests services from the middle-tier. Details about the MyProxy credential repository system can be found in [46].

3.2.8 Datasets Metadata Repository

Data is available from wide variety of sources (observation systems, instruments, simulations) in different forms and formats. Integrating such heterogeneous databases and data servers is challenging because not only are the APIs of the various data providers different, but the databases and data servers often employ independent terminology, which leads to semantic mismatch between the data sources. The differences between the data sources are difficult to reconcile, since data formats introduced by the data providers represent their unique, extensive experience of using and optimizing the data for specific applications. Assuming that all data providers choose to maintain (and possibly evolve) their proprietary schemas and APIs, it is necessary to provide dynamical mechanisms for data conversion on the fly in addition to supporting data discovery and lookup. This can be achieved by introducing a filter or a coupler responsible for the data conversion. Depending on the data structure, the coupler can be implemented simply as a format conversion, interpolation, or a complex code capable of moving data between numerical grids of different topologies and resolutions, satisfying some physical constraints. The format of the data at the data source, and the format of the data expected by its consumer (data sink) determine which coupler to be used.

In section 3.2.1 the advantages of using metadata to represent applications residing on distributed computational resources were mentioned. Similarly metadata for datasets supports annotation of data and retrieve the required data in the desired form/format by searching the attributes of the data (metadata). It is easy to perform search and query operations on the metadata since the datasets are most often in binary format and often difficult to deduce what it represents. If data is in a self-describing format (*e.g.,* HDF [92], NetCDF [93]) then the metadata can be created automatically otherwise metadata is created through the process of dataset registration, similar to the application registration process. By associating a datatype with the metadata it is possible to determine in advance if a particular dataset can be used with a specific application

or what datatype is required to execute a given application, thereby identify semantic mismatch on datasets. If such a mismatch occurs then data couplers (or filters) can be implemented and registered in the application repository, so that the couplers could be applied on such data mismatches. When accessing large scale datasets from a remote location one could execute these filters and obtain reduction in space and time by copying the smaller subsampled dataset. By linking datasets metadata with the application metadata, if desired datasets are not available then the application(s) that can generate such datasets can be configured and executed to obtain new datasets.

3.2.9 Metadata-based job submission

In order to submit an application for execution on a remote HPC system users must first explicitly login to the selected system, prepare the application for submission, create batch scripts specific to the queuing system on that machine, and then submit the batch script. To monitor the status of the job in the queue user has to login again and use the appropriate command-line utilities. Once the job is complete the output files produced may be copied to the local workstation for further processing and analysis or copied to a mass storage facility for archiving. To execute the same application on a different machine, the same process is repeated by using machine-specific scripts and commands. If any bug fixes or changes are made to the code or the scripts then those changes need to be propagated back to all the systems on which the application is installed.

Grid software toolkits like Globus [31] provide command-line clients that can be used to submit applications on multiple remote hosts using a single sign-on proxy certificate. There is no need of logging in to each machine explicitly, instead machine status can be obtained through the Grid Information Services [35] and GridFTP [41] can be used to transfer files between local workstations and remote resources as well as directly between two remote resources. The Resource Specification Language (RSL) [37] is used to submit jobs without requiring any batch-system-specific commands. When jobs are submitted

through the globus toolkit clients, globus returns a URL that can be used to query the status of the job and also obtain the standard output and error streams generated. Nevertheless, users are still responsible for creating the RSL script, submitting the script, and monitoring the status of the job as the job progress. The job submission service provided by Globus is a transient service and the user has to save the URL returned during the submission to monitor the job status. The different steps performed to submit a job using Globus command-line clients are as follows:

- select a machine to execute the application;
- learn how to execute the application by finding out where the executable is, environment variables and values that should be set, different arguments and switches used by the application;
- create RSL string to stage files, execute the application, and download results;
- learn how to setup the execution of an application, edit configuration files, provide appropriate parameters, input files required, preprocessing steps that should be performed, output files generated, and postprocessing steps required to analyze the final results;
- verify that the data is preprocessed and available in the format required by the application at specific locations on the target host machine, otherwise preprocess the data;
- generate a proxy certificate and submit the application for execution on the host selected with the RSL string and hostname as arguments for the command-line grid clients; and
- use the URL returned by the submit command to monitor the status of the submitted job and check the standard output and error streams.

To minimize user intervention and provide customized assistance for job submission we use the information captured in the application metadata to create the RSL scripts or machine specific batch scripts. Also, when applications are submitted for execution

the corresponding job proxy object is saved in the job repository and this job object contains information about the current status of the job and links to input files used and output files generated. Thus, the job submission service provided by ECS does not require manual generation of RSL and it is also a persistent service. The different steps performed to submit a job using the application metadata are as follows:

- select an application using the metadata repository,
- create an instance of the application for a specific host,
- configure the application using the GUIs provided with default values,
- submit the application, and
- monitor the status.

All of the above operations are performed by the user using the front-end clients while the corresponding middle-tier user-level services handle the task of creating proxy objects and accessing the back-end resources. The knowledge about the configuration of the code is encapsulated in the metadata and presented to the user in the form of GUIs. The user does not need to know anything about the low-level Globus interfaces, syntax of RSL or syntax of batch schedulers on the remote systems. Thus services provided by ECS in the middle-tier reduce user intervention and automate the process of job submission.

3.2.10 Job Monitoring

HPC applications are typically submitted for execution through a batch system using batch scripts to maximum resource utilization and satisfy the policy enforced by the stakeholder. An application submitted to the batch system will be queued until it is scheduled for execution. Often users check if their job is running and if so look at the outputs produced to determine the progress of the executing application. To determine the status of the job in the queue users must explicitly logging in to the system and then using the batch system specific command-line tools check the status of their job. Users

typically have the option to check the status of a specific job or check status of all jobs that belong to the user or check all the jobs in a specific queue and each batch system has its own syntax. To check the status of a specific job users should remember the *job id* returned during job submission. Once the user determines that the job is running the user can look at the output produced so monitor the progress of the application (*e.g.,* whether the solution is converging, how many more iterations still left).

The Globus toolkit provides command-line clients to determine the status of a job submitted for execution and to obtain the standard output and error streams. These grid clients eliminate the need for explicit logins and knowledge of system specific batch commands. But in order to use these clients users need to remember the lengthy URL returned by during the job submission process. Also, users cannot identify which job corresponds to which URL by looking at the URL. Even though users can retrieve standard output and error streams if they need to access any other output files generated during execution, then they have to either login explicitly to the back-end resource or transfer the file manually to user's desktop. By way of contract, the ECS job monitoring service provides simpler access to monitor the status of jobs executing on back-end resources and access different output files produced by these jobs.

When an application is submitted for submission using ECS, a proxy job object is created in the middle-tier and this proxy object in turn creates an instance of the application in the back-end. For Globus-based job submission, Globus Resource Allocation Manager (GRAM) [37] provides a callback mechanism to notify any change in the job status and the job proxy object executing in the middle-tier updates the status of the job object. The job object contains information about the project, task, application, time of submission, actual job start time, current job status (queued, active, complete), queue id returned by the batch system (if job is queued or active), and completion time (if job is complete). Thus users can look at the job repository to determine the status of their jobs by identifying a job based on any of the job attributes.

Users can also select a specific job and obtain more detailed information like the input files used, input parameters specified, and output files generated (including standard output and error). Users can also select any of the input/output files and download it to their local workstation for postprocessing or analysis. Since all job objects are stored persistently in the job repository, users can obtain all job related information even after the job is complete and there is no need to remember any job ids or URLs. Thus the job monitoring service provided by ECS provides a simpler interface to monitor the status of the applications after submission while providing historic information for different instances of the application along with the information about how, when, and where the products were generated.

When applications are submitted to batch queues on back-end resources which use Kerberos-based authentication and authorization mechanisms there is no mechanism available to notify the change in the status of the submitted jobs (except email notification). Thus, one has to execute a separate process in the middle-tier that periodically connects to the back-end and obtains the status of all submitted jobs. This job monitoring process must be run with the credentials of a valid back-end user to obtain the job status since it has to connect to the corresponding back-end resource. Instead of running a separate job monitoring process for each user one could run one process per machine and then the appropriate entries in the job repository are updated whenever there is a change in the job status. Since a job proxy object is created in the middle-tier for all jobs created on the back-end, interactive jobs can be easily monitored using the job proxy object.

3.2.11 File Transfer

Computational simulations require input files to be available locally where the application is executed and the final output of the simulation is also written as output files residing on the same system. Input files are obtained from various data sources and

will not be available on the system where the application will be executed. Thus, one has to manually copy the necessary data files to the appropriate back-end resource and preprocess this data if it is not in the format expected by the application. Similarly the output files generated by the successful execution of the application are manually copied to a data archive, copied to another remote resource for postprocessing, downloaded to the local workstation for visualization and analysis, or distributed to end-users through online repositories or web sites. Typically these copies are performed through command-line tools like *ftp*, *rcp*, and *scp* by providing username, password, and filenames. Grid clients like GridFTP support file transfer between grid-enabled resources using the GSI-based proxy credentials, thereby eliminating the need for providing username and password while copying. Similarly Kerberos-based clients like *kftp* and *krcp* support data transfer across Kerberos-based resources.

ECS uses these basic services to perform file transfer transparently during job submission and job monitoring. During job submission the various input and parameter files registered in the application metadata are copied from their source locations to the appropriate working directory on the selected machine. Once the execution is completed the output files produced are also copied to the corresponding destination locations as specified during the configuration and submission. Also when the user selects a particular job from the job repository and wishes to view any of the input or output files, then the files are transparently copied from the back-end to the front-end. Using this basic file transfer service a uniform remote filebrowser tool is developed to provide seamless access to files distributed across grid and non-grid environments. This file browser not only allows users to list files and navigate through different directories on a back-end machine but also provides capability to remotely edit the files and save the changes on the back-end. For files that use self-describing format (like NetCDF, HDF) users can view the metadata associated with such files without downloading and opening the entire files. For such files, one can extract the metadata associated with the files by applying

filters at the data source and then display the metadata extracted. This is a useful feature since users can determine the contents of the file without reading it completely or downloading the entire file, thereby reducing the time required for file copies as well as reduce redundant storage space.

3.2.12 User Profile

In order to support different types of users in a multidisciplinary computational system a *role* is assigned to each user based on functionality required by that user. It is possible that a single user may assume different roles and the administrator can assign these roles based on user requests. The administrator can also customize these roles by adding new functionality to existing roles or by creating new roles based on a different set of functionality. These roles can be used to provide not only different types of service for each user but also different types of user interfaces. Some of the sample roles along with their corresponding functionality are as follows:

- **Decision-maker** A decision-maker typically is not an expert in computational simulations or modeling requires access to information and knowledge to make tactical decisions.
- **General public** General public requires access to already processed and annotated information. This user requires search and query capabilities to access and retrieve required information. Information requested may be an animation, image, plot, or chart.
- **Analyst** An analyst is involved in validating, analyzing, and evaluating different applications. This user requires access to data from multiple sources (*e.g.,* experimental, observation, simulations). An analyst may require access to execute preconfigured models, support for advanced visualization of results, and access to format convertors, data filters, interpolation/extrapolation routines, and data subsampling and compression techniques.

- **Operator** An operator is involved in executing validated and preconfigured applications on a routine basis by providing a fixed set of parameters. An operator may not be a domain expert and may not be involved in making changes to the source code and recompile applications. Typically, the final product produced by executing an application will be ready to be consumed by the end-user (all necessary postprocessing is also done as part of the application). An operator requires access to a specific set of distributed resources to obtain input datasets, execute the application, and save the final products. This user may also require special support to execute tasks at specific instances of time.
- **Developer** Developers are users with the most requirements. They are involved in developing new applications, incorporating new algorithms, validating applications, maintaining and supporting applications, creating preconfigured task streams for execution by an operator or analyst, and so on. Developers require access to distributed compute resources, datasets, and software components, support for creating and composing complex applications from standalone applications distributed across organizational boundaries, job submission and monitoring, secure and efficient data transfer, persistence and pedigree services.
- **Administrator** Administrators are involved in administering and managing the distributed computational system by creating user accounts, setting appropriate permissions and access control to support sharing of different system components (applications, datasets, and products). They are also involved in keeping the system up and running, performing necessary upgrades, and maintain a running system.
- **GCE Developers** Software developers who create domain specific portals or desktop applications and require programmatic access to services provided by the middle-tier and access back-end resources.

Users in the role of *decision-maker*, *public*, *operator*, and *administrator* are typically not domain experts and they often require simple user-friendly interfaces that can be accessed from anywhere. Users in the *developer* role and *analyst* role are domain experts and typically require wide variety of functionality and access to local desktop applications for the purposes of preprocessing, visualization, and postprocessing. GCE developers and maintainers require programmatic access to integrate different services and create domain specific environments. Based on a mapping between different user roles and services, a user profile is maintained for each user and the different roles assigned to a user are maintained in the user profile stored as part of the user information in the user repository. In addition to the user role and services assigned to that role, a user profile also contains additional information about user preferences, contact information, list of resources the user has access to, and credentials required to access these resources.

3.3 Front-end Clients

In a multi-tier architecture the front-end clients provide graphical user interfaces (GUIs) through which user requests are accepted, forwarded to the appropriate middle-tier services for processing, and display the responses generated by the middle-tier. These interfaces hide all the complexities involved in performing the needed actions in the middle-tier and back-end while automating some of the tasks that require human intervention. Furthermore, these user-friendly interfaces helps users to learn the system quickly and train new users without encumbering the advanced user. Typical problem solving environments provide one type of front-end interface and this interface is well suited for only one type of user and cannot effectively support all the different types of users described in section 3.2.12. This section describes how multiple front-end interfaces are supported using the ECS framework. To create a domain specific problem solving environments only the front-end needs to be customized for that specific environment.

It is also possible to provide multiple clients for different types of users within a single distributed system.

3.3.1 Web Browser Based Clients

The majority of the browser based GCEs use HTML [60] for displaying content along with Javascript [94] for providing client-side processing while dynamic content is generated using Servlets [57], Java Server Pages (JSPs) [58], and CGI/Perl scripts on the server side. To support such web clients we use a JSP based web-tier and the JavaBean components [24] to access the services and functionality supported by the ECS as shown in figure 3.7. JSP tags are used to create reusable presentation templates and Javascript is used to perform error checking and minimize unnecessary connection to the web server. The communication between the browser and the web-tier is through HTTPS and the communication between the web-tier and the application-tier is through Java RMI or RMI over IIOP [2] depending on whether the web-tier and application-tier are running in a single machine or distributed across multiple machines. The web server and the application server are typically hosted behind a firewall and applications outside the firewall cannot access the application server directly, access to the application server is allowed only from a specific web server. Also, to access the application server, users must have an account on the application server with appropriate assigned roles and permissions.

Web browser-based clients are ideal when middle-tier services are to be accessible from anywhere on the Internet without the need for any special software. Since all the processing occurs at the server side, there is sufficient latency involved in getting responses back from the server and the response time also depends on the network characteristics too. It is furthermore difficult to provide rich graphics and access to tools and applications on the user's desktop using browser-based clients. Browser-based clients are suitable for users in the role of decision-maker, public, operator, and administrator (as

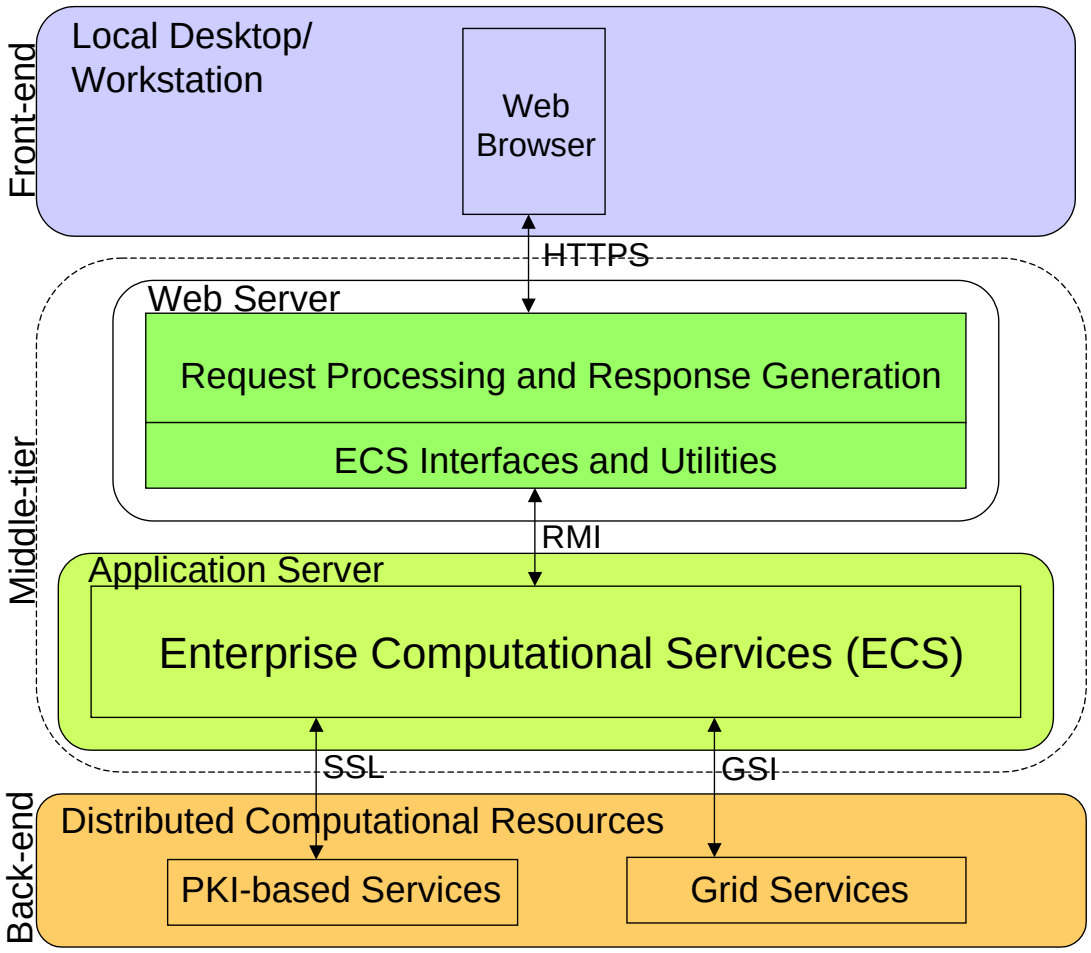


Figure 3.7: Illustration of the ECS framework supporting a browser based front-end client accessing Globus-based back-end

described in section 3.2.12) since ease of access and ubiquitous access are more important than response time and detailed graphical capabilities.

Using a browser-based client one can access all the different types of back-end resources with certain restrictions. Grid-based and simple PKI-based back-end resources are accessed from the middle-tier on behalf of the user using user's credentials (either uploaded by the user or retrieved from the Myproxy server [46]). For accessing Kerberos-based back-end resources that do not support credential delegation, one of the options is to use a lightweight version of the web server [82] that runs on the desktop and uses user credentials created on the desktop for job submission, monitoring, and file transfer as shown in figure 3.8. Since the HTTP and HTTPS protocols do have the ability to carry Kerberos credentials, some of the options available to transfer the credentials securely to the middle-tier include tunneling the HTTP request through a kerberized protocol such as kerberized rsh, using Kerberos-based CORBA security services, and sending the user credentials (the Kerberos ticket) as a cookie in an encrypted HTTPS request [74].

3.3.2 Desktop applications written in Java

Front-end clients that require interactive and customized GUIs typically use Java Abstract Windows Toolkit (AWT) and Swing components [95] to create desktop applications that provide the required functionality. All the functionality provided by the web-tier to process requests and generate responses will be duplicated on the client side in the desktop application. Such clients use the JavaBean components directly within the desktop application and access the ECS services using Java RMI (since both the client and the server use Java) as shown in figure 3.9. To ensure secure communication, the Java RMI calls are tunneled through HTTPS connections with the application server which is typically behind a firewall. The RMI server on the application server accepts secure connections only at specific ports, access to these services are authenticated, and only authorized users are allowed to access these services.

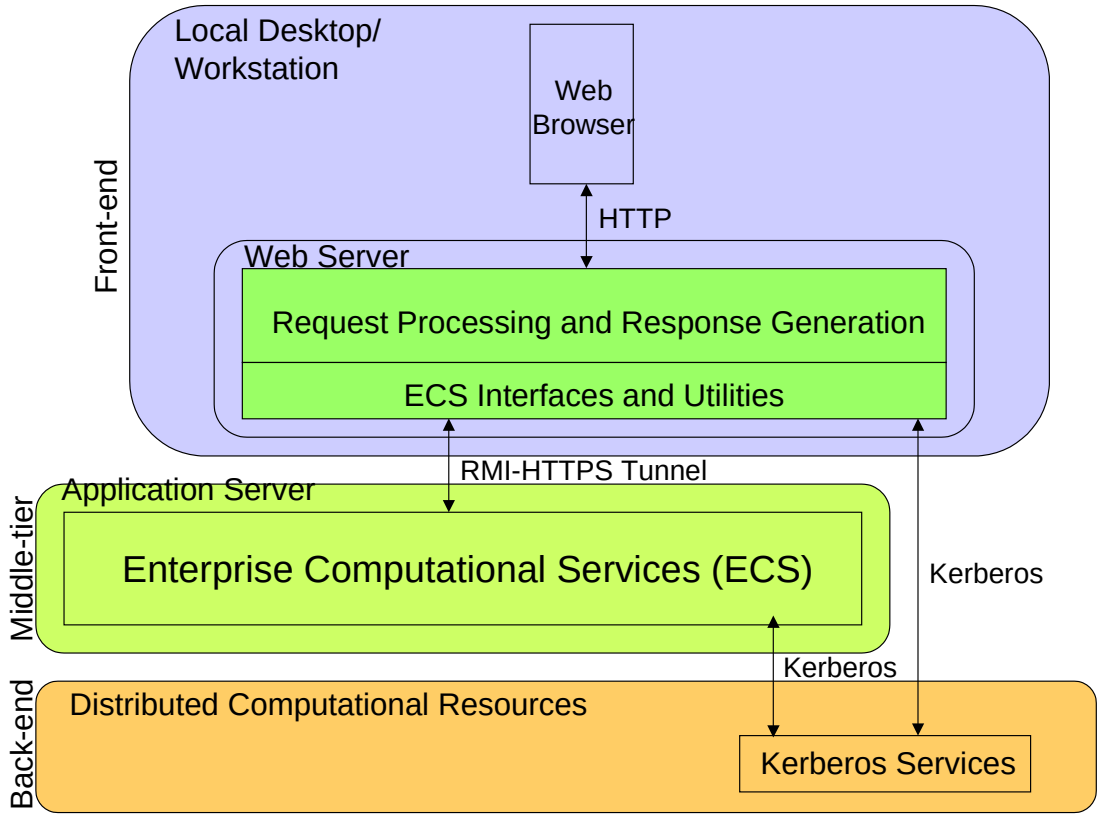


Figure 3.8: Illustration of the ECS framework supporting a browser based front-end client accessing Kerberos-based back-end

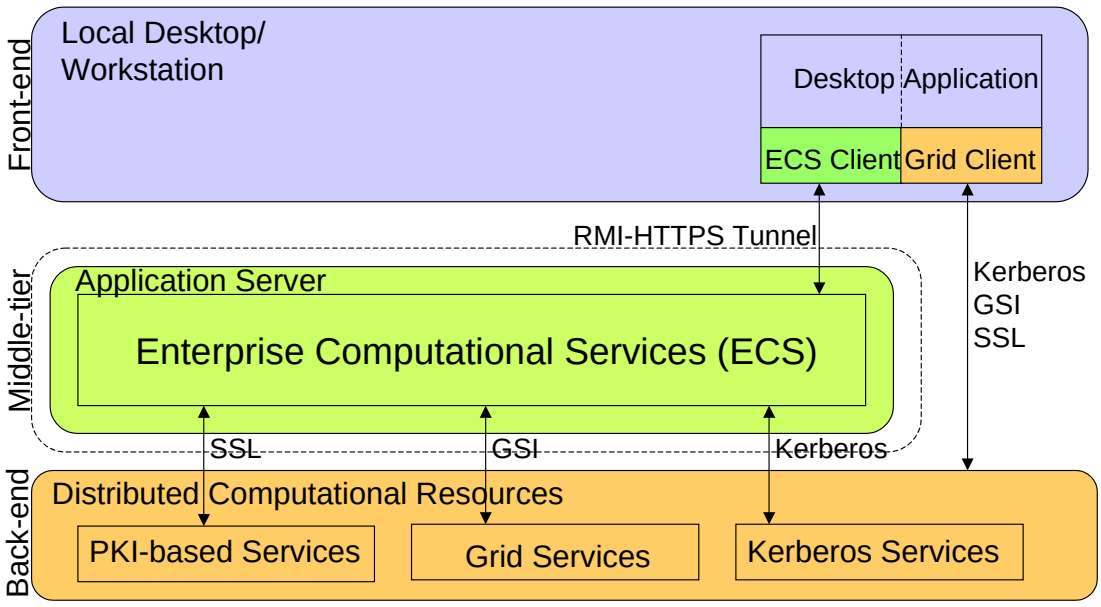


Figure 3.9: Illustration of the ECS Framework supporting Java based front-end clients

Desktop applications are suitable for front-end clients that require rich graphical capabilities, interactive and personalized GUIs, faster response time, lower latency, and access to local desktop resources and applications. Desktop applications along with other necessary libraries and utilities should be downloaded, installed, and maintained on the user desktop. Whenever new interfaces or functionality is provided by these clients the newer version of the client software must be downloaded. This software maintenance and upgrade problem can be easily solved by using the Java Web Start technology [96]. The application is downloaded, installed, and launched by just clicking on a webpage link and the application will connect to the server whenever any updates are made. There is no need to manually check and download the latest versions. Desktop applications are well suited for developers and analysts who require rich functionality and access to local desktop resources and tools.

Desktop applications can access the back-end for job submission and monitoring and file transfer, either from the middle-tier or directly from the front-end. Thus desktop clients are more suitable when Kerberos based authentication and authorization schemes

are used to access the back-end (especially when credential delegation is difficult to achieve with secure identification cards).

While performing job submission and monitoring directly from the front-end, if there is a callback mechanism provided by the back-end to update the status or notify any events, then the front-end client should be active and listening for such events. If the front-end client is closed then we will not be able to update the corresponding job status. To address this issue, the middle-tier services are used for job submission and monitoring for Grid-based back-end resources and job submission is performed directly from the front-end on Kerberos-based back-end resources. A separate process running in the middle-tier performs the job monitoring and this monitoring service updates the job repository entry in the middle-tier. Thus, even if the front-end client is closed immediately after job submission the job status is updated on the server independently and when the front-end client is reopened the updated job table is displayed by connecting to the middle-tier.

3.3.3 Other Stand-alone Clients

Use of JavaBean components with Java RMI assumes that the front-end client is developed in Java. Since GUI can be implemented using several programming and scripting tools it is also important to support such applications. To support such applications, the different services provided by ECS are deployed as Web Services. The front-end client uses HTTPS protocol for communication between the WSDL server as shown in figure 3.10. The WSDL server is running behind a firewall and the JavaBean components deployed as web services use Java RMI or RMI over IIOP to communicate between the WSDL server and the application server. Using this approach, users have to first authenticate with the WSDL server, obtain a handle to the requested service, and then invoke the appropriate methods on the service object. The web service in turn authenticates with the application server and then performs the corresponding services

on the application server including accessing the back-end resources. Users must have accounts on the WSDL server and the server authenticates user requests. The application server also performs further authentication and authorization to ensure that the user has necessary credentials and privileges to access the application server. Both grid and non-grid resources can be accessed using these clients similar to the front-end clients written in Java.

The advantage of using web services to support standalone clients is that services provided by ECS can be accessed from any client in a language and platform independent manner. Thus front-end GUI development can be done using any language on any platform. The WSDL interface for the ECS clients also provides a programmatic interface for accessing services and functionality provided by the ECS middle-tier and can also be used to provide interoperability with different portals. Third-party clients can make use of the ECS services as portlets inside customized portals or desktop applications. Users in the role of GCE developers can exploit these interfaces to develop domain specific problem solving environments.

3.3.4 Discussion

All the different types of front-end clients described above can be supported for a given distributed computing environment implemented using the architecture shown in Figure 3.1. There are several options available to access the middle-tier and the back-end using these clients based on the authentication and authorization mechanisms supported at the back-end and the language of implementation. Kerberos-based resources can be accessed only from the front-end (when credential delegation is impossible) while Grid-based resources can be accessed directly from the front-end as well as the middle-tier irrespective of whether the front-end client is browser-based or a desktop application. Desktop applications accessing Grid-based back-end resources through the middle-tier have two options: using Web Services and using Java RMI. If the desktop application is

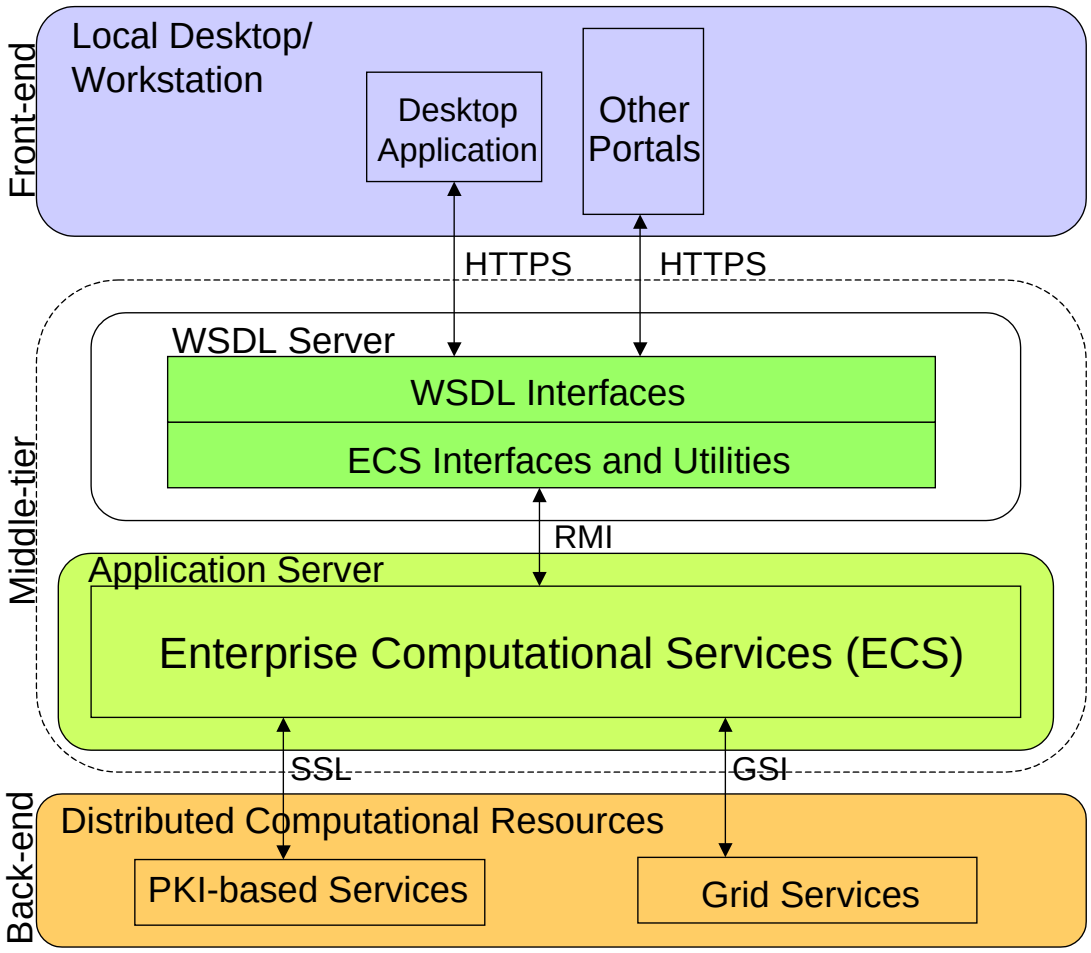


Figure 3.10: Illustration of ECS Framework supporting language and platform independent front-end clients

written in Java then using Java RMI is the clear choice whereas for applications written in other languages using Web Services is the best option. A front-end client written in Java can use RMI to access services implemented in Java and use Web Services to access third-party services or services implemented in other languages and platforms.

When Windows platform are used to develop clients (either browser based or desktop applications) using Windows specific programming languages and tools like Active Server Pages, Visual Basic, and so on, the ECS services can be accessed using the WSDL interfaces for the JavaBean components. Multiple clients can access the same underlying services provided by ECS irrespective of the language, platform, and type of front-end client. There is no modification of the ECS implementation either to support these different front-end clients or to support the different back-end resources. Thus the ECS framework provides separation of concerns and interoperability between different service providers irrespective of the implementation language and deployment platform.

3.4 Security

Traditionally users access HPC resources using command-line clients like *ssh* and *krsh*. Depending on the authentication and authorization mechanism employed at the host computer users have to provide username and password or valid credentials to obtain access to these resources. The authentication and authorization mechanisms are chosen by the resource providers and users do not have a choice as to which one to use. Typically the NSF supercomputing centers use PKI-based mechanisms while the Major Shared Resource Centers (MSRC) use Kerberos-based mechanisms for authentication and authorization. For secure access using Kerberos, users must have a valid ticket while using Globus, users must have a valid proxy certificate. In order to access these resources using a multi-tier architecture, should users access the back-end directly from the front-end then the same mechanism is employed. But when resources are accessed from the middle-tier on behalf of the user, then the middle-tier must have access to

user's credentials. Since GSI-based proxy certificates support delegation of credentials, the middle-tier can access Grid-based back-end resources on behalf of the user using the proxy certificate. When Kerberos authentication and authorization schemes are used in conjunction with a secure identification card to generate a one-time passcodes, credential delegation is difficult to achieve and thus the back-end resources are accessed directly from the front-end. Thus, there is no difference in accessing a Kerberos-based back-end either through command-line clients or ECS clients.

For accessing Grid-based resources from the middle-tier, there are two options available. The first approach involves the user creating a proxy certificate on his or her local desktop, uploading the certificate to the middle-tier, and then the middle-tier will access the back-end on user's behalf. Since a secure connection is used for communication between the front-end and the middle-tier there is no issue of compromising the proxy certificate. The disadvantage with this approach is that if the user is not at his or her local desktop that has access to users permanent credentials and unable to create and upload the proxy credentials then the user cannot access the back-end. In order to solve this problem and provide secure access from anywhere one can use the online credential repository such as MyProxy. Users can upload their credentials to the credentials repository and when required the middle-tier can request user's MyProxy passphrase and then retrieve the credentials from the server. To retrieve the credentials from the MyProxy server the middle-tier process must be executing as a user with a valid certificate and only this trusted middle-tier user can retrieve the certificate. Since the proxy certificate stored in the repository is encrypted using the MyProxy passphrase, then even if the MyProxy server is compromised it would require additional effort to decrypt the stored certificate.

In addition to ensuring that users credentials are not compromised, this architecture also provides several additional layers of security. First, users must authenticate with the front-end client which provides the first level security and then the user has to

authenticate with the application-tier to access any services provided by ECS. In order to access any methods provided by ECS users must have an account of the application-server with appropriate realms and authorization roles. When services are accessed through web services, users have to authenticate with the WSDL server first and then the application server. Finally to access back-end resources, valid user credentials are required. Also, all communication between the front-end and the middle-tier is performed through encrypted channels, thereby providing privacy in terms of data transferred.

3.5 Summary

The emphasis of this work is to develop a framework that can be used to create different domain specific DCSS and these systems can be realized either as browser based portals, standalone desktop applications, third-party systems or a combination of different clients based on the type of user. The extensible architecture employed here allows one to support different types of front-end clients in addition to supporting both grid-based and non-grid based back-end resources. The capabilities provided by the ECS framework used to validate the hypothesis of this dissertation are as follows:

- Using a role based user profile the ECS framework is able to support all the different types of users within a DCSS. This is the only framework, that the author is aware, that can support the comprehensive set of services and interfaces to support the diverse requirements of different users of modern computational systems.
- The ECS framework provides a unique approach for automatic creation of proxy objects using metadata based application repository (as described in section 3.2.1). These proxy objects support creation of complex multistep tasks out of standalone interdependent applications distributed across multiple organizations without modifying existing applications. Thus, ECS framework supports creation of distributed applications from standalone applications. The ECS framework relying

extensively on metadata about the application automates several tasks that would otherwise require human intervention, thereby improving user productivity.

- In addition to providing basic services to access distributed remote resources ECS provides high-level user services and persistent services. Persistent services are unique to this framework and are used to save, search, retrieve, and share applications, configuration information, and final products generated. Persistent services are also used to provide historic information and create pedigrees. The high-level user services enable the development of advanced problem solving environments to support advanced users as well as support the transition of applications from a development mode to an operational mode. The support for application transition from research and development to operational use is an important contribution.
- The ECS framework is based on the commodity component technologies supported by the Java 2 Enterprise Edition (J2EE). The J2EE platform is widely supported and several application servers are available, thus the ECS framework can be deployed on any J2EE complaint application server by only modifying the deployment settings. These application servers can be distributed across multiple servers to provide high availability, scalability, and fault tolerance. Therefore the services provided by ECS can be distributed as well, for example, users could select applications from multiple application metadata repositories hosted on distributed servers and create a task in the user workspace using another server. Individual services provided by ECS can be delivered through Portlets [97].
- Web Services (WSDL interfaces) provide a language neutral and platform independent method for accessing services provided by ECS from other third-party applications. ECS clients can be written in any programming language or development environment (Java, Visual Basic, C++, C#). Similarly any third-party service can also be integrated into the ECS framework. Thus, the ECS

framework supports interoperability between different computational portals or PSEs.

CHAPTER IV

IMPLEMENTATION DETAILS

The different services provide by the ECS framework are implemented using the Enterprise Java Beans (EJB) [4] and Java Bean components [24] supported by the Java 2 Enterprise Edition (J2EE) platform [5]. Entity beans are used to implement persistent services and Java Bean interfaces are used to access the EJBs instead of directly exposing the EJB interfaces. The Java Bean interfaces encapsulate the implementation details of the EJBs and developers of front-end clients are not required to be familiar with using EJBs. Also, the Java Bean interfaces are used to deploy services provided by the ECS framework as Web Services.

A hybrid approach of using the database and the eXtensible Markup Language (XML) [62] is followed to combine the advantages of the efficient search and query capabilities supported by the databases and the extensibility provided by XML. During the design and development of the application metadata by using XML we are able to modify and update the schema without adding new fields to the database, requiring recompiling and redeployment. Also the use of XML automates the task of GUI generation to gather the application metadata by applying XSL stylesheets [98]. When the schema is changed the corresponding stylesheets are updated and there is no need to add any code in the GUI to capture the new changes. By serializing the XML documents to a string and storing in the database we are able to provide better data management. By storing the key fields in the database be are able to support efficient search and query by localizing a particular object first and then extracting more detailed information about that object. Then an XML DOM object [62] is created in memory by loading the XML

string and DOM functions are used to traverse and extract required nodes, elements, and attributes. In this chapter the database schema for all the entity beans and the Java Bean interfaces for all services prototyped are described.

4.1 Application Repository

Application metadata is stored as the Model Entity Bean and the database schema for the Model object is shown in figure 4.1. The modelId is the primary key used to uniquely identify each application in the repository and this field is not directly exposed to the end-user. Typically the name field is provided by the user and internally the modelId is created for each application. The group field is used to organize applications into different groups and when an application is created in user's private space then the userId is used as the group. When an application is exported or published then the group will be set to "public" by default while applications in the shared area is set to "registered." If there are special user groups setup then during the export operation user has to select the specific group that should be allowed to access the application and then the group field will be set to that particular value. The modelInfo field contains the application metadata as an XML string and only application specific metadata is stored in this field since all machine specific information is stored in the Host Entity Bean. When a new application metadata is added to the repository only the name, group, and modelInfo fields are required whereas the modelId and dateCreated are assigned internally using the EJB creation.

The home interface (see figure 4.2) to the Model EJB provides methods to create the Model object and queries based on the attributes while the remote interface (see figure 4.3) provides methods to get and set individual model attributes. The Java Bean interface shown in figure 4.4 encapsulates the EJBs and provides methods that can be used by JSP developers to develop web-based clients. The same interface is also deployed

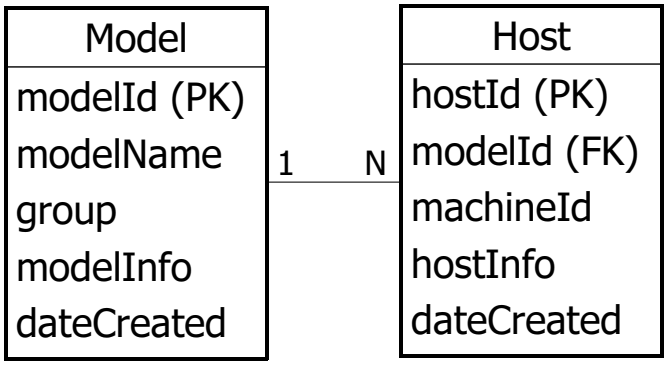


Figure 4.1: Illustration of the database schema for the Model and Host EJBs.

as a web service that can be accessed by clients using any programming language or development platform.

Any given application can be registered on multiple machines and for each machine an application is registered there is an entry in the Host table consisting of Host Entity Beans. The hostId is the primary key with the modelId as the foreign key. The hostname corresponds to the machine name defined in the resource repository and the hostInfo field contains a serialized XML string that represents the machine specific parameters for the given application. The advantage of storing machine specific parameters in a separate table is that it separates the application specific parameters from the machine specific parameters, thus when the application is registered on a new machine a Host EJB object is created without modifying the Model EJB. The Java Bean interface for accessing the Host EJB is provided in Appendix C.

Metadata is automatically generated using the application metadata XML schema (see sample application metadata in Appendix A) and the values provided by the user during application registration. By applying XSL stylesheets on the XML schema appropriate forms to capture the application metadata are generated. An XML DOM is created based on the schema and the values provided by the user are added to the appropriate nodes of the DOM. The DOM is serialized and converted to a string and stored in the appropriate info fields. The different steps involved in registering an

```
package ecs.ejb;

import java.util.Collection;
import java.rmi.RemoteException;
import javax.ejb.*;

public interface ModelHome extends EJBHome {

    public Model create(String modelName, String group,
                        String modelInfo)
           throws RemoteException, CreateException;
    public Model findByPrimaryKey(String modelId)
           throws FinderException, RemoteException;
    public Collection findByGroup(String group)
           throws FinderException, RemoteException;
    public Collection findAll()
           throws FinderException, RemoteException;
}
```

Figure 4.2: Home interface for the Model EJB

```
package ecs.ejb;

import javax.ejb.EJBObject;
import java.rmi.RemoteException;

public interface Model extends EJBObject {

    public String getModelId() throws RemoteException;
    public String getModelName() throws RemoteException;
    public void setModelName(String modelName) throws RemoteException;
    public String getGroup() throws RemoteException;
    public void setGroup(String group) throws RemoteException;
    public String getModelInfo() throws RemoteException;
    public void setModelInfo(String modelInfo) throws RemoteException;
    public long getDateCreated() throws RemoteException;
}
```

Figure 4.3: Remote interface for the Model EJB

```
package ecs.services;

import ecs.ejb.*;
import java.util.*;

public interface ModelInterface {

   public String newModel(String modelName, String group,
                          String modelInfo) throws Exception;
   public Model getModel(String modelId) throws Exception;
   public String getModelInfo(String modelId) throws Exception;
   public void setModelInfo(String modelId, String modelInfo)
          throws Exception;
   public void setModelName(String modelId, String modelName)
          throws Exception;
   public Collection getModelByGroup(String group) throws Exception;
   public Collection getAllModels() throws Exception;
   public String cloneModel(String newModelName, String newModelGroup,
                            String currentModelId) throws Exception;
   public void deleteModel(String modelId) throws Exception;
}
```

Figure 4.4: Java interface for accessing the application repository

application into the application repository are shown in figure 4.5. Users select the register application option through the front-end client. The request processing tier loads the XML schema, applies an XSL transformation, and generates a GUI to obtain the application metadata. Once the user has provided all the required information, the processing tier creates a DOM and then invokes the newModel method of the Model object. The Model Java Bean in-turn invokes the create method of the Model EJB and then returns the modelId to the processing tier. Since only application-specific information is stored in the Model EJB, machine-specific information is added to the Host EJB using the modelId and machine-specific information. Figure 4.6 illustrates the same operation when ModelBean is deployed as a web service. Once a handle to the Model object is obtained, all operations defined on the Model object can be performed using this handle.

4.2 User Workspace

The user workspace includes repositories for user, project, task, and application. All these repositories are implemented as entity beans and the corresponding database schema for the different EJBs is shown in figure 4.7. For each user the workspace contains a hierarchy of projects, tasks, and applications and this hierarchy is denoted by the one-to-many relationships between user and project, project and task, and task and application. Figure 4.7 illustrates the different fields used by the entity beans along with the primary and foreign keys used by each entity. The Java interfaces for accessing these EJBs are provided in Appendix C. When an application is selected from the application repository and added to a task, the corresponding application object is created. The applicationInfo field of the application object includes application-specific and the machine-specific metadata for an instance of an application. The taskInfo field in the task object has the information about the dependencies between different applications when multistep tasks are created.

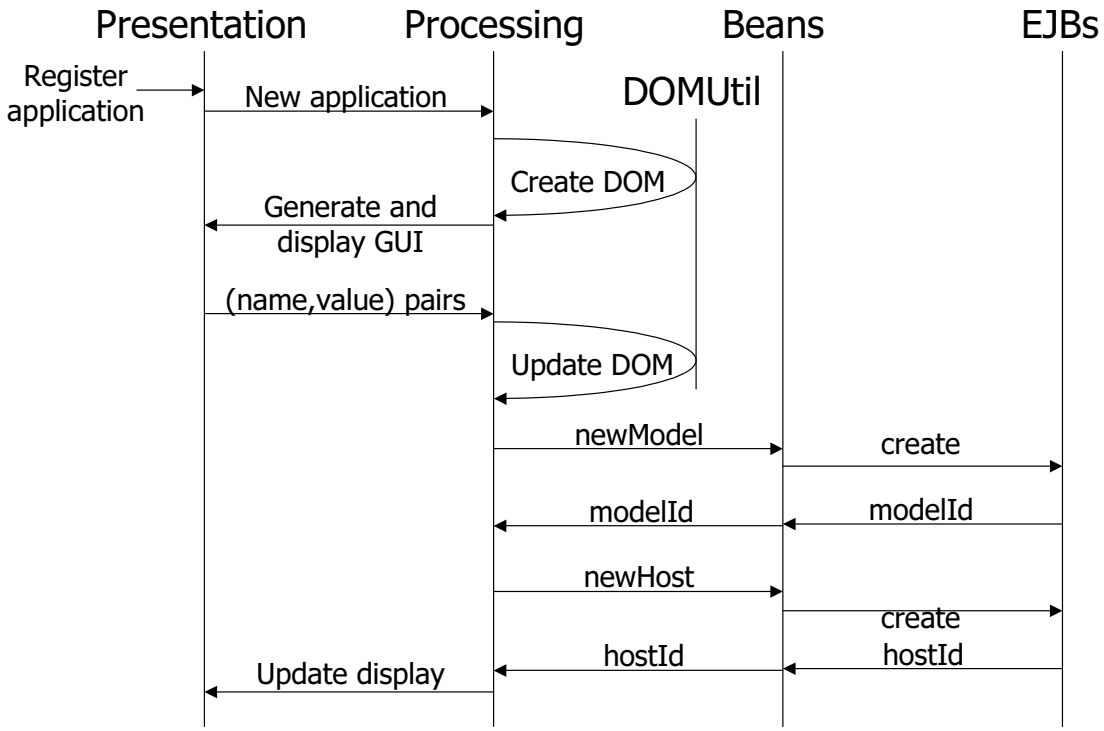


Figure 4.5: A sequence diagram to illustrate the interaction between the presentation, processing, Java Bean, and EJB layers

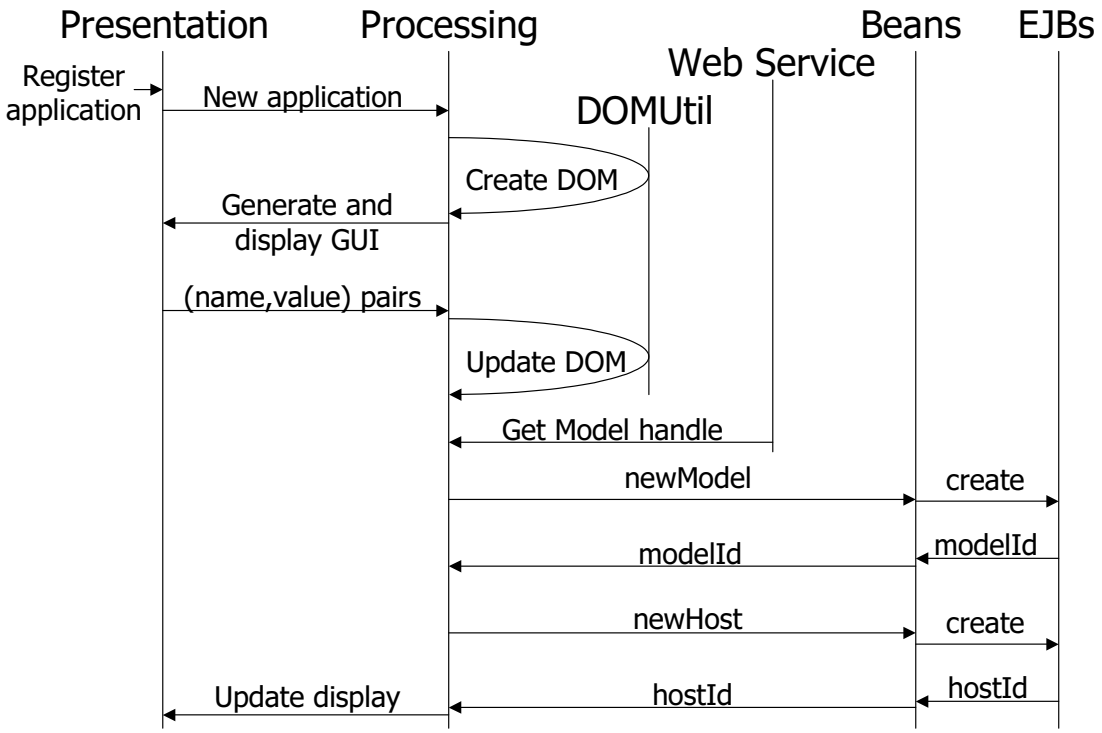


Figure 4.6: A sequence diagram to illustrate the interaction between different layers of the multi-tier architecture when services are deployed as web services

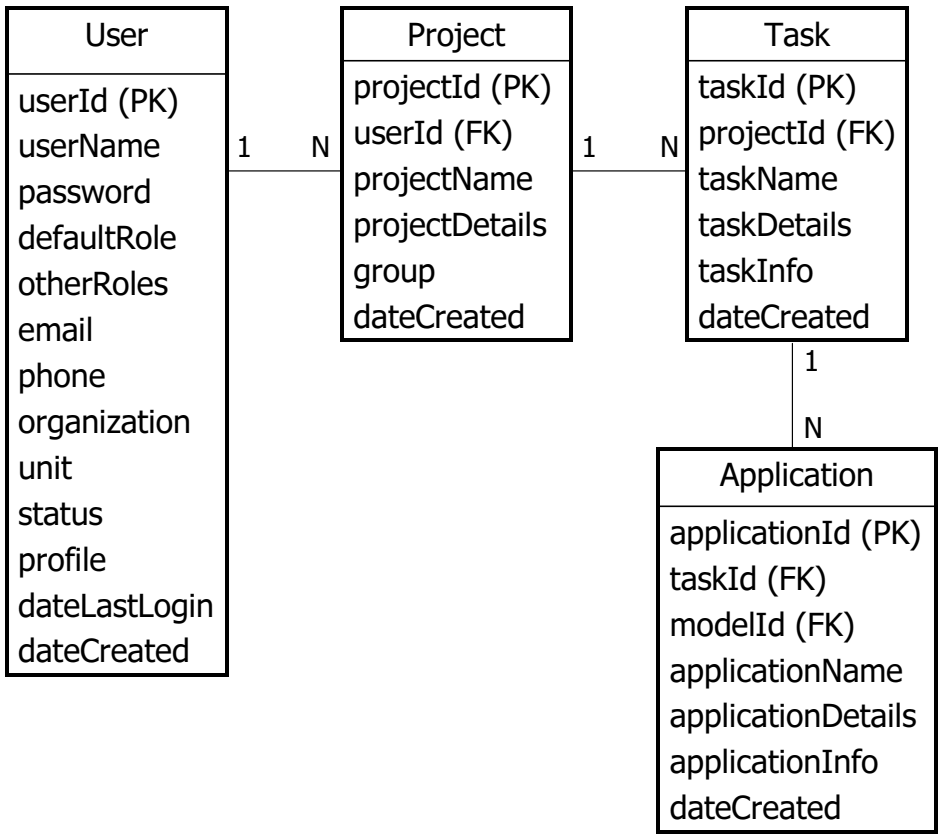


Figure 4.7: Illustration of the database schema for the different EJBs used to implement user workspace

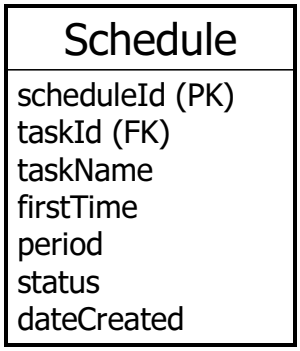


Figure 4.8: Illustration of the database schema for the schedule table

4.3 Workflow Manager and Scheduler

The workflow manager uses the taskInfo field of the task object to schedule the different applications that are part of a task as described in section 3.2.3. For advanced scheduling a schedule table is maintained and the database schema for the schedule table is shown in figure 4.8. Only scheduling of configured tasks is supported and the taskId is used to reference the specified task. The schedule table has additional fields to indicate when a task must be scheduled, how frequently the task must be scheduled, and current status of a scheduled task.

4.4 Job Repository

Job repository maintains proxy objects in the *active* or *complete* states. These proxy objects are created when applications are submitted directly from the application repository and when configured tasks in the user workspace are submitted. In order to maintain information about jobs submitted from both application repository and user workspace, the job table is maintained as an independent table with all the required workspace information. For submission from the application repository default workspace values are used. The database schema for the job table is shown in figure 4.9. The jobInfo field contains both application-dependent and machine-dependent

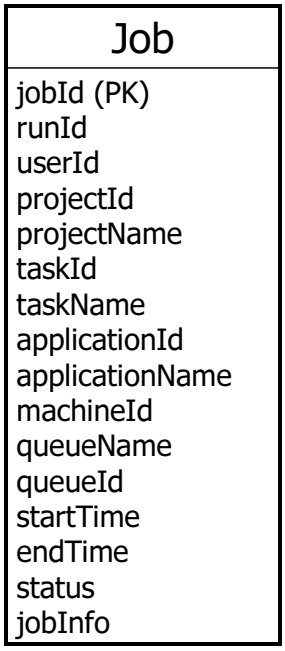


Figure 4.9: Illustration of the database schema for the job repository

configuration information in addition to pointers to standard input, output, and error streams and input and output files registered. The Java interfaces for accessing and manipulating job objects in the job repository is provided in Appendix C.

4.5 Deployment

The services provided by ECS are deployed as a J2EE application [5]. JSP-based web clients could be deployed as separate web components or combined with the EJBs to create a single J2EE application. Desktop clients written in Java use the Java Bean interfaces to access the EJBs using Java RMI. To support clients developed using other programming languages or development platforms (*e.g.,* C# and .NET) the Java Beans are deployed as web services. The WSDL [50] server generates the appropriate WSDL documents and SOAP messages that can be used by third-party clients to access the services provided by ECS.

CHAPTER V

USING ENTERPRISE COMPUTATIONAL SERVICES FRAMEWORK

This chapter describes two distributed computational simulation systems (DCSS) developed using the Enterprise Computational Services (ECS) framework. These systems are described here to illustrate how services provided by the ECS framework are used to satisfy the requirements of modern multidisciplinary computational simulations.

5.1 Distributed Marine Environment Forecast System

The Distributed Marine Environment Forecast system (DMEFS) is a research testbed for demonstrating the integration of various enabling technologies and components into an "open" framework in which validated climate, weather, and ocean (CWO) models are developed and transitioned to operational use. This section discusses how the ECS framework is used in the development of DMEFS while detailed description of DMEFS can be found in [17, 19, 90].

5.1.1 Requirements

Some of the key requirements of DMEFS are as follows:

- facilitate rapid development, integration, testing, validation, and deployment of various METeorology-OCeanography (METOC) meta-applications, such as the Navy Coastal Ocean Model (NCOM) [99], Coupled Ocean-Atmosphere Model Prediction System (COAMPS) [100], and the Wave Model (WAM) [101];

- provide a collaborative environment among diverse group of users, including computer specialists, model developers, and operational users;
- provide an integrated environment where applications can be validated, shared between users, and transitioned for operational use;
- provide the means of substantially reducing the time to develop, prototype, test, validate, and transition simulation models;
- provide a secure and controlled environment to manage computational resources owned and managed by different users;
- provide unified access to diverse collection of computational resources, interdependent standalone applications, and data sources while hiding complexity of the heterogeneous, distributed resources; and,
- simplify applications programming by establishing visual metaphors for distributed computing to capture the operational semantics of the different computational models while abstracting the data-sharing problem for users.

The system is also expected to evolve in function, user-interface, and methodologies, but be suitable for testing various approaches utilizing real data and configuration. It should be emphasized that the requirements of DMEFS is not building models, performing modeling research, or producing operational forecasts, but rather developing the underlying infrastructure to facilitate research and operations in a distributed HPC environment. By performing both the research and development and the operations of validated models within the same infrastructure, an immense amount of saving in time and money is expected in terms of sharing of data, common tools, documentation, training, and management.

5.1.2 Design

The requirements of a generic DCSS described in Chapter I captures all the requirements of DMEFS and thus the ECS framework has all the necessary services

in-place to meet the requirements of DMEFS. In Chapter III several options to develop DCSS based on the ECS framework were discussed. In order to support the diverse set of DMEFS users, both a browser-based front-end and a standalone desktop client are provided. Users can access DMEFS through both the front-end clients. A user working on his or her local workstation requiring access to visualization and analysis tools could use the standalone client. The same user while away from his or her primary system can use the browser-based front-end to access DMEFS. While the back-end systems used by DMEFS in an operational setting would be Kerberos-based, the system designed here supports both Globus and Kerberos-based back-ends. Figure 5.1 illustrates the multi-tier architecture of DMEFS based on the ECS framework. Different users and distributed computational resources are integrated through the ECS framework. This section describes how other requirements of DMEFS are realized using the ECS framework.

5.1.2.1 Supporting multiple users

Figure 5.2 provides a high-level view of the functionality provided by DMEFS required to support 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 executes validated codes to produce routine forecasts and might have limited knowledge on how the simulations actually work. It is critical to hide complexity of both the distributed computational environment and applications from the operator. The customer need not run models at all; instead he or she may access the results produced by the simulations. 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 configuration and management of users and resources. As the domain

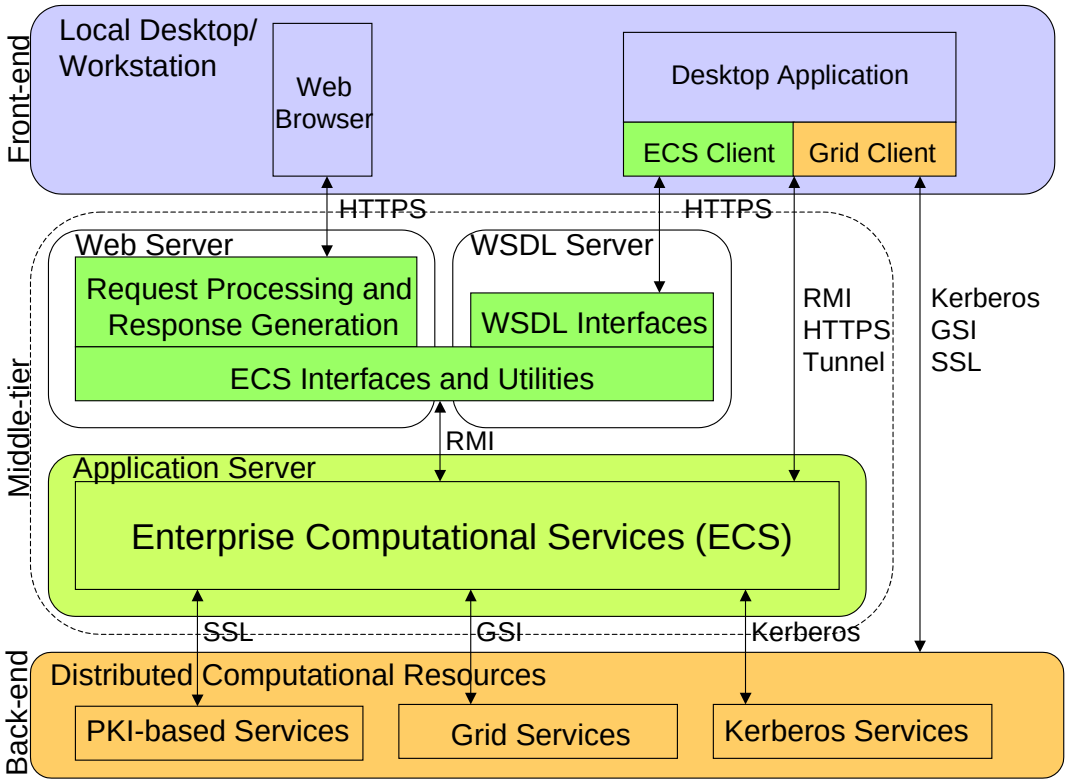


Figure 5.1: Multi-tier architecture of DMEFS based on the ECS framework supports multiple front-end clients along with diverse back-end computational resources

expertise level varies from one user category to another, the requirements of these users in terms of the functionality also varies correspondingly.

The functionality required by different users of DMEFS is illustrated in figure 5.3 [19]. Different categories of users are shown on the left-side of the diagram. The administrator manages users and resources. The developer activities involve management of his or her user space (creating, copying, sharing, and destroying projects and tasks), composing computational tasks, running them, and analyzing the results. To compose a task, the developer creates or selects a task context, adds applications descriptors to it, and specifies relationships between the constituent applications. The developer 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 developer 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. GUIs assist users with the application configuration process. Optionally, the developer may publish the task (list of configured applications and their relationships). This is a mechanism for transition of computational tasks from research and development to operations. The operator typically imports a published task, reconfigures it as needed, and executes 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.

All DMEFS users must first register and obtain accounts to access the system. During user registration, users can select the different roles that they are likely to assume and based on these selections when the system administrator approves the user account corresponding roles are assigned to users. A single user can be assigned multiple roles. Once the user logs in to DMEFS then different services based on the role assigned to the user are provided through the front-end user interfaces. If users are assigned multiple

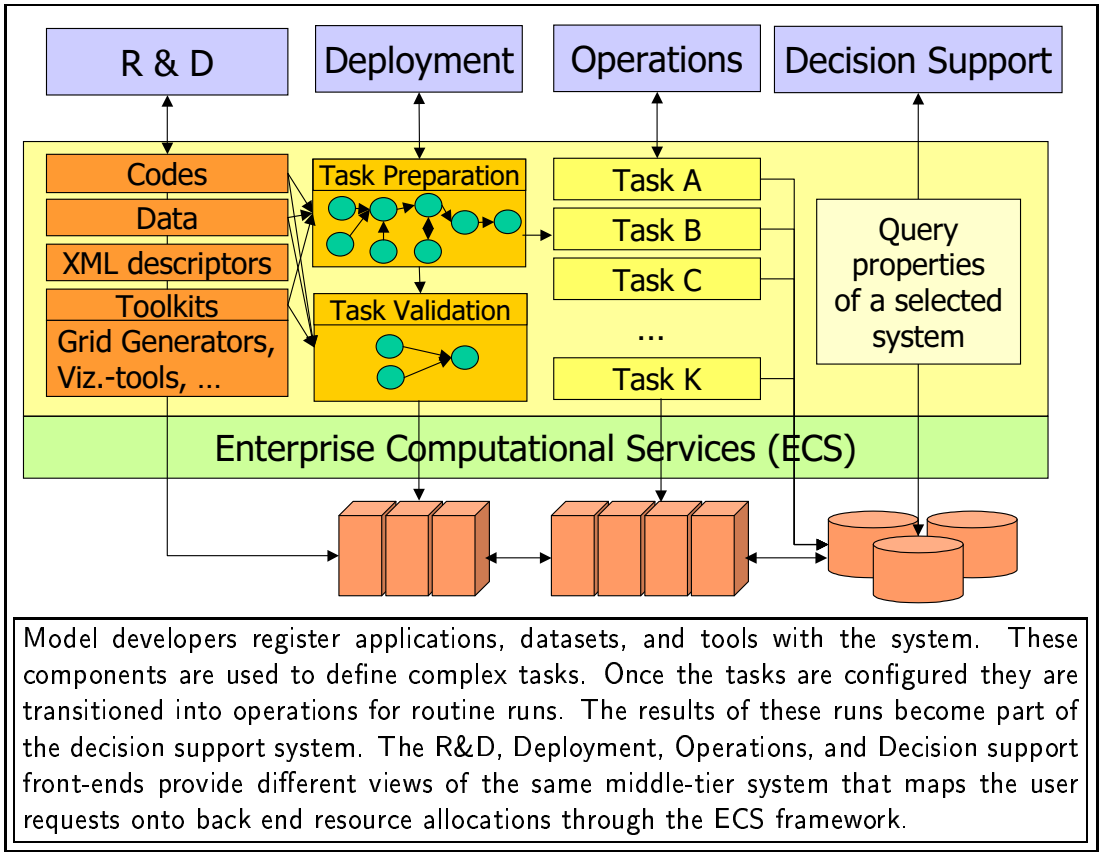


Figure 5.2: Overview of the functionality provided of DMEFS

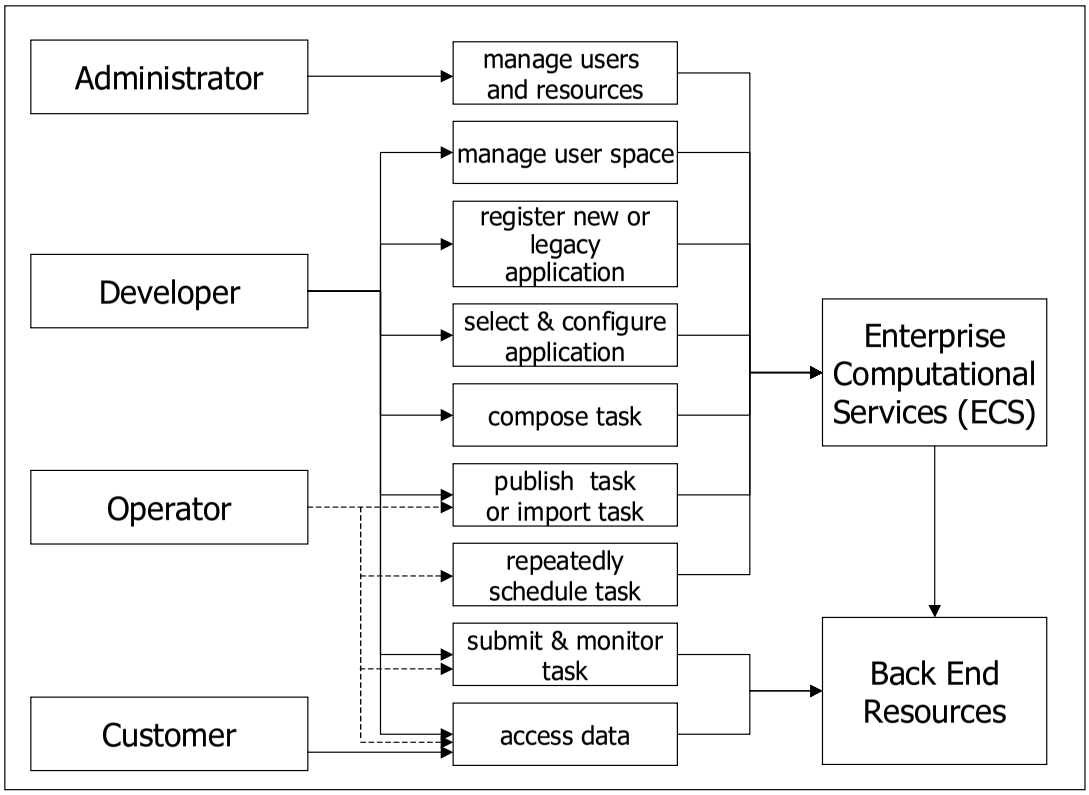


Figure 5.3: Illustration of functionality required by different users of DMEFS

roles they can change roles by selecting the appropriate role. Access control to different functionality is provided both at the presentation-tier and the application-tier. At the presentation-tier, appropriate buttons or icons are enabled or disabled based on user's role. At the application-tier each user is assigned to a realm, group, and role, and users cannot access methods that are not assigned to their roles. Thus, user role is used to provide access control and provide different user interfaces and services within a common infrastructure.

5.1.2.2 Executing METOC applications

Deploying and sharing various fully functional METOC applications within a common environment is one of the key requirements of DMEFS. In order to deploy and share applications using the ECS framework, application metadata is first gathered and then the metadata is saved in the application metadata repository. This metadata is used to configure applications, compose complex tasks involving several applications, stage necessary input files, submit the application, and unstage the output files. To determine if the metadata is sufficient to perform all these operations, a representative collection of applications are developed and used to evaluate the metadata. In addition to these applications three fully functional METOC applications, NCOM[99], COAMPS [100], and WAM [101] are used to validate the metadata-based approach followed by the ECS framework.

Executing an application typically involves submitting a machine-specific executable or a shell-script. The application reads data from input files and writes data back to output files. Input files are classified into parameter files and data files. Parameter files are ASCII files used to provide different options or switches for specifying application characteristics while data files are often binary files and provide data for initial conditions, boundary conditions, and other physical properties. For the purposes of capturing application metadata, applications are classified into following categories:

1. simple executable
2. executable with command line options
3. executable which requires setting environment variables before execution
4. executable with parameter files where the parameter filenames are predefined by the application
5. executable with parameter files where the parameter filenames are provided as command-line arguments
6. executable and parameters provided in a single shell script

For each of the above mentioned categories multiple input and/or output files could be used by the application and several such applications could be composed together as a task by defining the data dependencies between individual applications. Sample applications belonging to each category were developed and registered in the application metadata repository using the front-end GUIs provided by DMEFS. During application registration users can register different parameters files and for each file register different parameters that could be changed during configuration along with default values. Forms are automatically generated during configuration to accept values for these parameters. It is also possible to register customized GUIs for reading different parameter values and appropriate GUI will be displayed during configuration. In addition to parameter values other machine-specific information such as working directory, location of specific input/output files, and batch queue related values are also obtained during configuration. While the application-specific information is used to generate automatic GUIs for obtaining parameter values, the machine-specific information is used to generate appropriate batch scripts based on the target system selected.

Input files required by the application along with the expected location of these files and locations from where these files are to be obtained is also stored in the application metadata. Similarly each output file that will be generated is also registered along with the information about the location of the output files and the final destination where

the output files must be copied at the end of the simulation. While input files could come from another simulation or a data repository, output files could be copied to an archive or users local desktop for post-processing or as input to another simulation. This information is used to stage and unstage files automatically without requiring user intervention. This configuration information is also stored as part of the job object and could be used to determine what results were generated, when they were generated, how they were generated, what input parameters and files were used, and so on.

Using a registered application users can add new machines (assuming that the application can execute on that machine) or share the application with other users. When an existing application is registered on a new machine only the corresponding machine-specific parameters are provided. There is no need for the user to develop any machine-specific batch scripts or batch queue commands. Using the information from the resource repository the access mechanism for job submission, file transfer, and batch queue details are obtained. For Globus-based back-end resources RSL string is generated while appropriate batch scripts are generated for Kerberos-based back-ends.

The three METOC models were successfully registered and tested within DMEFS environment. The individual applications were not modified in any way for the purposes of registration, configuration, or submission. When these applications were instantiated multiple times by the same user with the same set of parameters often times they overwrite the earlier outputs. Even though this may be acceptable within single user systems, when an application is shared by multiple users, each user must be able to specify individual input sources and output destinations. If applications expect input files to be in specific locations with respect to the executable and this location cannot be specified through command-line arguments or parameters in a parameter file or environment variables then all the required input files must be registered explicitly, so that these files are copied to appropriate locations during file staging and unstaging. By modifying the scripts and by providing appropriate metadata during registration it is

possible to avoid unnecessary file copies when input files can be shared between users when the application is shared. Thus, DMEFS promotes sharing of applications and data between multiple users even though applications were not designed a priori to be used in a multiuser environment.

5.1.2.3 Connecting multiple standalone applications

One of the requirements of DMEFS is the capability to compose complex applications using a collection of interdependent standalone applications. This functionality is provided by DMEFS using the application repository, user workspace, and workflow manager services provided by the ECS framework. Individual applications are registered through the DMEFS front-end interfaces and stored in the application repository as abstract applications. These abstract applications are selected, configured, and added to the user workspace as a collection of application proxy objects (ready applications) to create a task object. GUIs with default values provided during registration are automatically generated using the application metadata to help users with the configuration process. The dependencies between the individual applications are represented in the task descriptor and the task thus created can be submitted for execution. The workflow manager schedules the appropriate application for execution on the specific back-end resource and performs necessary staging and unstaging of files as described by the application metadata. Thus, users are able to compose complex applications from distributed standalone applications, configure individual applications, submit applications on distributed computational resources, and monitor the progress using the GUIs provided by DMEFS.

A task can be submitted multiple times (with or without modifying the present configuration) and for each instance of a submitted application a job object is created and the job object can be used to monitor job status along with the configuration information and pointers to input and output files. Users can monitor the status of different

applications executing on different machines through a single snapshot provided by the job repository service. There is no need for users to log in to different back-end resources to submit jobs or monitor their status. All the required information is available from a single place through the front-end GUI. Users can even access the output files produced during the execution by selecting a specific job. Since the task and the corresponding applications with their configurations are stored persistently in the user workspace, users can resubmit these tasks with new application-specific values or machine-specific values. Thus, DMEFS provides automation, reusability, and repeatability for routine operations performed by different users.

5.1.2.4 Reuse, Sharing, and Transition from development to production

Once a task is composed, executed, tested, and validated users can share that task along with the corresponding configurations, input datasets, and output results by exporting or publishing the task. Other authorized users can then import that task to their private workspace, submit the task (with or without modifying the configuration), and analyze the results produced. Since configuring METOC applications requires extensive domain knowledge and familiarity with the application this process of exporting a preconfigured and validated application reduces the effort required to install, configure, and validate applications. This functionality is also used to transition an application from research and development to operational use. The operator can import a validated task and perform routine simulations without being concerned about downloading, upgrading, installing, compiling, or configuring validated models. Since both the researcher and the operator are accessing these applications through the same infrastructure, DMEFS provides a common platform for application development, validation, and transition to operations. Thus DMEFS provides a mechanism to reduce the time required to design, prototype, test, validate, and transition simulation models by reducing the time required to learn how to configure and execute validated applications.

5.1.2.5 Access to data produced by simulations

The configuration information stored as part of the application metadata contains description of the various products generated after the successful execution of the applications. This information is used to provide access to different products generated by computational simulations performed through DMEFS. Users can select a project, task, or application and obtain the list of products generated based on the selection. Users could select an application from the metadata repository and obtain the list of products generated by that particular application. In addition to providing a list of available files, for output files written in NetCDF [93] format, users can preview the metadata stored in the file without copying the file to user's desktop. The capability to browse a data file at the data source eliminates unnecessary data copy and reduces the time required to search and retrieve required data files. Future versions of DMEFS could provide additional operations such as sub-sampling and format conversions on the remote data, thereby providing the capability to perform remote operations on datasets. The application metadata has place-holders for linking datatypes and metadata for datasets and this information could be used to identify semantic mismatches in datasets as well as perform predefined remote operations on the data.

5.1.3 User Interfaces

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 [19]. Java Server Pages (JSP) technology, including custom tags and JavaBeans, are used to generate dynamic, context sensitive web content.

The challenge in design of browser-based 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, the amount of information simultaneously displayed on the

screen must be maximized while keeping the interface intuitive and minimizing the page refresh time and this is achieved through the use of HTML frames (see Figure 5.4). The upper frame provides access to a set of common portal services: remote file browser, status of the user jobs, status of machines, user log, help, and sign-out of a DMEFS session. The bottom frame provides an automatically refreshing short status of the machines accessible thorough the portal and a placeholder for system messages (none in this particular screen dump). The middle frame is split into three: the left frame exposes the interface of the user workspace, the right frame shows the interface of the application metadata, and finally the center frame provides the interface of the workflow manager and scheduler subsystem. The user creates or selects a task using the workspace interface, creates or selects application descriptors using the metadata interface, and composes and configures a computational task using the workflow manager interface. If the user is assigned other roles then user can change role by clicking on one of the allowed roles and a new interface with the functionality available in that role is displayed.

To support advanced users who require access to local desktop resources and visualization tools a Java based desktop application is also provided. The desktop application can access services provided by ECS using Java-RMI or through the WSDL interfaces. Since the desktop application is written in Java, using Java-RMI seems to be a natural choice but the ability to use WSDL interfaces to ECS illustrates that front-end clients written using other programming languages like C++ and non-Java platforms like .NET and C# can also exploit the services provided by the ECS framework.

5.1.4 Summary

DMEFS fully exploits the services provided by the ECS framework to provide a comprehensive set of features to support diverse set of users with multiple user interfaces, provides an integration environment for developing, testing, validating, and transitioning METOC models, connects multiple standalone applications executing

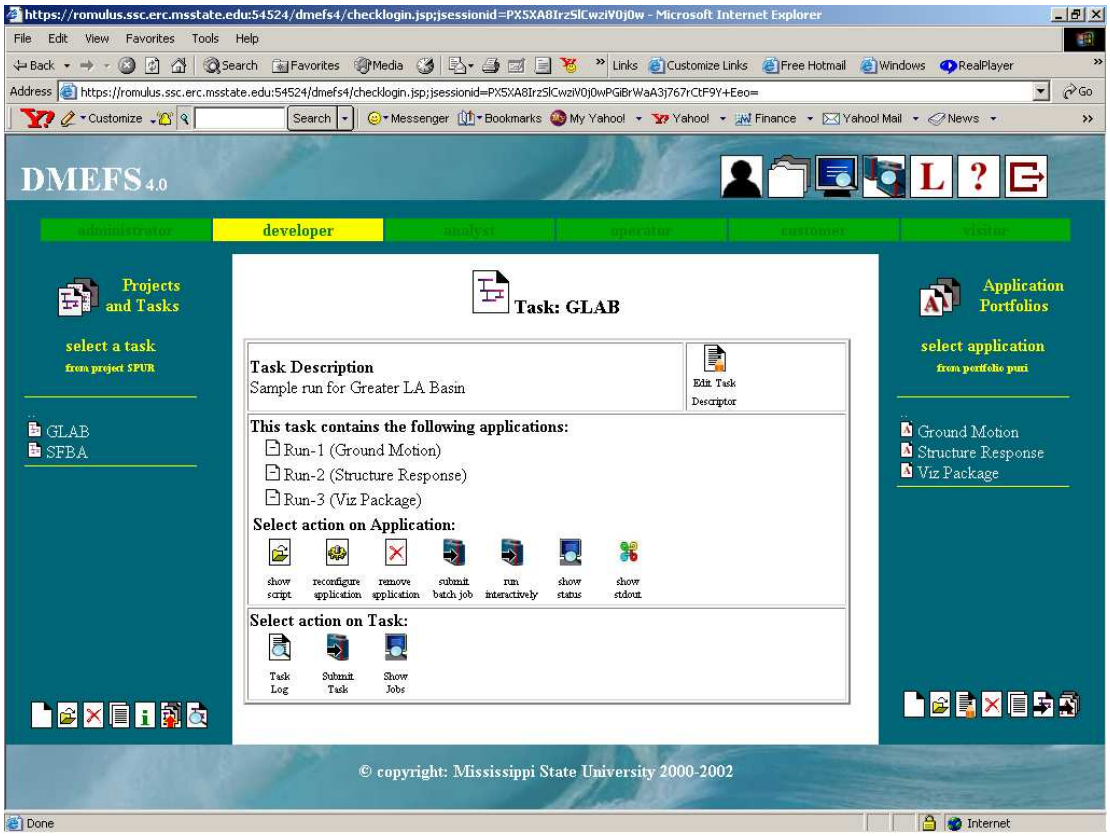


Figure 5.4: A snapshot of the user interface for the user in the developer role in DMEFS

on geographically distributed computational resources, promotes reuse and sharing of applications and configuration information, and provides persistent access to inputs used and outputs produced by computational simulations. Thus, DMEFS provides a collaborative environment for diverse group of users with interfaces and services based on their skill-level and expertise.

5.2 Distributed Simulation Framework for Seismic Performance of 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 techniques. A distributed interactive simulation framework is developed to facilitate investigation of the performance of urban regions resulting from major earthquakes and educating 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. As a large-scale simulation system this could become a service for NSF's Network for Earthquake Engineering Simulation (NEES) [102] program. SPUR could provide an important link between NEES structural simulation resources aimed at single structures and the goal of improving earthquake performance of the built infrastructure. In addition, urban region damage simulation and visualization capabilities of SPUR could be validated using field data acquired by NEES geotechnical and structural laboratories. SPUR would take advantage of the anticipated NEES computational, data, network, and collaboratory services.

The present process involved in simulating the ground motion and structural response and visualization of the combined effects is shown in figure 5.5. The three main

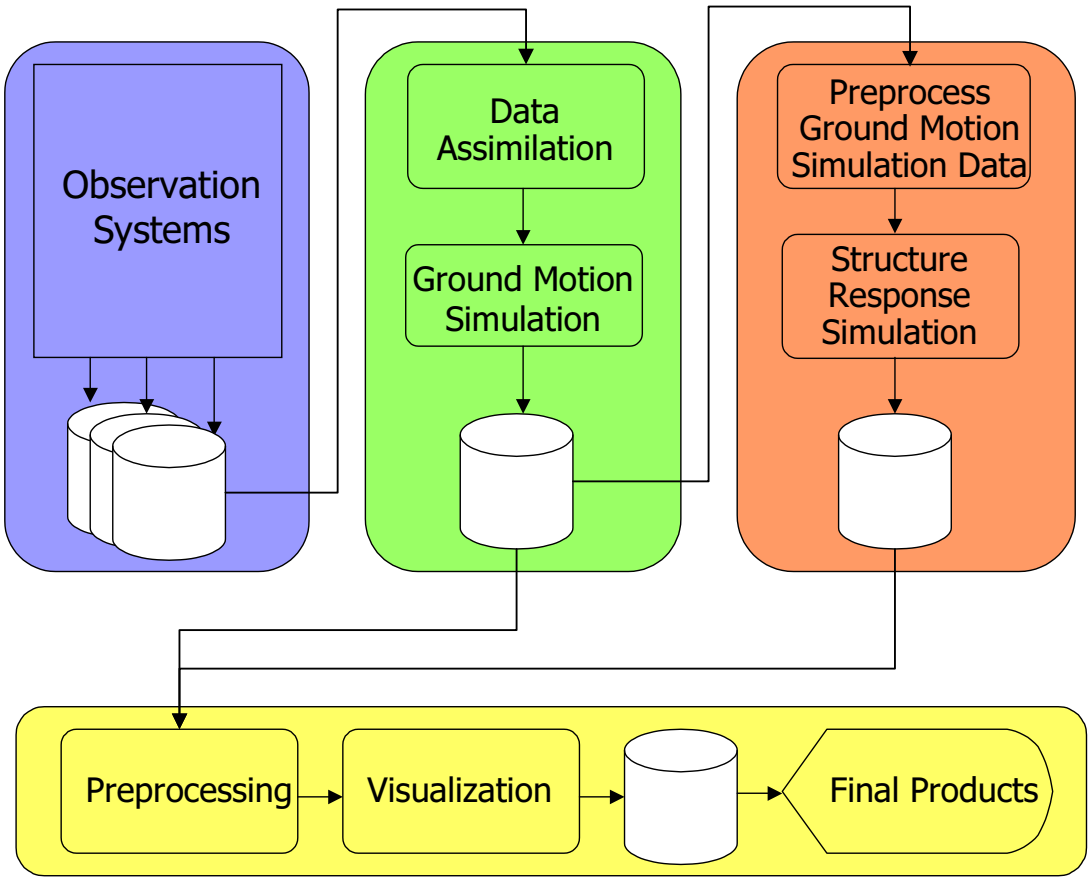


Figure 5.5: Illustration of present simulation system for SPUR

components of this simulation, ground motion modeling, structure response modeling, and visualization are performed by research groups from three different organizations. Ground motion simulation uses data from observation systems or data inventories and the results of the ground motion simulation are used by the structure response simulation. The results from the simulation of ground motion and structure response are used by the visualization application to generate final products which could be images, animations, or walk-throughs in an immersive environment. This approach has several drawbacks and they are as follows:

- to obtain the final product(s) several steps are required and each of these steps are performed manually;
- human intervention and co-ordination (technical expertise) required during each step;
- copying large datasets takes time and space and also data is duplicated at several locations;
- requires access to several heterogeneous computational resources that are geographically distributed and belongs to different organizations;
- simulation system not suitable for use by a consumer who may not have any knowledge about computational simulations (for example, decision-makers); and,
- final products produced do not contain information about how the products were generated (no metadata associated with the final products).

Thus, there is need for a distributed simulation framework for SPUR in order to solve some of the above mentioned problems associated with the present SPUR simulation system. Figure 5.6 provides a high-level view of the distributed simulation system for SPUR. This distributed system encapsulates the distributed components and provides a unified view of the complete system using customized front-end clients. Using the front-end interfaces users can configure and execute simulations, access visualization results, or browse different products available as a result of various simulations and

visualizations. The requirements for such a distributed simulation framework for SPUR are as follows [18]:

- hide complexities of distributed computing and automate routine tasks for performing simulations;
- provide a collaborative environment for exchange of information between different user groups;
- provide an environment that will
  - support diverse set of users (researchers, decision-makers, general public),
  - provide seamless access to distributed computational resources belonging to different organizations (file transfer, remote job execution and monitoring), and
  - provide a PSE where complex applications can be composed from standalone applications and the process of generating the final product is automated;
- provide an environment that integrates observation and simulation capabilities to create a "SPUR virtual laboratory"

The distributed simulation system for SPUR is designed as a multi-tier, computational web portal (see section 3.3.1) using the ECS framework as shown in Figure 5.7. The browser-based front-end allows users to select and visualize input data coming from ground motion simulations, specify parameters of the structures to be studied, execute the structure response simulation, specify the parameters for visualization, and visualize the outcome. User requests are processed and responses are generated by the presentation-tier while the application-tier provides the various middle-tier services such as persistent services, high-level user services, and remote access services. Distributed simulation and visualization applications along with the computational resources and data sources form the back-end tier.

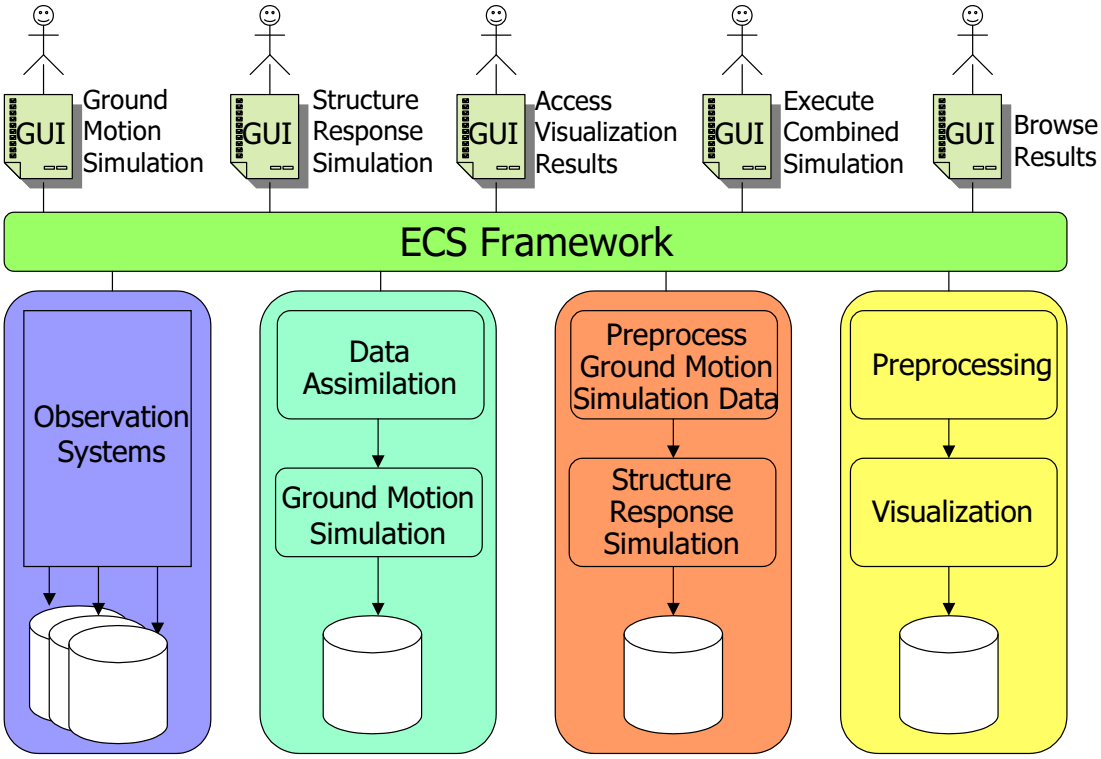


Figure 5.6: Distributed simulation system for SPUR

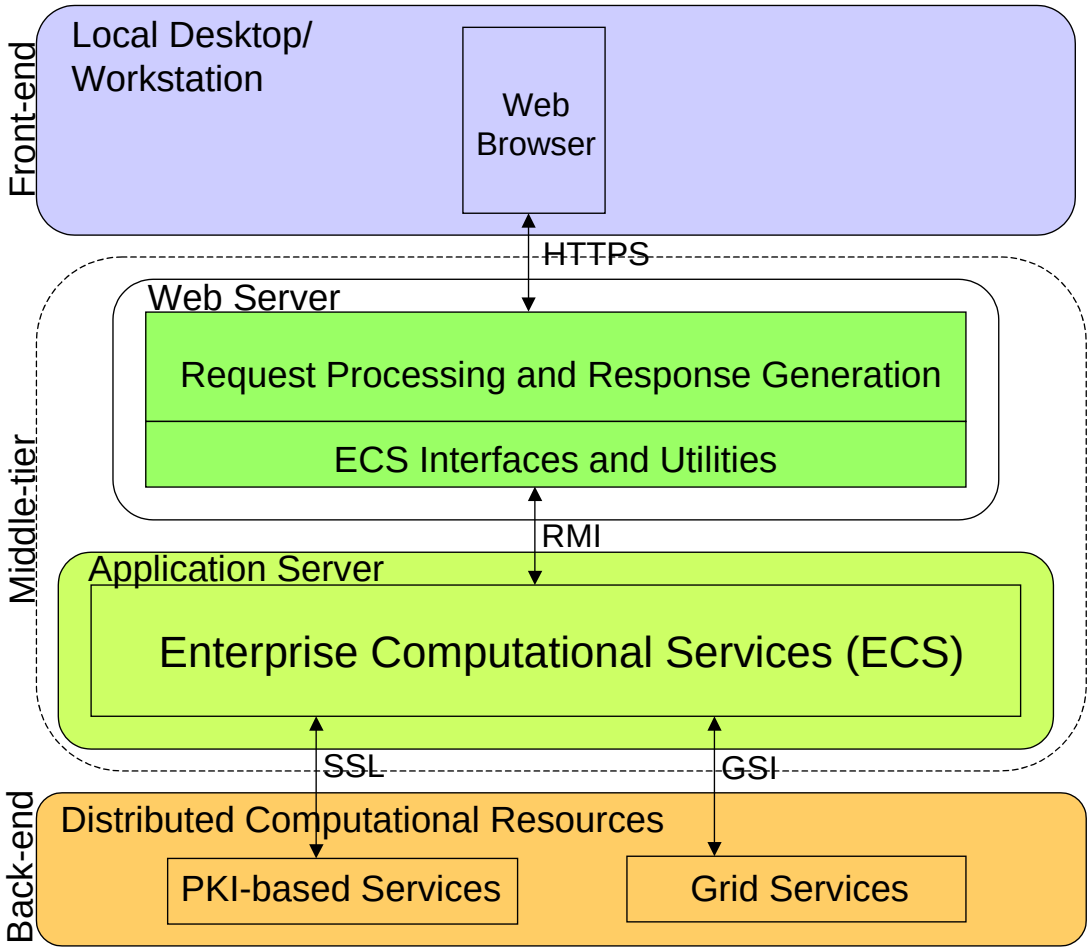


Figure 5.7: Multi-tier architecture for the distributed simulation system for SPUR based on the ECS framework

The initial implementation of the distributed system for SPUR uses the visualization of ground motion simulation as the starting point of the simulation [103, 104]. Open System for Earthquake Engineering Simulation (OpenSees) [105] software is used for performing the structure response simulation. The structure response simulation is setup by advanced users using TCL scripts or C/C++ drivers to use the OpenSees library. The GUI provided by the front-end (figure 5.8) allows the user to display the ground motion simulation images as an animation and the user can select specific time steps and the subregion to be used for the structure response simulation along with the spacing of the grid points for the selected region and the back-end machine where the OpenSees library is installed and simulation will be executed. Before executing the structure response simulation, users can also select the format of the visualization output (GIF, animated GIF, or MPEG movie), icon type to represent the building (sticks, sticks with square tops, wireframe with square top, semitransparent building with square top, or solid building with square top), and view angle (top full view, top and zoomed in center view, tilted aerial view of whole field, or tilted aerial view of central zoomed in portion) [104]. Once simulation and visualization are complete users can view the output produced and outputs from each run are saved separately in the user workspace. Users could select different regions for the simulation or for a selected region select different visualization parameters.

The SPUR portal allows user to setup a script to performing structure response simulation using OpenSees, display execute the structure response simulation, view results produced from earlier runs, and check the status of all jobs. Thus, the SPUR portal combines ground motion simulation results with the structure response simulation and visualization. The user workspace, task composition, and job submission services provided by ECS framework are used by the portal. The completed SPUR portal would act as an integration system for combining different distributed applications (ground

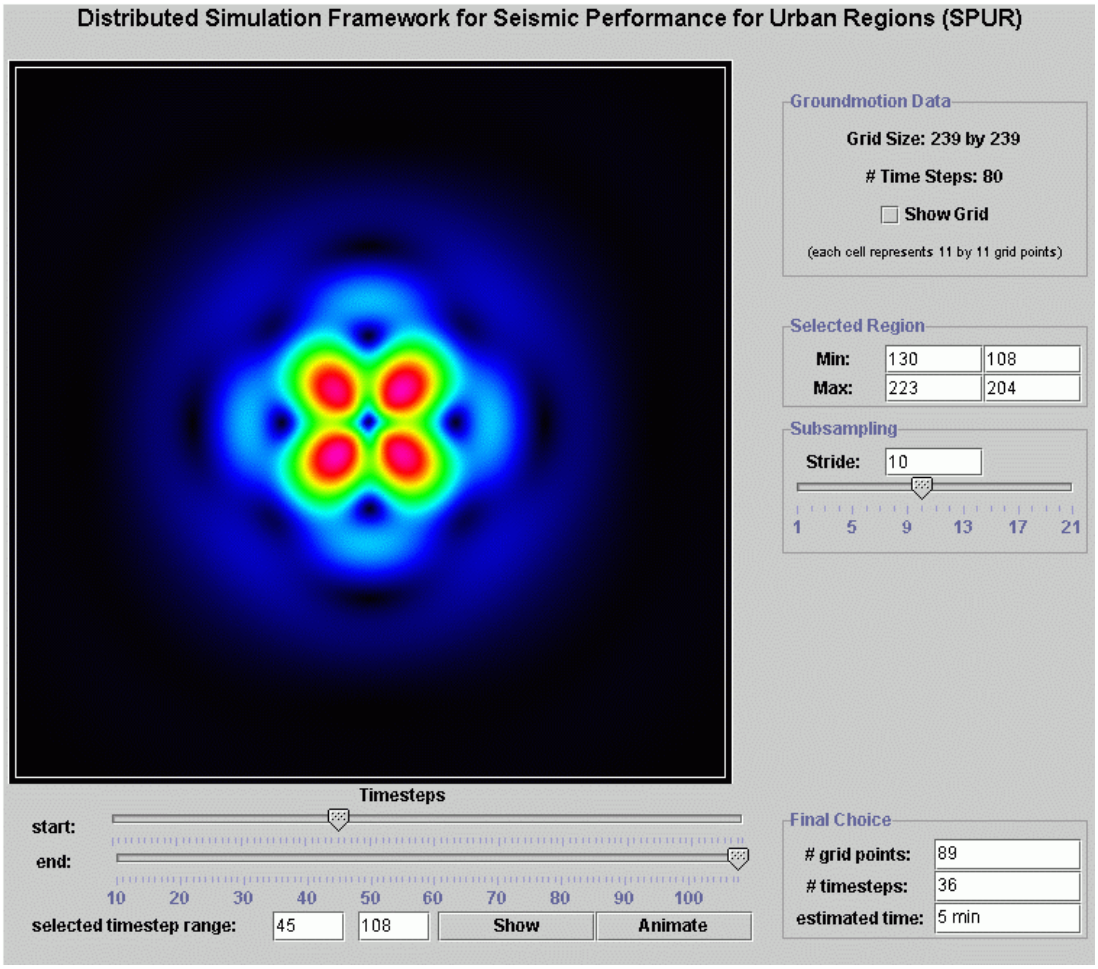


Figure 5.8: Applet used to display the ground motion simulation and select subregion for structure response simulation

motion simulation, structure response simulation, and advanced visualization) with the computational, data, network, and collaboratory services to be provided by NEES.

5.3 Constructive Proof

The ECS framework described in this dissertation is used as the basis for developing distributed computational simulation systems (DCSS), DMEFS and SPUR described in this chapter being two such examples. These DCSS provide a common infrastructure for different users to collaborate and exchange required information. Section 5.1 described how multiple front-end user interfaces to a single distributed system were developed using the ECS framework. These front-end interfaces could be implemented in any language or platform. Even though several services provided by the framework were implemented to show the functionality and capability, if these services are provided by a third-party they can be used by a DCSS developed using the ECS framework. Therefore the framework is open, extensible, and can evolve if new requirements are added. Figure 5.9 illustrates how the ECS framework provides a common infrastructure that integrates different users with distributed computational resources, applications, and datasets. A standalone application A residing on a machine X is represented by ECS as a proxy object A_X in the corresponding user space and several such proxy objects are created for different standalone application to form a distributed task. This task is saved and shared with other users.

A DCSS developed using the ECS framework supports integration of users by providing customized interfaces and services based on the users role. Using the application metadata distributed standalone applications are connected together to work together as a single task. The application metadata is also used to automate routine tasks of staging and unstaging data, generation of batch scripts, and job submission. The user workspace supports reuse and sharing of configured tasks along with the associated data and configuration information. Other persistence services allow users to access

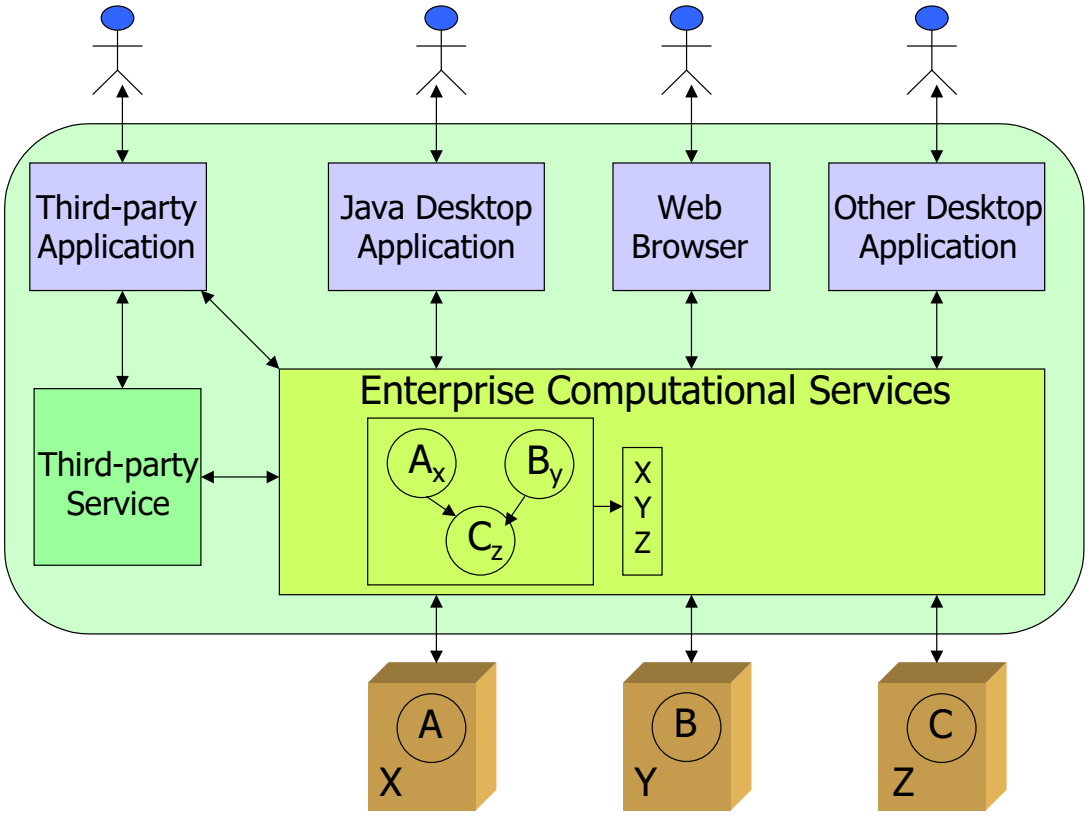


Figure 5.9: Illustration of how the ECS framework is used create a DCSS that integrates different users with distributed computational resources, applications, and datasets

historic information, provide annotation for results produced, and provide the capability to repeat computational simulations. An advanced user can perform development, testing, and validation and then share the application with other users. Novice users can import preconfigured and validated applications and use them without requiring extensive training to use and configure the application. Applications can be configured and submitted to multiple distributed resources from one place. Similarly applications executing on multiple platforms and status of distributed machines are monitored using the DCSS front-ends.

All these new functionality provided by a DCSS reduce the time required to perform routine operations by automating some these tasks (human intervention is reduced), support reuse and sharing of application configuration and data (learning time is reduced), and provides a common place for obtaining all information required to perform computational simulations. The shorter the time needed to complete any of the routine operations such application configuration, job submission, job monitoring, task creation, and task sharing results in higher user productivity.

The new functionality provided by DCSS does not compromise the functionality, performance, or security of existing applications. Since existing applications are not modified, there is no loss in current functionality, in fact the DCSS provides new functionality which could benefit the user. There is also no impact on the performance since the applications are submitted for execution as is without any modification. For example, when multiple standalone applications are used to compose complex tasks, the DCSS reduces human intervention required to stage and unstage data, monitor status, and submit applications. This could in fact result in overall better performance and is never worse than performing these tasks manually. Even while monitoring status of different machines and submitting individual applications the time required to log in to each machine, determine the status, and then submit the application would be reduced through the use of a DCSS, since all the required information is available at one place

for the user to make the decision or the resource broker can decide which machine to submit the given application.

In order to access the back-end resources a DCSS relies on the existing security mechanism the user is currently using. Therefore the security mechanism used to access back-end resources with and without a DCSS is the same. A DCSS based on the ECS framework provides several additional layers of security in terms of accessing the middle-tier services which can be used to selectively enable or disable certain features based on users role. Since only proxy objects are maintained in the middle-tier and users still requires valid credentials to access the back-end resources, even if the middle-tier is compromised actual applications and data that reside on the back-end are not compromised.

The ECS framework presented in this dissertation is used to prototype DCSS that satisfy the requirements of modern multidisciplinary computational simulations by integrating disparate users, resources, and applications without compromising the functionality, performance, or security of existing applications. Thus, the hypothesis is proved constructively by first prototyping the framework and then prototyping two DCSS using this framework. A DCSS provides additional functionality that could potentially reduce time spent by users to learn and perform routine computational simulations and the reduction in time in-turn results in better user productivity.

CHAPTER VI

SUMMARY AND CONCLUSIONS

Modern computational simulations involve interaction between a variety of distributed entities including computational resources, applications, and users. Chapter I described the drawbacks of present computational simulation systems and explained the requirements for developing modern distributed computational simulation systems (DCSS). Some of the key requirements are as follows:

- reduce user intervention through automation of routine computational tasks,
- provide a distributed computing paradigm suitable for multidisciplinary computational simulations by connecting interdependent standalone applications,
- provide secure access to distributed computational resources,
- support diverse set of users ranging from domain experts to decision-makers within a common infrastructure,
- provide the capability to annotate applications, the inputs they use, and the outputs they produce,
- provide the capability to save, reuse, and share applications, datasets, and configurations used to perform computational simulations, and

Based on these requirements it was evident that there is a need for a distributed computing environment suitable for users of multidisciplinary simulation to connect disparate applications, datasets, products, resources, and users into single user-friendly system while hiding the complexities of distributed computing from the users of this system. Since it is not practical to create a single distributed system that is

suitable for all computational domains, the common features required for developing DCSS were abstracted and a components-based open framework, called the Enterprise Computational Services (ECS) Framework, was created to develop different domain specific DCSEs. The ECS framework was developed using a novel combination of Grid, Internet, and Distributed computing technologies to connect standalone software components or applications into a single user-friendly system while reducing the complexities of multi-site, multi-program, multi-resource, multi-organizational computing. Internet technologies were used to provide a user-friendly front-end, a problem solving environment (PSE), and discover and lookup service providers and services, while Grid technologies formed the basis for accessing geographically distributed resources (this includes hardware, software, and data) securely. The distributed object/component technologies were used to develop a reusable distributed middleware that provides high-level user services, persistence services, and remote access services.

The ECS framework is based on the Java 2 Enterprise Edition (J2EE) [5] multi-tier architecture and Web Services [7]. The multi-tier architecture is logically divided into three tiers: front-end, middle-tier, and back-end [5]. The multi-tier architecture encapsulates distributed computational resources and applications in the back-end and hides the implementation details of the middle-tier from the user by providing user-friendly front-end interfaces. The front-end provides easy-to-use graphical user interfaces and programmatic interfaces to access the back-end resources. The front-end client can be a web browser, standalone application (written in Java, C++, or Visual Basic), or a .NET application. The ECS framework accommodates multiple clients appropriate for different types of users within a single distributed computational environment.

A reusable components-based middle-tier provides a common service layer to support these multiple front-ends clients and diverse back-end resources. The middle-tier performs all the necessary translation of user requests to appropriate actions in the middle-tier in addition to providing a set of unique services not supported by existing

transient Grid services. Persistence services provide repositories for proxy objects created by ECS for application metadata, users, projects, tasks, applications, jobs, and resources. Remote access services exploit grid services to perform job submission and monitoring, file transfer, and obtain machine status while utilizing the persistence services to obtain information required for job submission, file transfer, and resource description. High-level users services like user workspace, workflow manager, scheduler, and resource broker use remote access services and persistence services to provide high-level abstractions to the user by hiding the details about the repositories and access mechanisms. All three groups of services provided by ECS can be accessed directly through the ECS interfaces.

The emphasis of this work was to develop a framework that can be used to create different domain specific DCSS and these systems can be realized either as browser based portals, standalone desktop applications, third-party systems or a combination of different clients based on the type of user. The extensible architecture employed allows one to support different types of front-end clients in addition to supporting both grid-based and non-grid based back-end resources. The unique capabilities provided by the ECS framework used to validate the hypothesis of this dissertation were as follows:

- Using a role based user profile the ECS framework is able to support the comprehensive set of services and interfaces to support the diverse requirements of different users of modern computational systems.
- The ECS framework provides a unique approach for automatic creation of proxy objects using metadata based application repository. These proxy objects support creation of complex multistep tasks out of standalone interdependent applications distributed across multiple organizations without modifying existing applications. Thus, ECS framework supports creation of distributed applications from standalone applications and automates several tasks that would otherwise require human intervention, thereby improving user productivity.

- In addition to providing basic services to access distributed remote resources ECS provides high-level user services and persistent services. Persistent services are unique to this framework and are used to save, search, retrieve, and share applications, configuration information, and final products generated. Persistent services are also used to provide historic information and create pedigrees. The high-level user services enable the development of advanced problem solving environments to support advanced users as well as support the transition of applications from a development mode to an operational mode. The support for application transition from research and development to operational use is an important contribution.
- The ECS framework is based on the commodity component technologies supported by the Java 2 Enterprise Edition (J2EE). The J2EE platform is widely supported and several application servers are available, thus the ECS framework can be deployed on any J2EE complaint application server by only modifying the deployment settings. These application servers can be distributed across multiple servers to provide high availability, scalability, and fault tolerance. Therefore the services provided by ECS can be distributed and individual services provided by ECS can be delivered through Portlets [97].
- Web Services (WSDL interfaces) provide a language neutral and platform independent method for accessing services provided by ECS from other third-party applications. ECS clients can be written in any programming language or development environment (Java, Visual Basic, C++, C#). Similarly any third-party service can also be integrated into the ECS framework. Thus, the ECS framework supports interoperability between different computational portals or PSEs.

Two DCSS, Distributed Marine Environment Forecast System (DMEFS) and Distributed Simulation System for Seismic Performance of Urban Regions (SPUR),

developed using the ECS framework described in Chapter V illustrated how services provided by the ECS framework were used to satisfy the requirements of modern multidisciplinary computational simulations. The DCSS developed using the ECS framework supports integration of users by providing customized interfaces and services based on the users role. Using the application metadata distributed standalone applications were connected together to work together as a single task. The application metadata was also used to automate routine tasks of staging and unstaging data, generation of batch scripts, and job submission. The user workspace supports reuse and sharing of configured tasks along with the associated data and configuration information. Other persistence services allow users to access historic information, provide annotation for results produced, and provide the capability to repeat computational simulations. An advanced user can perform development, testing, and validation and then share the application with other users. Novice users can import preconfigured and validated applications and use them without requiring extensive training to use and configure the application. Applications can be configured and submitted to multiple distributed resources from one place. Similarly applications executing on multiple platforms and status of distributed machines are monitored using the DCSS front-ends.

All of this new functionality provided by a DCSS reduces the time required to perform routine operations by automating some of these tasks (human intervention is reduced), supports reuse and sharing of application configuration and data (learning time is reduced), and provides a common place for obtaining all information required to perform computational simulations. Any reduction in time required to complete any of the routine operations such application configuration, job submission, job monitoring, task creation, and task sharing results in higher user productivity.

The new functionality provided by DCSS does not compromise the functionality, performance, or security of existing applications. Since existing applications are not modified, there is no loss in current functionality. There is also no impact on the

performance since the applications are submitted for execution "as is" without any modification. In order to access the back-end resources a DCSS relies on the existing security mechanism the user is currently using. Therefore the security mechanism used to access back-end resources with and without a DCSS is the same. A DCSS based on the ECS framework provides several additional layers of security in terms of accessing the middle-tier services which can be used to selectively enable or disable certain features based on users role. Since only proxy objects are maintained in the middle-tier and users still requires valid credentials to access the back-end resources, even if the middle-tier is compromised actual applications and data that reside on the back-end are not compromised.

The ECS framework presented in this dissertation was used to prototype two DCSS that integrates disparate users, resources, and applications without compromising the functionality, performance, or security of existing applications. The DCSS, DMEFS and SPUR, provide additional functionality that could potentially reduce time spent by users to learn and perform routine computational simulations and the reduction in time in-turn could results in better user productivity. The DCSS prototyped show that it is feasible to develop DCSE described in Chapter I using the ECS framework without compromising the functionality, performance, or security of existing applications, thereby validating the hypothesis.

CHAPTER VII

FUTURE WORK

The focus of this dissertation was to develop an open framework which supports interaction between diverse set of users and distributed computational resources and multidisciplinary applications. The Enterprise Computational Services (ECS) framework provides a core set of services which can be exploited to support different users requirements. These services could be used to perform computational simulations, access data, query information, or extract knowledge through a common infrastructure as shown in figure 1.5. In this dissertation services required to support computational simulations (services shown inside the dotted-box in figure 1.5) were prototyped. Services required to support efficient access to datasets through the use of metadata repositories, creation of information repositories, operations to support conversion of data to information, creation of knowledge bases, and operations to extract knowledge from information were not prototyped. Similar to the creation of application metadata repositories, metadata repositories for datasets could also be created and linked with the application metadata. Thus, whenever a new simulation is performed the dataset metadata repository could be used to obtain input data required for the simulation as well as update the dataset metadata repository with the results of the simulation. Similarly, the dataset metadata repository could be linked with the application metadata repository and when data are not available in the form or format requested by the user, appropriate applications that can generate the required data are executed. These applications could be simple format convertors or complete simulations depending on the user request and the data currently available. This would allow users to browse for specific data products and

when not available the services provided by ECS could transparently execute appropriate applications and generate the required data on-the-fly. Linking of application metadata and dataset metadata could provide a mechanism for automatic creation of metadata for datasets whenever an application is executed.

The present realization of ECS exploits the Commodity Grid Kits (CoG) [16] to access back-end resources for authentication and authorization, job submission and monitoring, and file transfer. The emerging Open Grid Services Architecture (OGSA) [48] seeks to provide an interface based on web services for managing grid service instances. The OGSA specification provides the interfaces and behaviors of a *grid service* and the Open Grid Services Infrastructure (OGSI) [49] provides the corresponding implementation for these interfaces. Services provided by ECS are also published as web services, but these interfaces were prototyped prior to the OGSI specification and do not comply with the OGSI specification. ECS web service interfaces could be updated to be complaint with the OGSI specification so that the ECS framework could exploit OSGI.

REFERENCES

[1] R. Boisvert and J. Rice, "From scientific software libraries to problem solving environments," *IEEE Computer Science and Engineering*, no. 3, pp. 44–53, 1996.

[2] M. Henning and S. Vinoski, *Advanced CORBA Programming with C++*. Addison-Wesley Pub Co, 1999. ISBN: 0201379279.

[3] R. Grimes, *Professional Dcom Programming*. Wrox Press Inc, 1997. ISBN: 186100060X.

[4] R. Monson-Haefel, *Enterprise JavaBeans*. O'Reilly and Associates, 3rd ed., 2001. ISBN: 0596002262.

[5] N. Kassem and E. Team, *Designing Enterprise Applications with the Java 2 Platform, Enterprise Edition*. Addison–Wesley, June 2000. ISBN 0-201-70277-0.

[6] J. Richter, *Applied Microsoft .NET Framework Programming*. Microsoft Press, 2002. ISBN: 0735614229.

[7] E. Cerami, *Web Services Essentials*. O'Reilly and Associates, 2002. ISBN: 0596002246.

[8] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid: Enabling scalable virtual organizations," *International Journal Supercomputer Applications*, vol. 3, no. 15, 2001.

[9] A. Chervenak, I. Foster, C. Kesselman, and C. S. S. Tuecke, "The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets," *Journal of Network and Computer Applications*, no. 23, pp. 187–200, 2001.

[10] "MATLAB." http://www.mathworks.com/. Last accessed: Mar 16, 2003.

[11] "Khoros." http://www.khoral.com/. Last accessed: Mar 16, 2003.

[12] "Advanced Visual Systems (AVS)." http://www.avs.com/. Last accessed: Mar 16, 2003.

[13] E. N. Houstis, J. R. Rice, S. Weerawarana, A. C. Catlin, P. Papachiou, K.-Y. Wang, and M. Gaitatzes, "Parallel (//) ellpack: A problem solving environment for pde based applications on multicomputer platforms."

http://www.cs.purdue.edu/research/cse/pellpack/paper/pellpack-paper-1.html. Last accessed: Mar 16, 2003.

[14] G. C. Fox, D. Gannon, and M. Thomas, "A summary of grid computing environments," *Concurrency and Computation: Practice and Experience*, vol. 14, pp. 1035–1044, November-December 2002. Special Issue on Grid Computing Environments.

[15] I. Foster and C. Kesselman, eds., *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, Inc., 1999. ISBN 1-55860-475-8.

[16] G. von Laszewski, I. Foster, J. Gawor, W. Smith, and S. Tuecke, "Cog kits: A bridge between commodity distributed computing and high-performance grids," in *ACM Java Grande 2000 Conference*, pp. 97–106, June 2000.

[17] T. Haupt, P. Bangalore, and G. Henley, "A computational web portal for distributed marine environment forecast system," in *9th International Conference on High-Performance Computing and Networking, HPCN Europe 2001*, (Amsterdam), pp. 104–113, Springer-Verlag Berlin Heidleberg New York, June 2001. ISSN 0302-9743, ISBN 3-540-42293.

[18] "Distributed simulation framework for seismic performance for urban regions." http://www.erc.msstate.edu/~jmeyer/SPUR/. Last accessed: Mar 16, 2003.

[19] T. Haupt, P. Bangalore, and G. Henley, "Mississippi computational web portal," *Concurrency and Computation: Practice and Experience*, vol. 14, pp. 1275–1287, November-December 2002. Special Issue on Grid Computing Environments.

[20] P. V. Bangalore, T. Haupt, S. K. Ghafoor, A. K. Kalyanasundaram, M. Khoutornenko, and B. Raman, "An Open Architecture for Developing Grid Computing Environments," 2003. Technical Report.

[21] "Distributed Marine Environment Forecast System." http://www.erc.msstate.edu/geotech/DMEFS/index.html. Last accessed on: March 16, 2003.

[22] A. Gordon, *The COM and COM+ Programming Primer*. Prentice Hall, 2000. ISBN: 0130850322.

[23] W. Grosso, *Java RMI*. O'Reilly and Associates, 2001. ISBN: 1565924525.

[24] R. Englander, *Developing Java Beans*. Java Series, O'Reilly and Associates, 1997. ISBN: 1565922891.

[25] "MSR Millennium Project." http://research.microsoft.com/sn/Millennium/. Last accessed: March 16, 2003.

[26] "High Level Architecture (HLA)." https://www.dmso.mil/public/transition/hla/. Last accessed on: March 15, 2003.

[27] "Runtime Infrastructure (RTI)." https://www.dmso.mil/public/transition/hla/rti/. Last accessed on: March 15, 2003.

[28] B. A. Allan, R. C. Armstrong, A. P. Wolfe, J. Ray, D. E. Bernholdt, and J. A. Kohl, "The cca core specification in a distributed memory spmd framework," *Concurrency and Computation: Practice and Experience, Special Issue: Software Architectures for Scientific Applications*, vol. 14, pp. 323–346, April 2002.

[29] G. Follen, C. Kim, I. Lopez, J. Sang, and S. Townsend, "A corba-based distributed component environment for wrapping and coupling legacy scientific codes," *Cluster Computing*, vol. 5, pp. 277–286, July 2002.

[30] A. S. Grimshaw and W. A. Wulf, "The legion vision of a worldwide virtual computer," *Concurrency and Computation: Practice and Experience*, vol. 14, November-December 2002. Special Issue on Grid Computing Environments.

[31] I. Foster and C. Kesselman, "Globus: A metacomputing infrastructure toolkit," *International Journal of Supercomputer Applications*, vol. 11, no. 2, pp. 115–128, 1997.

[32] D. W. Erwin, "Unicore – a grid computing environment," *Concurrency and Computation: Practice and Experience*, vol. 14, no. 13–15, pp. 1395–1410, 2002.

[33] "Global Grid Forum." http://www.ggf.org. Last accessed: March 16, 2003.

[34] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke, "A security architecture for computational grids," in *5th ACM Conference on Computer and Communications Security Conference*, pp. 83–92, 1998.

[35] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, "Grid information services for distributed resource sharing," in *Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10)*, IEEE Press, 2001.

[36] I. Foster and G. von Laszewski, "Usage of ldap in globus." ftp://ftp.globus.org/pub/globus/papers/ldap_in_globus.pdf. Last accessed on: March 16, 2003.

[37] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, "A resource management architecture for metacomputing systems," in *IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing*, pp. 62–82, 1998.

[38] "Portable Batch System (PBS)." http://www.nas.nasa.gov/Groups/SciCon/Origins/Cluster/PBS/index.html. Last accessed on: March 16, 2003.

[39] J. Basney and M. Livny, *High Performance Cluster Computing*, vol. 1, ch. Deploying a High Throughput Computing Cluster. Prentice Hall, 1999.

[40] J. Bester, I. Foster, C. Kesselman, J. Tedesco, and S. Tuecke, "Gass: A data movement and access service for wide area computing systems," in *Sixth Workshop on I/O in Parallel and Distributed Systems*, 1999.

[41] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, S. Meder, and S. Tuecke, "Gridftp protocol specification." Global Grid Forum (GGF) GridFTP Working Group Document, September 2002. http://www.globus.org/research/papers/GridftpSpec02.doc.

[42] G. von Laszewski, I. Foster, J. Gawor, and P. Lane, "A java commodity grid kit," *Concurrency and Computation: Practice and Experience*, vol. 13, no. 8–9, pp. 643–662, 2001. Special Issue on Grid Computing Environments.

[43] "Python globus (pyglobus)." http://www-itg.lbl.gov/gtg/projects/pyGlobus/. Last accessed on: March 15, 2003.

[44] M. Parashar, G. von Laszewski, S. Verma, J. Gawor, K. Keahey, and N. Rehn, "A corba commodity grid kit," *Concurrency and Computation: Practice and Experience*, vol. 14, pp. 1057–1074, November-December 2002. Special Issue on Grid Computing Environments.

[45] T. Ylonen, "Ssh – secure login connections over the internet," in *The Sixth USENIX Security Symposium*, pp. 37–42, July 1996. http://www.usenix.org/publications/library/proceedings/sec96/full_papers/ylonen/.

[46] J. Novotny, S. Tuecke, and V. Welch, "An online credential repository for the grid: Myproxy," in *Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10)*, IEEE Press, 2001.

[47] A. S. Grimshaw, "Easy to use object-oriented parallel programming with mentat," *IEEE Computer*, pp. 39–51, May 1993.

[48] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "The physiology of the grid: An open grid services architecture for distributed systems integration." Open Grid Service Infrastructure Working Group, Global Grid Forum, June 2002. http://www.globus.org/research/papers/ogsa.pdf.

[49] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, D. Snelling, and P. Vanderbilt, "Open grid services infrastructure (ogsi)." Open Grid Service Infrastructure Working Group, Global Grid Forum, February 2003. http://www.gridforum.org/ogsi-wg/drafts/draft-ggf-ogsi-gridservice-23_2003-02-17.pdf.

[50] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, "Web services description language (wsdl) 1.1." http://www.w3.org/TR/wsdl. Last accessed on: March 15, 2003.

[51] D. Arnold, S. Agrawal, S. Blackford, J. Dongarra, M. Miller, K. Seymour, K. Sagi, Z. Shi, and S. Vadhiyar, "Users' Guide to NetSolve V1.4.1," Innovative Computing

Dept. Technical Report ICL-UT-02-05, University of Tennessee, Knoxville, TN, June 2002.

[52] T. Suzumura, H. Nakada, M. Saito, S. Matsuoka, Y. Tanaka, and S. Sekiguchi, "The ninf portal: An automatic generation tool for the grid portals," in *Java Grande*, pp. 1–7, November 2002. http://ninf.apgrid.org/papers/javagrande02suzumura/javagrande02suzumura.pdf.

[53] C. Baru, R. Moore, A. Rajasekar, and M. Wan, "The sdsc storage resource broker," 1998.

[54] M. D. Beynon, T. Kurc, U. Catalyurek, C. Chang, A. Sussman, and J. Saltz, "Distributed processing of very large datasets with datacutter," *Parallel Computing*, vol. 27, no. 11, pp. 1457–1478, 2001.

[55] E. Thomsen, *Olap Solutions: Building Multidimensional Information Systems.* John Wiley and Sons, 2002. ISBN: 0471400300.

[56] "HTTP – Hypertext Transfer Protocol." http://www.w3.org/Protocols/. Last accessed on: March 15, 2003.

[57] J. Hunter, *Java Servlet Programming.* O'Reilly and Associates, 2nd ed., 2001. ISBN: 0596000405.

[58] S. Brown, R. Burdick, J. Falkner, B. Galbraith, R. Johnson, L. Kim, C. Kochmer, T. Kristmundsson, and S. Li, *Professional JSP.* Wrox Press Inc, 2001.

[59] M. Bellinaso and K. Hoffman, *ASP.NET Website Programming: Problem - Design - Solution.* Wrox Press Inc, 2002.

[60] "HyperText Markup Language (HTML)." http://www.w3.org/MarkUp/. Last accessed on: March 15, 2003.

[61] S. White, M. Fisher, R. Cattell, G. Hamilton, and M. Hapner, *JDBC(TM) API Tutorial and Reference: Universal Data Access for the Java(TM) 2 Platform.* Addison-Wesley, 2nd ed., 1999. ISBN: 0201433281.

[62] "Extensible Markup Language (XML)." http://www.w3.org/XML/. Last accessed on: March 15, 2003.

[63] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer, "Simple object access protocol (soap) 1.1." W3C Note, May 2000. http://www.w3.org/TR/SOAP/.

[64] J. Newmarch, *A Programmer's Guide to Jini Technology.* Apress, 2000. ISBN 1-893115-80-1.

[65] E. Freeman, S. Hupfer, and K. Arnold, *JavaSpaces Principles, Patterns, and Practice.* Addison–Wesley, August 1999. ISBN 0-201-30955-6.

[66] R. Bjornson, N. Carriero, D. Gelernter, and J. Leichter, "Linda, the portable parallel," Tech. Rep. 520, Yale University Department of Computer Science, January 1988.

[67] A. Oram, ed., *Peer-to-Peer : Harnessing the Power of Disruptive Technologies*. O'Reilly and Associates, March 2001. ISBN: 059600110X.

[68] "Universal description, discovery and integration (uddi)." http://www.uddi.org/specification.html. Last accessed on: March 15, 2003.

[69] "WWW //ELLPACK: A Web Based Problem Solving Environment for Partial Differential Equations." http://www.webpdelab.org/. Last accessed: Mar 16, 2003.

[70] M. Lorch and D. Kafura, "Symphony – a java-based composition and manipulation framework for computational grids," in *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, May 2002. http://symphony.cs.vt.edu/publications/ccgrid2002-symphony.pdf.

[71] R. Rheinheimer, J. I. Beiriger, H. P. Bivens, and S. L. Humphreys, "The asci computational grid: Initial deployment," *Concurrency and Computation: Practice and Experience*, vol. 14, pp. 1351–1363, November-December 2002. Special Issue on Grid Computing Environments.

[72] V. Mann and M. Parashar, "Engineering interoperable computational collaboratories on the grid," *Concurrency and Computation: Practice and Experience, Special Issue: Grid Computing Environments*, vol. 14, pp. 1569–1593, November-December 2002.

[73] K. Schuchardt, B. Didier, and G. Black, "Ecce – a problem solving environment's evolution toward grid services and a web architecture," *Concurrency and Computation: Practice and Experience*, vol. 14, pp. 1221–1239, November-December 2002. Special Issue on Grid Computing Environments.

[74] T. Haupt, E. Akarsu, G. Fox, and C. Youn, "The gateway system: Uniform web-based access to remote resources," *Concurrency: Practice and Experience*, vol. 12, pp. 629–642, 2000.

[75] G. Aloisio and M. Cafaro, "Web-based access to the grid using the grid resource broker portal," *Concurrency and Computation: Practice and Experience*, vol. 14, pp. 1145–1160, November-December 2002. Special Issue on Grid Computing Environments.

[76] W. A. W. III, I. Bird, J. Chen, B. Hess, A. Kowalski, and Y. Chen, "A web services data analysis grid," *Concurrency and Computation: Practice and Experience*, vol. 14, pp. 1303–1311, November-December 2002. Special Issue on Grid Computing Environments.

[77] A. Natrajan, A. Nguyen-Tuong, M. A. Humphrey, M. Herrick, B. P. Clarke, and A. S. Grimshaw, "The legion grid portal," *Concurrency and Computation: Practice*

and Experience, vol. 14, pp. 1365–1394, November-December 2002. Special Issue on Grid Computing Environments.

[78] "The NPACI Hotpage Grid Computing Portal." https://hotpage.npaci.edu. Last accessed on: March 15, 2003.

[79] K. A. Iskra, R. G. Belleman, G. D. van Albada, J. Santoso, P. M. A. Sloot, H. E. Bal, H. J. W. Spoelder, and M. Bubak, "The polder computing environment, a system for interactive distributed simulation," *Concurrency and Computation: Practice and Experience*, vol. 14, pp. 1313–1335, November-December 2002. Special Issue on Grid Computing Environments.

[80] C. Johnson, S. Parker, D. Weinstein, and S. Heffernan, "Component-based, problem-solving environments for large-scale scientific computing," *Concurrency and Computation: Practice and Experience*, vol. 14, pp. 1337–1349, November-December 2002. Special Issue on Grid Computing Environments.

[81] A. Schreiber, "The integrated simulation environment tent," *Concurrency and Computation: Practice and Experience*, vol. 14, pp. 1553–1568, November-December 2002. Special Issue on Grid Computing Environments.

[82] S. Krishnan, R. Bramley, M. Govindaraju, R. Indurkar, A. Slominski, D. Gannon, J. Alameda, and D. Alkaire, "The xcat science portal," in *ACM/IEEE conference on Supercomputing (SC2001)*, November 2001. http://www.extreme.indiana.edu/xcat.

[83] G. C. Fox, D. Gannon, and M. Thomas, eds., *Concurrency and Computation: Practice and Experience, Special Issue: Grid Computing Environments*, vol. 14. Wiley, November-December 2002.

[84] T. Haupt, E. Akarsu, and G. Fox, "Webflow: A framework for web based metacomputing," *Future Generation Computer Systems*, vol. 16, pp. 445–451, 2000.

[85] M. Thomas, M. Dahan, K. Mueller, S. Mock, C. Mills, and R. Regno, "Application portals: Practice and experience," *Concurrency and Computation: Practice and Experience*, vol. 14, pp. 1427–1443, November-December 2002. Special Issue on Grid Computing Environments.

[86] J. Novotny, "The grid portal development kit," *Concurrency and Computation: Practice and Experience*, vol. 14, pp. 1129–1144, November-December 2002. Special Issue on Grid Computing Environments.

[87] G. von Laszewski, M. Russell, I. Foster, J. Shalf, G. Allen, G. Daues, J. Novotny, and E. Seidel, "Community software development with the astrophysics simulation collaboratory," *Concurrency and Computation: Practice and Experience, Special Issue: Grid Computing Environments*, vol. 14, pp. 1289–1301, November-December 2002.

[88] "The information power grid launchpad portal." https://portal.ipg.nasa.gov/launchpad/servlet/launchpad. Last accessed on: March 15, 2003.

[89] "Jini-based portal augmenting grids." http://matsu-www.is.titech.ac.jp/šuzumura/jipang/. Last accessed on: March 15, 2003.

[90] T. Haupt, "Distributed simulation systems." Unpublished, Draft: Version 0.6.

[91] W. Stallings, *Operating Systems: Internals and Design Principles*. Prentice Hall, 1997. ISBN 0-13-887407-7.

[92] "Hierarchical Data Format (HDF)." http://hdf.ncsa.uiuc.edu/. Last accessed on: March 15, 2003.

[93] "Network Common Data Form (NetCDF)." http://www.unidata.ucar.edu/packages/netcdf/. Last accessed on: March 15, 2003.

[94] D. Flanagan, *JavaScript: The Definitive Guide*. O'Reilly and Associates, 4th ed., December 2001. ISBN: 0596000480.

[95] P. Niemeyer and J. Knudsen, *Learning Java*. O'Reilly and Associates, 2nd ed., July 2002. ISBN: 0596002858.

[96] "Java web start." http://java.sun.com/products/javawebstart/. Last accessed on: March 16, 2003.

[97] C. Vieregger, "Develop java portlets." http://www.javaworld.com/javaworld/jw-02-2003/jw-0207-iviews.html, February 2003. Last accessed on: March 16, 2003.

[98] "Extensible Stylesheet Language (XSL)." http://www.w3.org/Style/XSL/. Last accessed on: March 15, 2003.

[99] P. J. Martin, "Description of the navy coastal ocean model, version 1,0," Tech. Rep. NRL Formal Report 7322-00-9962, Naval Research Laboratory, Stennis Space Center, MS, 2000.

[100] R. M. Hodur, "Development and testing of the coupled ocean/atmosphere mesoscale prediction system (coamps)," Tech. Rep. NRL Memorandum Report 7533-93-7213, Naval Research Laboratory, Monterey, CA, 1993.

[101] H. Gunther, S. Hasselmann, and P. Janssen, "The wam model – cycle 4," tech. rep., Deutsches KlimaRechenZentrum, Hamburg, Germany, 1992.

[102] "George E. Brown, Jr. Network for Earthquake Engineering Simulation (NEES) Program." http://www.nees.org/. Last accessed on: March 16, 2003.

[103] H. Bao, J. Bielak, O. Ghattas, L. Kallivokas, D. OHallaron, J. Shewchuk, and J. Xu, "Large-scale simulation of elastic wave propagation in hetero geneous media on parallel computers," *Computational Methods in Applied Mechanical Engineering*, vol. 152, pp. 85–102, 1998.

[104] P. Chopra, J. Meyer, and A. Fernandez, "Immersive volume visualization of seismic simulations: A case study of techniques invented and lessons learned," in *Case Studies, IEEE Visualization*, (Boston, MA), Oct-Nov 2002.

[105] "Open system for earthquake engineering simulation." http://opensees.berkeley.edu/. Last accessed on: March 16, 2003.

APPENDIX A

APPLICATION METADATA

Application-specific metadata

```
<?xml version="1.0" encoding="UTF8" ?>
<AAD maxAppId="1">
   <application id="0">
   <signature>
     <name>Sample Example</name>
     <keywords />
     <version>1.0</version>
      <credit>
         <author>
           <name>Puri Bangalore</name>
           <institution>ERC</institution>
           <contact>puri@erc.msstate.edu</contact>
        </author>
        <institution />
     </credit>
  </signature>
   <description>
     This program uses two environment variables INPUTDIR and
     OUTPUTDIR and reads the file specified as the first command-line
     argument from the INPUTDIR and writes output file to the
     directory OUTPUTDIR using the second command-line argument as the
     output filename.
   </description>
  <documentation />
  <support />
   <reginfo>
     <registeredBy>puri</registeredBy>
     <created>2003.Mar.14, 03:47 PM</created>
     <lastModified>2003.Mar.14, 04:14 PM</lastModified>
  </reginfo>
   <arguments>
      <argument multiplicity="1" syntax="$v" id="modelarg2">
        <name>OutputFileName</name>
        <description>Name of the output file</description>
        <order>2</order>
        <type />
        <restrictions />
        <value />
     </argument>
      <argument multiplicity="1" syntax="$v" id="modelarg1">
        <name>InputFileName</name>
        <description>Name of the input file</description>
        <order>1</order>
```

```
      <type />
      <restrictions />
      <value />
   </argument>
</arguments>
 <parameterfiles>
   <file idref="infile1" />
</parameterfiles>
 <inputfiles>
    <file id="infile1">
      <name>InputFile</name>
      <metadata />
      <description>Name of Input File</description>
   </file>
</inputfiles>
 <outputfiles>
    <file id="outfile1">
      <name>OutputFile</name>
      <metadata />
      <description>Name of Output File</description>
   </file>
</outputfiles>
 <gui>
   <jsp />
   <class />
   <url />
</gui>
 <QOS>
    <cpu>
      <min />
      <max />
   </cpu>
    <memory>
      <min />
      <max />
   </memory>
   <adaptionrule />
    <environment>
       <variable id="modelenviron1">
         <name>INPUTDIR</name>
      </variable>
       <variable id="modelenviron2">
         <name>OUTPUTDIR</name>
      </variable>
   </environment>
```

```
  </QOS>
   <source>
     <host />
     <cvsroot />
     <path />
  </source>
  </application>
</AAD>
```

Machine-dependent metadata

```
<?xml version="1.0" encoding="UTF8" ?>
<target id="titan.erc.msstate.edu">
  <build />
   <run>
      <environment idref="modelenviron1">
        <value>/tmp</value>
     </environment>
      <environment idref="modelenviron2">
        <value>/tmp</value>
     </environment>
      <inputs>
         <file idref="infile1">
           <srcpath>/home/puri</srcpath>
           <srcmachine>titan.erc.msstate.edu</srcmachine>
           <srcname>$ARGUMENT:InputFileName</srcname>
           <destpath>$ENVIRONMENT:INPUTDIR</destpath>
           <destname>$ARGUMENT:InputFileName</destname>
        </file>
     </inputs>
      <outputs>
         <file idref="outfile1">
           <srcpath>$ENVIRONMENT:OUTPUTDIR</srcpath>
           <srcname>$ARGUMENT:OutputFileName</srcname>
           <destpath>/home/puri</destpath>
           <destname>$ARGUMENT:OutputFileName</destname>
           <destmachine>vulcan.ssc.erc.msstate.edu</destmachine>
        </file>
     </outputs>
     <executable>/ccs/mssl/ecs/dmefs/puri/work/example6</executable>
     <workdir>/tmp</workdir>
     <maxtime />
     <mintime />
     <maxwalltime />
```

```
      <maxcputime />
      <maxmemory />
      <minmemory />
      <queue>interactive</queue>
      <hostcount />
      <count>1</count>
      <grammyjob />
      <dryrun />
      <project />
      <stdin />
      <stdout />
      <stderr />
    </run>
  </target>
```

APPENDIX B

MACHINE METADATA

```
<?xml version="1.0" encoding="UTF8" ?>
<list>
  <machine id="titan.erc.msstate.edu">
    <description>Sun multiprocessor Server</description>
    <OS>Solaris 2.8</OS>
    <accessType>globus</accessType>
    <batchSystem>PBS</batchSystem>
    <location>Engineering Research Center, Starkville, MS</location>
    <connect>rsh</connect>
    <copy>globus</copy>
    <interactive>yes</interactive>
    <loginNode>titan.erc.msstate.edu</loginNode>
    <processors>8</processors>
     <batch>
       <type />
       <submit>globus</submit>
       <queuestatus />
       <queueserver />
        <queues>
          <queuename>default</queuename>
          <queuename>supermsparc@super1.erc.msstate.edu</queuename>
          <queuename>ultramsparc@ultra1.erc.msstate.edu</queuename>
       </queues>
    </batch>
     <compilers>
       <make>/bin/make</make>
       <gnumake>/usr/local/gnu/bin/gmake</gnumake>
       <f77>/opt/SUNWspro/bin/f77</f77>
       <f90>/opt/SUNWspro/bin/f90</f90>
       <CC>/opt/SUNWspro/bin/cc</CC>
       <CPP>/opt/SUNWspro/bin/CC</CPP>
    </compilers>
 </machine>
</list>
```

APPENDIX C

JAVA INTERFACES

The Java Bean interfaces for the services provided by Enterprise Computational Services are provided here.

ModelBean

```
package ecs.services;

import ecs.ejb.*;
import java.util.*;

public interface ModelInterface {

   public String newModel(String modelName, String group,
                          String modelInfo) throws Exception;
   public Model getModel(String modelId) throws Exception;
   public String getModelInfo(String modelId) throws Exception;
   public void setModelInfo(String modelId, String modelInfo)
          throws Exception;
   public void setModelName(String modelId, String modelName)
          throws Exception;
   public Collection getModelByGroup(String group) throws Exception;
   public Collection getAllModels() throws Exception;
   public String cloneModel(String newModelName, String newModelGroup,
                            String currentModelId) throws Exception;
   public void deleteModel(String modelId) throws Exception;
}
```

HostBean

```
package ecs.services;

import java.util.*;

public interface HostInterface {

    public String newHost(String modelId, String machineId,
                          String hostInfo)
           throws DataAccessException;
    public String getHostInfo(String hostId) throws DataAccessException;
    public void setHostInfo(String hostId, String hostInfo)
           throws DataAccessException;
    public Collection getHostByModelId(String modelId)
           throws DataAccessException;
    public Collection getHostByMachineId(String machineId)
```

```
            throws DataAccessException;
    public Collection getAllHosts()
            throws DataAccessException;
    public String cloneHost(String newModelId, String newMachineId,
                            String currentHostId)
            throws DataAccessException;
    public void deleteHost(String hostId) throws DataAccessException;
}
```

UserBean

```
package ecs.services;

import java.util.*;
import ecs.ejb.*;

public interface UserInterface {

    public boolean authenticateUser(String userId, String password)
            throws DataAccessException;
    public boolean authorizeUser(String userId, String role)
            throws Exception;
    public String newUser(String userId, String userName, String password,
                          String defaultRole, String otherRoles,
                          String email, String phone, String organization,
                          String unit, String status) throws Exception;
    public User getUser(String userId) throws Exception;
    public Collection getAllUsers();
    public Collection getUserByRole(String role);
    public Collection getUserByOrganization(String organization);
    public Collection getUserByUnit(String unit);
}
```

ProjectBean

```
package ecs.services;

import java.util.*;

public interface ProjectInterface {

   public String newProject(String userId, String projectName,
                            String projectDetails, String group)
          throws DataAccessException;
```

```
   public String cloneProject(String newProjectName, String newGroup,
                              String currentProjectId, String parentUserId)
          throws DataAccessException;
   public void deleteProject(String projectId) throws Exception;
   public ArrayList getAllProjectList() throws DataAccessException;
   public ArrayList getProjectListByUserId(String userId)
          throws DataAccessException;
   public ArrayList getProjectListByGroup(String group)
          throws DataAccessException;
   public Collection getAllProjects() throws DataAccessException;
   public Collection getProjectByUserId(String userId)
          throws DataAccessException;
   public Collection getProjectByGroup(String group)
          throws DataAccessException;
   public String getProjectName(String projectId)
          throws DataAccessException;
   public String getProjectGroup(String projectId)
          throws DataAccessException;
   public String getProjectDetails(String projectId)
          throws DataAccessException;
}
```

TaskBean

```
package ecs.services;

import ecs.ejb.*;
import java.util.*;
import org.w3c.dom.*;
import org.globus.security.*;

public interface TaskInterface {

    public String newTask(String projectId, String taskName,
                          String taskDetails, String taskInfo)
           throws DataAccessException;
    public Task getTask(String taskId) throws DataAccessException;
    public String getTaskInfo(String taskId) throws DataAccessException;
    public void setTaskInfo(String taskId, String taskInfo)
           throws DataAccessException;
    public Collection getAllTasks() throws DataAccessException;
    public Collection getTaskByProjectId(String projectId)
           throws DataAccessException;
    public String cloneTask(String newTaskName, String currentTaskId,
```

```
                            String parentProjectId)
            throws DataAccessException;
    public void deleteTask(String taskId) throws DataAccessException;
    public String submitTask(String taskId, GlobusProxy globusProxy,
                             String submitType)
            throws DataAccessException;
    public String getTaskName(String taskid) throws Exception;
    public String getTaskProjectId(String taskid) throws Exception;
}
```

ApplicationBean

```
package ecs.services;

import java.util.*;
import org.globus.security.GlobusProxy;

public interface ApplicationInterface {

   public String newApplication(String taskId, String applicationName,
                                String applicationDetails, String modelId,
                                String applicationInfo)
           throws Exception;
   public String cloneApplication(String newApplicationName,
                                  String currentApplicationId,
                                  String parentTaskId)
           throws Exception;
   public void deleteApplication(String applicationId)
           throws Exception;
   public String submitApplication(String projectId, String taskId,
                                   String taskName, String appId,
                                   String appName, String appInfo,
                                   String runId, GlobusProxy globusProxy,
                                   String submitType)
           throws Exception;
   public String getApplicationInfo(String appId) throws Exception;
   public String getApplicationName(String appId) throws Exception;
   public String getApplicationTaskId(String appId) throws Exception;
   public String getApplicationModelId(String appId) throws Exception;
   public String getApplicationDetails(String appId) throws Exception;
   public void setApplicationInfo(String appId, String appInfo)
           throws Exception;
   public void setApplicationName(String appId, String appName)
           throws Exception;
```

```
   public void setApplicationModelId(String appId, String modelId)
          throws Exception;
   public void setApplicationDetails(String appId, String appDetails)
          throws Exception;
   public ArrayList getApplicationListByTaskId(String appId)
          throws Exception;
   public ArrayList getApplicationIdsByTaskId(String taskId)
          throws DataAccessException;
}
```

JobBean

```
package ecs.services;

import java.util.*;
import ecs.ejb.*;

public interface JobInterface {

   public String newJob(String runId, String userId, String projectId,
                        String projectName, String taskId, String taskName,
                        String applicationId, String applicationName,
                        String machineId, String queueName, String queueId,
                        long startTime, long endTime, String status,
                        String jobInfo);
   public Job getJob(String jobId);
   public String getJobInfo(String jobId);
   public void setJobInfo(String jobId, String jobInfo);
   public String getStatus(String jobId);
   public void setStatus(String jobId, String status);
   public long getEndTime(String jobId);
   public void setEndTime(String jobId, long endtime);
   public long getStartTime(String jobId);
   public void setStartTime(String jobId, long starttime);
   public String getQueueId(String jobId);
   public void setQueueId(String jobId, String status);
   public Collection getJobByUserId(String userId);
   public Collection getJobByProjectId(String projectId);
   public Collection getJobByTaskId(String taskId);
   public Collection getJobByApplicationId(String applicationId);
   public Collection getJobByRunId(String runId);
   public Collection getJobByMachineId(String machineId);
   public Collection getJobByQueueName(String queueName);
   public Collection getJobByQueueId(String queueId);
```

```
   public Collection getJobByStartTime(long startTime);
   public Collection getJobByEndTime(long endTime);
   public Collection getJobByStatus(String status);
   public Collection getAllJobs();
   public Collection getJobByUserIdMachineId(String userId,
                                             String machineId);
   public void deleteJob(String jobId);
}
```

ScheduleBean

```
package ecs.services;

import java.util.*;

public interface SchedulerInterface {

   public boolean isSchedulerActive();
   public void activateScheduler() throws Exception;
   public void deactivateScheduler() throws Exception;
   public void createTimer(String scheduleId, String taskId,
                           long firstTime, long period);
   public void cancelTimer(String scheduleId);
   public String newSchedule(String taskId, String taskName,
                             long firstTime, long period, String status)
          throws Exception;
   public ArrayList getSchedule(String scheduleId) throws Exception;
   public void setStatus(String scheduleId, String status)
          throws Exception;
   public ArrayList getScheduleByTaskId(String taskId);
   public ArrayList getScheduleByStatus(String status);
   public ArrayList getAllSchedules();
   public void deleteSchedule(String scheduleId) throws Exception;
}
```

SubmitBean

```
package ecs.services.globus;

import java.util.*;
import org.w3c.dom.*;
import org.globus.gram.*;
import org.globus.security.*;
import ecs.services.*;
```

```
import ecs.ejb.*;
import ecs.util.*;

public interface SubmitInterface {

   public String submitTask(String taskId, byte[] globusProxy)
          throws InvalidDOMException, DataAccessException,
                 SubmitException, JobException, GlobusProxyException;
   public String submitApplication(String appId, byte[] globusProxy)
          throws InvalidDOMException, DataAccessException,
                 SubmitException, JobException, GlobusProxyException;
   public String submit(Workspace ws, String domStr, byte[] globusProxy)
          throws InvalidDOMException, DataAccessException,
                 SubmitException, JobException, GlobusProxyException;
   public boolean checkStatus();
   public String submitRSL(String rslString, String hostname,
                           byte[] globusProxy)
          throws GlobusException, GlobusProxyException;
   public String submitCommand(String command, String directory,
                               String arguments, String stdout,
                               String stderr, String hostname,
                               byte[] globusProxy)
          throws GlobusException, GlobusProxyException;
   public String generateRSL(Document document, String jobId)
          throws InvalidDOMException;
}
```

APPENDIX D

IMPLEMENTATION DETAILS

Different software packages used to prototype the Enterprise Computational Services (ECS) framework and the Distributed Computational Simulation Systems are as follows:

- Java Development Kit 1.3 (java.sun.com/jdk1.3)
- ANT 1.4.1 (apache.org/ant)
- Orion application server and web server 1.52 (www.orionserver.com)
- WASP server 4.0 (www.systinet.com)
- Java CoG 0.9.13 (www.globus.org/cog/java)
- Xalan-Java version 1.2.D02 (http://xml.apache.org/xalan-j/)
- Xerces Java parser version 1.1.2 (http://xml.apache.org/xerces-j)