Ecce – A Problem Solving Environment's Evolution Toward Grid Services and a Web Architecture

Karen Schuchardt, Brett Didier, Gary Black
Pacific Northwest National Laboratory

Karen.Schuchardt@pnl.gov, Brett.Didier@pnl.gov, Gary.Black@pnl.gov

***Abstract**: The Extensible Computational Chemistry Environment (Ecce), an innovative problem solving environment (PSE), was designed a decade ago, before the emergence of the Web and Grid computing services. In this paper, we briefly examine the original Ecce architecture and discuss how it is evolving to incorporate both Grid services and components of the Web to increase its range of services, reduce deployment and maintenance costs, and reach a wider audience. We show that Ecce operates in both Grid and non-Grid environments, an important consideration given Ecce's broad range of uses and user community, and discuss the strategies for loosely coupled components that make this possible. Both in-progress work and conceptual plans for how Ecce will evolve are presented.*

1.0 Introduction

Problem-solving environments (PSEs) are problem-oriented computing environments that support the entire assortment of scientific computational problem-solving activities ranging from problem formulation to algorithm selection to simulation execution to solution visualization. PSEs support collaboration among people separated in space and time and provide access to a diverse set of resources that includes computational codes and compute resources. In other words, PSEs link a heterogeneous mix of resources including people, computers, data, and information within a seamless environment to solve a problem [1, 2]. Because of the broad functionality provided, PSEs are complex systems to build, maintain, and deploy. PSEs are domain specific, but their infrastructure requirements are common across domains. To reduce complexity and associated costs, PSEs use domain-independent services that can also be employed by other PSEs and computing applications. For a variety of reasons, including platform, language, and technology dependencies, the goal of producing reusable services has not been fully realized. In practice, overlapping services are typically developed and deployed separately for each PSE.

In recent years, there have been monumental changes in information technology, driven in large part by the materialization of Web computing for information delivery and business transactions. Accompanying these changes has been a movement toward open standards for protocols and application programming interfaces (APIs), a vigorous open source development community, sophisticated component frameworks, and low-cost commodity software aimed at supporting a distributed computing model. At the same time, the concept of Grid computing was being investigated and developed in research environments. Grid computing seeks to change the way in which scientists access distributed computing resources by creating infrastructure that enables automated services for resource scheduling and load balancing, data delivery, quality of service, authentication and delegation, and related issues [3]. As the Grid develops reliable and cost-effective solutions, researchers will expect these services to be available through their problem-solving environments. Together, these changes promise enormous potential benefits of increased throughput, improved collaboration, and elimination of time-consuming and mundane tasks that researchers often perform manually.

The movement toward standard protocols seen in both the Web and Grid environments is beginning to create reusable middleware and a service-oriented computing infrastructure for scientific computing applications. At the heart of this movement are Web servers and the HTTP protocol, and data interchange languages based on XML. Additionally, new developments in service and interface discovery promise to enable the interoperability of independently developed components with minimal agreement on technologies and infrastructure. The maturation of these and related technologies and the increased computer power and network performance to deal with the complexity/inefficiency of loosely coupled layers make new PSE designs possible, leading to an increasing interest in PSE development [4, 5, 6, 7, 8]. Computational portals such as Cactus [9] and Gateway [4] are one such design. Portal approaches restrict the user interface to the capabilities of a Web browser, which currently involves trading richness in the user interface for ease of deployment and reduced training requirements for users. We believe that the aforementioned developments will render the functional distinction between computational portals and PSEs obsolete in the coming years. When combined, these changes hold promise for a more-powerful, scalable, and flexible generation of problem-solving environments that can span scientific disciplines, host feature detection and other agents, and provide access from a wide range of devices and computing environments.

Against this backdrop, we have been investigating and prototyping higher-level middleware services that incorporate Web- and Grid-based technologies with the goal of creating generic, lightweight services that can be incorporated into problem-solving environments, portals, or any distributed computing application. Our efforts have focused primarily on data and metadata management, information services and interfaces, and Grid and non-Grid job submission. In this paper we describe an existing PSE – the Extensible Computational Chemistry Environment (Ecce) – and how these research investigations are being incorporated to expand services, reduce costs, and reach a wider audience. We also show how Ecce is being used as a testbed for Grid services.

2.0 Ecce Overview

Ecce is one component of the Molecular Science Software Suite (MS3) [10] developed at the Pacific Northwest National Laboratory (PNNL). MS3 is an integrated suite of comprehensive software that enables scientists to understand complex chemical systems by coupling advanced computational chemistry techniques with high-performance, parallel computing systems. As shown in Figure 1, MS3 consists of three components: NWChem provides advanced computational chemistry techniques, ParSoft provides efficient and portable libraries and tools that enable NWChem to run on a wide variety of parallel computing systems, and Ecce provides a suite of tools integrated within a problem-solving environment. Ecce's tools assist the user with many tasks, including the management of projects and calculations, construction of complex molecules and basis sets, selection of input options, distributed execution of computational models, real-time monitoring, and post-run analysis [11, 12]. Ecce and MS3 have been operational since 1997, and MS3 won an R&D 100 Award from *R&D Magazine* in 1999.

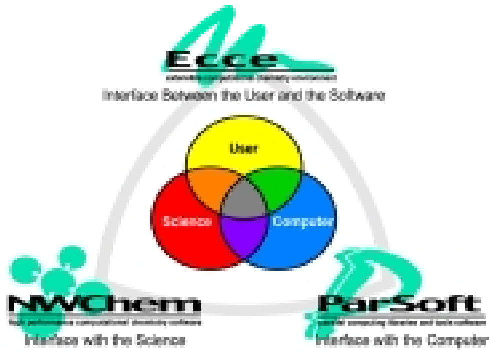
Ecce
Interface Between the User and the Software
User
Science
Computer
NWChem
Interface with the Science
ParSoft
Interface with the Computer

Figure 1. Molecular Science Software Suite (MS3): Ecce, NWChem, and ParSoft

As shown in Figure 2, the Ecce architecture addresses component-based application development, distributed code execution, and data and metadata management. Also addressed, but not shown in the figure, is a mechanism for integrating chemistry applications into the Ecce framework.

Ecce is composed of a suite of applications or components, each designed to assist the user with a single aspect of the research process. Thus, there are separate tools for constructing molecules, assigning basis sets, selecting input options, browsing resource availability, launching jobs, visualizing results, and creating and managing projects and calculations. This architecture is more complex than a single monolithic application but provides major benefits, including the capability to deploy tools independently while simplifying the development process. For example, when considering educational environments, the potential user base for a tool such as the molecule builder is much greater than the potential user base for the entire Ecce PSE.

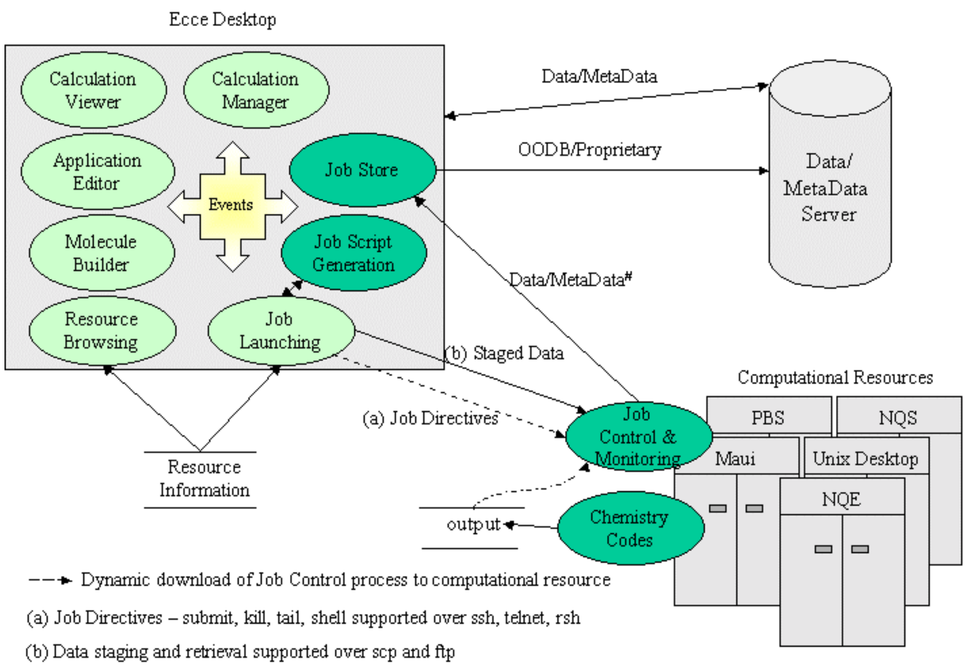

Figure 2. Operational view of original architecture

Users expect a PSE to provide a tightly integrated environment. To provide this feature while maintaining an application component architecture, Ecce uses a publish/subscribe event system to coordinate activities between applications. This approach is most evident when performing the calculation setup process: building a structure, assigning basis sets, and selecting input options must be done in a coordinated manner to maintain a consistent calculation definition and ensure the creation of valid and correct input files. Events are dispatched whenever data is changed, thus allowing tools to maintain a consistent state and user view. A further benefit of an event-based component architecture is that it provides a strong basis for collaborative work environments. A few additions to the architecture would enable users to cooperatively work on research problems.

An important aspect of any scientific PSE is its support for distributed execution of computational models. As shown in Figure 2, Ecce jobs can be launched to UNIX workstations or any cluster or supercomputer running batch systems such as Maui, NQE, NQS, or PBS. Files are staged to and from the compute server and all communications can be performed securely. Data is sent back to the client for real-time monitoring and visualization. Although Ecce is part of MS3, it is designed with an extension mechanism to support other chemistry codes. The commercial application Gaussian98 [13] is supported with plans to add support for GAMESS-US [14]. A data management component for persistently tracking the data and metadata associated with the chemistry research process underlies the Ecce framework. Until recently, this capability was accomplished through the use of an Object-Oriented Database Management System (OODBMS) and a common object model.

Although, Ecce was originally designed for use within PNNL's Environmental Molecular Sciences Laboratory, it is now deployed at multiple sites around the world. The potential user base includes the 400+ users/sites that have downloaded NWChem, researchers using other chemistry codes integrated within Ecce, and university students studying chemistry. These users have access to a wide variety of computing environments from major government-run computing centers with large numbers of resources, to individual researchers running primarily on their desktops, to commercial companies running entirely behind a firewall. Deployment to this range of computing environments is a major challenge for Ecce. With the upgrades to Ecce's architecture discussed in this paper, we expect the number of deployments to increase dramatically in the coming year, including deployments of individual components, such as the molecule builder, as well as the fully integrated PSE. A version of Ecce with the first steps toward a more flexible, scalable architecture was released in July 2001.

3.0 Evolving the Architecture

Existing PSEs, such as Ecce, provide extensive capabilities that are not easily recreated. The breadth of domain-specific functionality is achieved through the encoding of a significant amount of domain knowledge in the system architecture, which tends to promote a strongly coupled design. In Ecce's case, this functionality was achieved through the definition of a common object model and schema spanning not only chemistry domain objects but distributed computing and data management concepts as well. Unfortunately, a tightly coupled design tends to increase the costs of maintenance and deployment, impede the evolution and extension of the system, and limit component reuse. We are now reducing this coupling in Ecce by moving away from a PSE-wide schema, and instead are relying on open, dynamic schemas and translation services utilizing XML [15]. The adoption of a services-oriented architecture based on standard protocols will further reduce coupling, increase deployment options, and reduce costs.

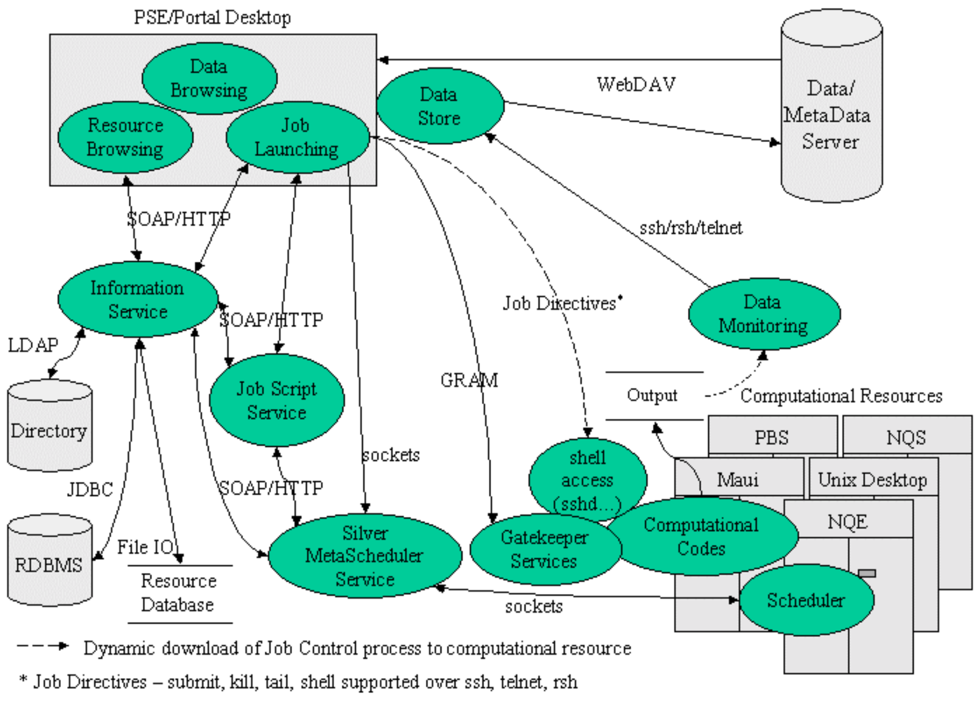

Figure 3. Operational view of planned architecture

Our efforts thus far to develop new PSE services have focused primarily on data and metadata management, information services and interfaces, and Grid and non-Grid job submission. Figure 3 depicts a test-bed Ecce architecture that interfaces to these middleware services. The data management architecture is designed around open, dynamic schemas and is implemented using the Web Distributed Authoring and Versioning (WebDAV or DAV) [16] protocol, an extension of HTTP. Resource information is accessed through an Information Service (IS) that provides a service-independent interface based on the Simple Object Access Protocol (SOAP) [17]. As shown in the diagram, the IS can transparently interface to various underlying implementations (e.g., LDAP, JDBC, flat files), thus providing deployment alternatives that can be tailored for varying operational requirements. Other services, such as a Job Script Service, access the IS to collect information required for their work.

Distributed execution services have been expanded to include Globus services (i.e., GateKeeper) and the Silver MetaScheduler [18]. We have also improved the integration of applications into the PSE framework by incorporating the use of scripting languages for describing the input requirements and dependencies of the computational modeling software. The use of scripting languages for performing this loose integration between the application and PSE allows application experts to define their interfaces. We now discuss our efforts in developing these services, the problems they address and benefits they provide, how they have been or are being applied by Ecce, and plans for future investigations.

3.1 Data and Metadata Management

Our vision for next-generation PSEs has evolved significantly over the last decade. We now look to PSEs to span scientific disciplines, incorporate collaboration capabilities, easily interface with commercial applications, host feature detection and other agents, and provide access from a wide range of devices. Realizing these goals requires a more flexible data management architecture than previously achieved. A key observation is that PSE components, although they manipulate common data artifacts, often interact through data flow, generating additional attributes or creating new objects related to data generated by another component. This observation leads to four design criteria as discussed in [15]:

Direct access to raw data. Access to data through a common object model, although useful in maintaining consistency, limits the representational power of applications added to the system. Providing direct access to the underlying persistent attributes of the data removes this constraint.

Self-describing data and data relationships. Without an object model common across all applications, another mechanism is needed to allow the discovery of data semantics. Using a self-describing data format (i.e., a format that provides metadata about the data), applications can use existing data in new ways and generate new data attributes and relations, as needed. Significantly, applications also can ignore existing relationships that have no meaning for them, or can translate the relationship semantics into their own domain ontology.

Schema-independent data stores. With self-describing data, the data storage system does not need to have deep knowledge of the application objects. By removing knowledge of the schema from the storage system, it becomes possible to support multiple independent or loosely coupled schemas within a single data store where these schemas can evolve without changes to the data store itself.

Separation of application-level objects from the data storage mechanism via standard protocol(s). Using a standard protocol for describing data management operations helps to maintain schema independence. Additionally, a standard protocol allows the implementation of the data store to be independent of the application technologies. Thus, the data store can be selected based on the performance, cost, and scaling requirements for a given PSE deployment and on the expected use patterns. Similarly, specifying a protocol instead of a programming interface enables client-side components to be independent of language and platform.

These four criteria lead to a very flexible, yet powerful architecture. Applications designed this way can be developed independently, yet integrated deeply based on a partial, post-development mapping between their respective schema descriptions. Data becomes discoverable by applications not considered during the original design process. Different schemas and different versions of schemas can co-exist without the need for synchronized deployments. Finally, scalable systems that transparently leverage the Web and Grid infrastructure can be deployed.

Ecce's original data architecture, built upon an OODBMS, common schema, and object-level component integration, provided a powerful approach to managing complex computational chemistry research data, codifying current practices, and allowing a high degree of integration between PSE components. In practice, the tight coupling resulting from the use of an OODB and common schema has had significant deployment and maintenance costs. As described in [12], OODBM systems have failed to mature and standardize as rapidly as expected. Significant problems include proprietary binary formats, tight coupling between the programming language and the OODB, lack of application development tools, and a painful schema evolution process due to outdated schema/application compilation cycles.

To move Ecce toward this vision for next-generation PSEs, Ecce's data architecture has been migrated to a Web-enabled, protocol-based solution. We have adopted the WebDAV protocol as the centerpiece of the data and metadata management architecture. DAV, an extension to HTTP 1.1, was originally designed to support collaborative authoring [19]. It provides structured XML-encoded requests for manipulating MIME-typed "*documents*" (i.e., get, put, move, copy, lock) and associated metadata (i.e., propfind, proppatch). DAV "*documents*" are not restricted to text-oriented formats and are more analogous to files or binary large objects. Each piece of metadata is an XML-encoded key-value pair in which the value may be simple text or contain complex data in, for example, the form of an XML object. New metadata can be added at any time, and applications can manipulate arbitrary subsets of metadata. For example, an application can request only the metadata values it understands from the server. Thus, the DAV protocol, with its constructs to logically organize opaque, typed data and to document that data with arbitrary metadata, maps directly into our criteria.

DAV currently supports only a simple, unordered container/contains relationship, but the wide range of data relationships used in PSEs (e.g., temporal, derivative, historical, and sequence, as well as the "is-a" and "has-a" object modeling dependencies) can be encoded using DAV's XML metadata properties [20]. Extensions to DAV, such as DAV Searching and Locating (DASL) [21], Advanced Collections [22], and Versioning [23], which are currently under development, promise additional PSE-relevant capabilities. XML provides rich capabilities for schema description (XML Schema) and translation (XSLT), thus avoiding name collisions (XML Namespaces) and representing relationships (XLink). The emergence of scientific domain languages defined in XML and generic XML parsing tools provide additional leverage (e.g., the Chemical Markup Language (CML) [24], MathML [25], the Extensible Scientific Interchange Language (XSIL)[26]).

The Ecce data model consists of class hierarchies for modeling experiments and computations, molecules, basis sets, compute jobs, and output data properties. Previously, these objects were persistently managed and served to applications by an OODBMS [15]. Figure 4 provides a simplified view of the object model. Briefly, the model shows a study subject (Molecule) on which tasks of an Experiment are performed, the results of which are a series of n-dimensional output Properties. As a user constructs molecules, assigns basis sets and other input options, and runs the job, all the data and associated metadata are captured within objects and persistently stored. The metadata is available for browsing, querying, replication, verification, and sharing, but only to applications that understand the Ecce schema. Important raw input and output files are also stored.

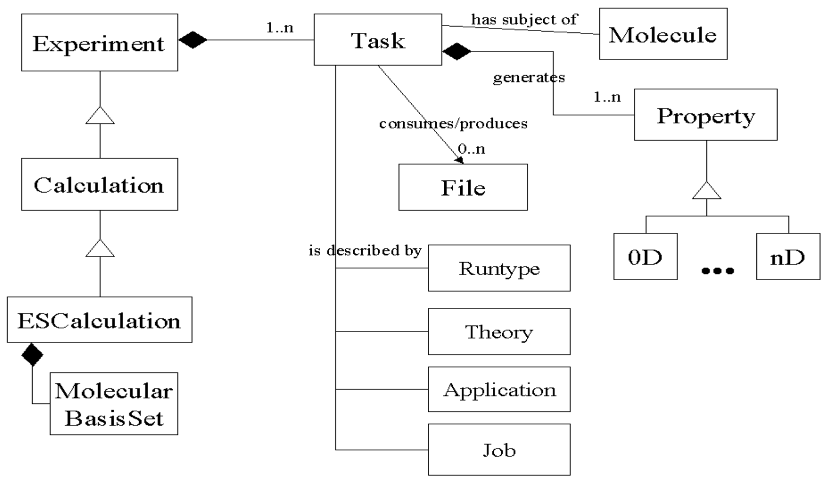

Figure 4. Simplified Ecce object model

In Ecce 2.0, the model shown in Figure 4 was mapped to the DAV constructs of collections, documents, and metadata. In general objects recognizable by domain scientists were mapped to separate DAV documents. This strategy allows access to raw data at the lowest level of granularity. Users and programs can access individual Ecce data objects via HTTP or DAV given their URLs and appropriate permissions on the server. We assigned information as metadata if it was descriptive of the document contents or could be useful for searching. In the future, we plan to describe relationships between objects in the metadata using Xlink, thus allowing dynamic interpretation of the schema independent of the physical layout.

Data services are crucial to both higher-level Grid services, such as those provided by PSEs or portals, and also to lower-level Grid services, such as scheduling and job launching. Ecce's conversion to DAV demonstrates the suitability of DAV for higher-level services. We believe that extensions to DAV, analogous to the GridFTP [27] extensions, can make DAV suitable for implementing lower-level services such as file staging which is required by schedulers and job launch services. Because HTTP derivatives offer a plug-in authentication mechanism, it should be practical to build a secure file staging capability based on DAV and the Grid Security Infrastructure (GSI) [28] and the Akenti authorization mechanism [29].

A complete description of the migration of the Ecce data architecture from an OODBMS to an open, Web-based data and metadata management system can be found in [11].

3.2 Information Services

An Information Service is a vital component to a distributed computing infrastructure, providing information on a wide range of community resources that includes people, computational machines, queues, software, and services. Information about community resources transcends application boundaries, requiring application-independent definitions and access. An IS can also be used to provide access to user and application-specific resource information, such as user preferences and application defaults. As shown in Figure 3, an IS can enable low-level services such as job script generation and metascheduling, and also user-level services such as browsing and selecting compute resources for task execution. Our investigation into information services has focused on defining a lightweight service layer that hides the details of the underlying implementation and provides simplified object-level operations that fulfill PSE requirements. This allows a PSE installation to be tailored to a particular site. It can be deployed for small sites (a few users and compute resources) that can be adequately served by a flat-file-based IS, or it can be deployed for an entire organization and utilize existing services (e.g., a standalone directory server or the MetaComputing Directory Service Version 2 (MDS-2) [30]). As a result, the same PSE code base can seamlessly support both Grid and non-Grid deployments.

Figure 3 illustrates our IS architecture. An HTTP-based service is used as a generic front-end to the IS. The front-end service can connect to a variety of underlying IS implementations (e.g., Directory Services, RDBMS, files) and communicates using the native protocols and interfaces of these implementations (e.g., LDAP, JDBC, File I/O). Communication to the IS front-end is performed using SOAP. SOAP is an XML-based messaging protocol that can be used to implement remote procedure calls and is supported over various transport protocols, including HTTP. The combination of HTTP and SOAP not only provides implementation independence from the underlying IS, but also platform and programming language independence. This is an important concern for deploying a general service in scientific computing environments where heterogeneous computing systems exist and preferred programming languages vary by scientific community.

The SOAP interface that we have defined for the IS is object based and supports operations to save, modify, delete and retrieve objects. Objects are transmitted to and returned from the IS as XML document fragments as defined by the Document Object Model. Since XML is used for transmitting objects, the interface provides data format independence from the application programming language and also from the underlying service implementation. Because objects are treated opaquely by the SOAP interface, new definitions and extensions to existing definitions can be created without requiring interface modifications. Schema definitions for our objects are created in XML using a schema definition language derived from the Directory Services Markup Language (DSML) [31] effort. We tailored the DSML schema definition language to make it less directory-specific and to maintain compatibility, through simple translation, with the Grid Object Specification Version 1.0 [32]. Defining our schema in XML provides access to a wide variety of programming language toolkits that simplify the translation of the schema into service-specific schema (e.g., Netscape Directory Server, OpenLDAP) and the automated generation of object-based software.

Our current implementation uses Apache's Tomcat servlet container with the Apache SOAP toolkit installed as the HTTP-based front-end. We developed a Java servlet that implements the save, modify, delete, and retrieve methods. These servlet methods use the Xerces parser to extract object attributes before they invoke method(s) implemented in a technology-dependent library to perform the desired operation against the underlying IS. Our back-end service is Netscape's Directory Server, and we developed an implementation of the technology-dependent library for LDAP-based services using Sun's Java Naming and Directory Interface (JNDI) [33]. We expect this implementation will also interface to other LDAP-based services, such as MDS-2.

Our work has made progress in providing a simple abstraction for object-based access to implementation-independent information services. To further abstract the client application from the IS back-end, we are also investigating both schema and query translation services. We expect that interoperability between different schema and schema versions will always be an issue in a widely distributed computing environment. To address this issue, we are investigating the incorporation of schema translation services into the IS architecture. Client requests will include identifiers for the schema and schema version. The IS will look up and invoke the required translation services automatically, potentially providing both syntactic and semantic translations. Because XML is the base language of the IS requests, many tools are available for performing these translations. Schema translations will improve the interoperability of widely distributed computing environments by allowing different systems to be upgraded independently.

Providing generic IS access also requires an abstraction from the query language supported by the IS back-end service. We are investigating two approaches to provide this query syntax independence. The first approach is to accept a single query syntax through the IS interface and translate the query expression to the appropriate syntax. We are investigating the XQuery [34] specification as a possible solution for this approach. The second approach is to accept any standard query syntax (e.g., LDAP, SQL, XQuery) and translate query expressions as necessary to interface with the underlying service. Because of the different querying capabilities of the various query syntaxes, there will not always be a direct mapping of operations from one query syntax to another. However, we expect that we can define a mapping that will satisfy typical PSE query requirements. Our efforts will build upon recent activity to translate a specified query into a query algebra [35], reformulate the algebraic statements into the query language of the targeted information source, and then issue the resulting query.

Currently, Ecce uses a flat-file-based IS, but an upcoming release will implement an interface to the SOAP/HTTP front-end service, providing installations that can be easily configured for different IS implementations.

3.3 Job Management

Scientific PSEs support automated submission and management of computational jobs. At a minimum, this consists of authentication, generation of a job script suitable for the target machine and application, staging files, and monitoring job status. We also believe that monitoring job progress via filtering of data output is an integral part of a PSE’s architecture. Through efforts such as Global Grid Forum (GGF) [36], higher-level services such as metascheduling are now being researched and prototyped. As low-level Grid services stabilize and higher-level services provide reliable improvement in job throughput and cost effectiveness, users will expect and demand that these services be available from their PSEs.

Scientists utilize the full range of computing platforms from single-processor desktop workstations through tera-flop clusters or supercomputers to run their computational models. PSEs can be deployed in any of these environments or a hybrid environment. Our experience has shown that a PSE should work with whatever computing infrastructure exists rather than specifying custom infrastructure requirements that create administrative barriers to deployment. As with an information service, job launching infrastructure should be independent of any application. There are multiple ways in which this can be accomplished. All UNIX-based computing platforms provide at a minimum, some standard protocol-based access such as telnet and rsh. Additionally, ssh is typically available to provide the equivalent access in a more secure environment. Effective job management components can be built on top of these protocols. A number of existing toolkits offer support for job submission, including Globus [37], Legion [38], and NetSolve [39]. More recently, there has been promising research on metascheduling [18]. As depicted in Figure 3, this environment leaves PSEs with the issue of working across the multiple service implementations that exist and will continue to exist in different organizations. We now describe how Ecce implements job management, how it has been used as a testbed for Globus job services and the Silver MetaScheduler, and how lessons learned can be applied to future Grid developments.

Ecce's primary job submission capabilities are built on top of the standard access protocols (telnet, rsh, ssh) and client-side processing that generates a job script from user launch request settings, stages files to the compute server using ftp, scp, or rcp, and authenticates to the server to submit the job. Any additional services required by Ecce, such as output monitoring, are dynamically staged to the target machine, leaving Ecce with a zero installation footprint on compute servers. Additionally, at the time of job submission, attempts are made to trap any errors that might result in a failed or partially completed job, such as directory permissions, invalid run or scratch directories, and low disk space availability. It is important for PSEs and other high-level tools to catch and report these errors before they impact the user. The access mechanism to the compute resource can be controlled on a per-resource basis. There are drawbacks to this design, including the reliance upon client-side UNIX services and csh server-side services. However, it has proved easy to deploy, is secure, reduces the scope of Ecce's responsibility, and allows Ecce to provide support for distributed computing at virtually any site without imposing new server-side security requirements or administrative tasks.

As a proof-of-concept, the current production release of Ecce integrates support for Globus job submission using a combination of the globusrun executable and the Resource Specification Language (RSL) [37]. The implementation relies upon Grid PKI authentication and instantiates an RSL request with information similar to that used by the original job submission components to create the job script. Unfortunately, we have found that RSL does not provide the level of job submission customization necessary to successfully run our applications. Because of this, Ecce is still responsible for generating job scripts and submitting these scripts as single jobs (no parallel jobs) in RSL. Additionally, since there is no mechanism to perform pre-submission validation or to dynamically send output back to the client as it is produced, the Ecce software must log into the target machine using a GSI-enabled ssh to perform the validation and initiate its output monitoring process.

To initiate investigation of how metascheduling will impact PSE architectures, a test-bed integration of Ecce and the Silver MetaScheduler was performed. Silver is an advanced reservation-based metascheduler capable of tracking the cumulative resources of any number of local or geographically isolated, heterogeneous, high-performance computing systems. Silver manages its own queue of meta-jobs and starts these jobs on the system that will best meet the needs of the job (e.g., earliest start time, minimum resource cost). If requested, Silver may also decompose a meta-job into multiple job components, stage these components to various independent systems, and when appropriate, start these job components in unison. Although Silver is still in early stages of development, a number of demonstrations have shown that job scheduling performance can be improved through any of the above-mentioned strategies, thus improving turnaround time and percent utilization and providing benefits to both users and administrators. A successful demonstration of Ecce using Silver to provide metascheduling services across machines at PNNL, the San Diego Super Computing Center, and the National Center for Supercomputing Applications (NCSA) was conducted at the SuperComputing 2000 conference. An NWChem calculation was set up within Ecce, which generated input and associated files for submission to Silver for scheduling with the criteria of minimizing queue wait time. Because of the nature of the physics and mathematics involved in electronic structure chemistry, no tests of decomposed jobs were performed. This demonstration presented some interesting challenges resulting mainly from the fact that at job submission time, the job is not bound to a resource and the customized job script cannot be generated. For the purposes of demonstration, Ecce pre-generated and pre-staged the files to every resource on which the job might eventually land. Once the machine was chosen, Ecce initiated its output data monitoring process.

These test beds and experience with the deployment of Ecce into vastly different environments have led us to investigate solutions that would enable a PSE to transparently support job submission in any of these environments. We propose a job script generation service and an enhanced job submission protocol along the lines of Globus Resource Allocation Manager [37] to provide this functionality.

In our experience with computational chemistry codes, the job scripts needed to set up the execution environment for an application—whether through direct command line submission, Globus RSL, or Silver—require a high degree of customization for each application, for the scheduler, and often to accommodate site policies. Execution scripts set environment variables (e.g., library paths, scratch directories) and parameters that are unknown until the compute resource is selected. They also perform application-specific customization, such as moving executables and data files to each allocated node. In some cases, modification of the input file may also be required. Because of this complexity, we have concluded that there is real value in developing a generic Job Script Service that encapsulates script generation and separates it from client applications, whether they are end-user applications or services such as metaschedulers.

To provide this service, we have investigated the use of our information service for the storage and maintenance of job script templates. Generic templates exist for batch system headers and computational codes within the IS and can be further tailored by resource. The script service locates a script template for the request, queries the information service for the application and machine information, substitutes these values in the script template, and returns the script to the client application. We have prototyped this service using SOAP over HTTP with XML content, and we use SOAP with MIME attachments [40] to deliver the generated script. Figure 3 depicts the use of this service by the PSE job launching mechanism and also the Silver MetaScheduler. This design encapsulates the creation of the complex application execution scripts and can be utilized by any PSE or portal.

We believe that a protocol approach to job submission, such as GRAM, is a sound strategy that will enable applications to access different implementations depending upon their deployment environment without necessitating changes to the PSE. However, we have found existing protocols to have deficiencies when analyzed against our requirements. We believe extensions are required in three areas.

Transparent support for scheduling and metascheduling: To provide seamless operation of a PSE in several environments will require the same protocol and job specification language to be supported by the various job submission mechanisms/services. The details of standardizing the protocol and specification language are being discussed by the GGF scheduling working group.

Generic job progress monitoring: We believe passive monitoring of job output is an important feature of PSEs. By passive monitoring, we refer to a process in which a third-party agent continually monitors the output file for new data. Data of interest to the user is extracted and sent to interested applications. This allows researchers to monitor the progress of long-running jobs and make decisions on whether to continue or make adjustments and restart the job. In a metascheduling environment where a job can be moved dynamically, this agent must also be stopped and relocated. Incorporating this functionality into the protocol guarantees that it will work under any implementation. In the simplest case, the agent could simply stream all of the data; however, providing filtering can greatly reduce the volume of data transferred from server to client. We have developed a generic filtering agent that takes as input an XML syntax specifying regular expression match patterns, which can be nested to define sets of match patterns that can be applied to different steps of the process flow. In the current Ecce implementation, matched blocks are sent back to a client application. In the new architecture we envision sending matched blocks as event notifications. Application codes with source code availability can be modified to publish these event notifications, providing a lightweight decoupled mechanism for real-time monitoring that is independent of any particular framework.

Job Tracking: In an environment where jobs can be dynamically moved to different resources, asynchronous communication back to the user is required. Though this is probably not part of the job submission protocol itself, it is an important aspect of providing robust user environments and should be included with schedulers. A separate protocol based on the Grid Notification Framework [41] could be implemented in a straightforward manner to provide this functionality while achieving a lightweight design based on loosely coupled components.

With these enhancements, PSEs like Ecce can simplify their job launching architecture and support a variety of computing environments.

3.4 Application Integration

Just as a PSE provides access to several kinds of computational resources and middleware services, it also typically provides access to multiple computational codes. Different codes may be included to span scientific disciplines, or they may simply reflect the fact that many similar codes are in use in the research community, each having its particular strengths for a given research problem. Integrating a code into a PSE involves supporting input setup through a user interface, supporting data translations to standard or interchange formats, and defining interaction between other modules in the system. There are a few primary strategies for supporting input setup: define a language for textually specifying input requirements from which an interface can be automatically generated; register information within an IS on how to find and invoke a user-supplied interface; and provide framework-specific tools to create framework-compliant interfaces. Data translations may involve attaching MIME type information to data files and specifying translation services capable of converting, extrapolating, or sub-setting the data. To make one-time creation of reusable interfaces for applications possible, standards for how to define these *application descriptors* are needed. Early-stage discussions regarding such standards have been initiated within the Grid Computing Environments community. Current discussions focus primarily on defining how and where to access applications. Extending these discussions to include protocols for data translation services and even integration protocols could enable application integration to be accomplished through the same service-oriented architecture emerging for middleware.

Ecce includes an architecture for integrating computational chemistry codes into its framework. Currently, this mechanism is specific to the Ecce framework. Full integration is a complex task, and providing a useful integration requires deep knowledge of the computational code, especially with regard to its inputs, dependencies and rules between inputs, and the effect of input specifications on the output. The goal of Ecce's integration architecture is to provide a loosely coupled mechanism that allows and encourages the integration effort to be done by a researcher who has expert knowledge of the application. A fully integrated code allows users to construct or import molecules, assign basis sets, run the job, monitor job output, and perform post-run analysis using existing Ecce tools while employing a customized application to specify input details. To support full integration requires an integrator to provide several things: an application for user selection of inputs; a script to combine the molecule, basis set, and input options into an input file; a job script template; and an output descriptor file with associated conversion scripts for parsing and translating output data as discussed previously.

Because Ecce already includes complex tools for specifying a chemical system and assigning basis sets, all that is needed is a customized interface for selecting algorithms, setting tolerances, and specifying other options. Based on input from collaborators and colleagues, we opted to pursue a multi-pronged approach to provide the desired flexibility. First, we defined a small protocol that defines the interaction between chemistry code interfaces and an Ecce editor. This allows users to create an interface in the toolkit of their choice (e.g., TCL or java). There are many such toolkits available, and many researchers are comfortable with one or more of them. The Ecce editor manages complex details like communicating with other tools, supporting a final edit option, maintaining consistency, and initiating job launching while delegating selection of input setup options to a third-party component. Second, we felt that a system with rules-based capabilities for specifying dependencies between inputs was very important for building interfaces that guide the user through valid combinations of options, especially in cases where there are hundreds of interrelated options where invalid or poor choices can result in wasted compute time. To support this capability, we have provided a package based on Amulet [42] that supports the construction of dialogs that conform to the look and feel of Ecce while incorporating rules logic. Our intent is to support a Python integration, thus providing a very accessible mechanism for creating sophisticated tools that does not require Ecce development expertise. A final strategy is to investigate options for generic GUI generation using a standard layout description language, perhaps augmented to support rule-based logic. Such an approach would provide a powerful non-programming method for creating user interfaces. However, without sophisticated editors, creating such an XML file is currently a more daunting task than using Python or TCL. To make this approach practical for our target audience, layout tools that can generate a standardized user interface schema description are required.

Ecce's application integration strategy has the advantages of being loosely coupled in the sense that source code interaction with the chemistry application is not required, and the required functionality is independent of the rest of the Ecce framework. However, the integration is framework specific. Progress in creating standard protocols and schemas for application integration could have enormous benefits in reducing duplication of effort both in defining frameworks and in integrating codes, and would also allow application builders to make their code available through a number of frameworks through a one-time effort.

4.0 Summary

The architecture changes discussed in this paper can have significant immediate and long-term benefits for PSEs. Deployment costs are reduced at the same time that deployment options can be tailored to each installation's cost and performance constraints. Perhaps more importantly, the canonical PSE architecture is evolving from a single integrated application suite to applications sitting on a domain-coordination layer built on distributed PSE middleware services that in turn leverage lower-level Grid services. Our redesign of the capabilities required by Ecce contributes to the requirements and design criteria for several PSE/portal middleware components.

Designing around protocol- and service-oriented architectures reduces barriers to system evolution. Without the need for complete agreement on a schema, components can be developed or extended independently. Third-party capabilities such as data mining and knowledge discovery agents can be readily accommodated. The maturity of HTTP-related mechanisms for supporting multiple security options and providing scalable performance and fault tolerance provides a wide range of options for deployment. The layering of standards such as SOAP and XML forms the basis for building loosely coupled PSE components.

The release of Ecce 2.0 with the updated architecture is already reaching a much wider user base. The reduced deployment costs have triggered many new downloads. Ecce is now positioned to deliver subsets of its functionality through portal services. Calculations and associated documents are already accessible from Web browsers. Using standard Web application development tools, dynamic views of users' existing projects and calculations can be generated. These views can include links to services that generate editors, images, applets, and/or Virtual Reality Modeling Language (VRML) files, all of which can be viewed from browsers with readily available plug-ins. The ability to deploy individual Ecce components as standalone tools and to use external applications to view Ecce data are also increasing user interest.

PSEs provide additional functions that are not discussed in this paper. For example, on the application side, this includes support for task-based workflow and real-time collaboration capabilities. On the infrastructure side, the movement toward a service-based computing environment places new demands on the security infrastructure, policy management, and service discovery and negotiation. Many of these topics still represent significant research challenges, but we believe the general principle discussed here of decoupling PSEs from the specifics of the service implementation, thus allowing them to be used in a variety of settings, will also be important in these areas. The GGF has emerged as a valuable community forum for discussing such issues and addressing the challenges through the establishment of standard protocols, interfaces, and tools. We plan to contribute both research and requirements to this process and continue to use Ecce as a test-bed platform for Grid services.

Acknowledgements

The authors would like to acknowledge the work of Dave Jackson (SuperCluster.org) and Scott Jackson (PNNL) for their work on the Silver MetaScheduler, their efforts in pulling together the SuperComputing 2000 Ecce/Silver demonstration, and their participation in discussions on architectures for metaschedulers and problem-solving environments. We would also like to acknowledge Jim Myers (PNNL) for his insights on PSE architectures and for his thoughtful comments that helped to guide this paper.

Pacific Northwest National Laboratory is operated by Battelle for the U.S. Department of Energy. The William R. Wiley Environmental Molecular Sciences Laboratory, located at PNNL in Richland, Washington, is a national scientific user facility sponsored by the U.S. Department of Energy's Office of Biological and Environmental Research.

References

[1] Gallopoulos S, Houstis E, and Rice, JR. 1994. Problem-solving environments for computational Science. *IEEE Computational Science and Engineering* 1994; Summer:11-23.
[2] Rice, JR and Boisvert, RF. 1996. From scientific software libraries to problem-solving environments. *IEEE Computational Science & Engineering* 1996; Fall:44-53.
[3] The Grid Blueprint for a new Computing Infrastructure. Morgan Kaufman Publishers, Inc; Edited by Ian Foster, Carl Kesselman, 1999.
[4] Fox G, Haupt T, Akarsu E, Kalinichenko A, Kim K, Sheethalnath P, Youn C. 1999. The Gateway System: Uniform Web Based Access to Remote Resources 1999; ACM Java Grande Conference.
[5] Accelerated Strategic Computing Initiative (ASCI), http://www.llnl.gov/asci/pse
[6] Problem Solving Research, Virginia Tech University, http://vtopus.cs.vt.edu/~pse/shortintro.html
[7] Euro Tools Problem-Solving Environments, http://www.irisa.fr/EuroTools/Sigs/PSE.html
[8] Problem Solving Environments, http://www.cs.purdue.edu/research/cse/pses
[9] Allen G, Goodale T, Lanfermann G, Radke T, Seidel E. 2000. The Cactus Code: A Problem Solving Environment for the Grid 2000. *Proceedings of the First EGrid Meeting*.
[10] Molecular Science Software Suite, http://www.emsl.pnl.gov:2080/mscf/about/descr_ms3.html
[11] Dixon DA, Dunning TH, Dupuis M, Feller DF, Gracio DK, Harrison RJ, Nichols JA, and Schuchardt KL. Computational chemistry in the Environmental Molecular Sciences Laboratory. Plenum Publications: 1999; Book Chapter in "High Performance Computing."
[12] Jones DR, Keller TL, Schuchardt KL, Taylor HL, and Gracio DK 1999. Extensible Computational Chemistry Environment Data Centered Framework for Scientific Research. *Domain-Specific Application Frameworks: Manufacturing, Networking, Distributed Systems, and Software Development* 1999; Chapter 24, Vol. Three, No. 0-471-332801.
[13] Gaussian98, http://www.gaussian.com
[14] GAMESS-US, http://www.msg.ameslab.gov/GAMESS/GAMESS.html
[15] Schuchardt KL, Myers JD, Stephan EG. 2001. Open Data Management Solutions for Problem Solving Environments: Application of Distributed Authoring and Versioning to the Extensible Computational Chemistry Environment. *Proceedings HPDC-10* 2001.
[16] RFC 2518 HTTP Extensions for Distributed Authoring – WEBDAV, http://andrew2.andrew.cmu.edu/rfc/rfc2518.html
[17] Simple Object Access Protocol (SOAP) 1.1, http://www.w3.org/TR/SOAP/
[18] Silver MetaScheduler, http://www.supercluster.org/projects/silver
[19] Fielding RT, Whitehead Jr. EJ, Anderson KM, Bolcer GA, Oreizy P, and Taylor RN. 1998. Web-based development of complex information products. *Communications of the ACM* August 1998; **41(**8):84-92.
[20] Scientific Annotation Middleware, http://collaboratory.pnl.gov/docs/collab/sam/
[21] DAV Searching & Locating – DASL, http://www.webdav.org/dasl/protocol/draft-dasl-protocol-00.html
[22] WebDAV Ordered Collections Protocol, http://www.ics.uci.edu/pub/ ietf/webdav/collection/draft-ietf-webdav-ordering-protocol-02.txt
[23] Goals for Web Versioning, http://www.webdav.org/deltav/goals/draft-ietf-webdav-version-goals-01.txt
[24] Rust PM, Rzepa HS, Write M, and Zara S. 2000. A universal approach to web-based chemistry using XML and CML. *Chem Commun* 2000; 1471-1472.
[25] Math Markup Language, http://www.w3.org/TR/REC-MathML/
[26] Extensible Scientific Interchange Language, http://www.cacr.Caltech.edu/SDA/xsil/
[27] Allcock W, Bester J, Breshnahan J, Chervenak A, Liming L, and Tuecke S. 2001. GridFTP: Protocol Extensions to FTP for the Grid. Internet Draft March 2001, http://www.gridforum.org.

[28] Butler R, Engert D, Foster I, Kesselman C, Tuecke S, Volmer J, and Welch V. 2000. Design and deployment of a national-scale authentication infrastructure. *IEEE Computer* 2000, **33**(12):60-66.

[29] Thompson M, Johnston W, Mudumbai S, Hoo G, Jackson K, Essiari A. 1999. Certificate-based Access Control for Widely Distributed Resources, *Proceedings of the Eighth Usenix Security Symposium* August 1999.

[30] Czajkowski K, Fitzgerald S, Foster I, Kesselman C. 2001. Grid Information Services for Distributed Resource Sharing. *Proceedings HPDC-10*.

[31] Directory Services Markup Language, http://www.dsml.org/1.0/dsml.html

[32] von Laszewski G, Fitzgerald S, Didier B, Schuchardt K. 2000. Defining Schemas for the Grid with GOS, the Grid Object Specification, Grid Forum Working Group Document GIS-WG-1. http://www.gridforum.org.

[33] Java Naming and Directory Interface (JNDI), http://java.sun.com/products/jndi

[34] XQuery 1.0: An XML Query Language, http://www.w3.org/TR/xquery/

[35] XQuery 1.0 Formal Semantics,W3C Working Draft, 7 June 2001, http://www.w3.org/TR/query-semantics/

[36] Global GridForum, http://www.gridforum.org

[37] Foster I, Kesselman C. 1997. Globus: A Metacomputing Infrastructure Toolkit. *Intl J. Supercomputer Applications* 1997; **11**(2):115-128.

[38] Grimshaw AS, Wulf WA, and the Legion team 1997. The Legion Vision of a Worldwide Virtual Computer. *Communications of the ACM* January 1997; **40**(1).

[39] Casanova H and Dongarra JJ 1997. Netsolve: a Network Server for Solving Computational Science Problems. *The International Journal of SuperComputer Applications and High Performance Computing* Fall 1997; **11**(3):212-223.

[40] SOAP Messages with Attachments, Dec. 2000, http://www.w3.org/TR/2000/Note-SOAP-attachments-20001211

[41] Gullapalli S, Czajkowski K, Kesselman C, Fitzgerald S. The Grid Notification Framework. Global Grid Forum, Grid Working Draft (GWD-GIS-019-01), http://www-unix.mcs.anl.gov/gridforum/gis/reports/notification/GIS-GridNotificationFramework.pdf

[42] Myers BA, McDaniel RG, Miller RC, Ferrency A, Faulring A, Kyle BD, Mickish A, Klimovitski A, and Doane P. 1997. The Amulet Environment: New Models for Effective User Interface Software Development. *IEEE Transactions on Software Engineering* June 1997; **23**(6):347-365.