

# Open Data Management Solutions for Problem Solving Environments: Application of Distributed Authoring and Versioning to the Extensible Computational Chemistry Environment

Karen Schuchardt, James Myers, and Eric Stephan  
Pacific Northwest Laboratory

[Karen.Schuchardt@pnl.gov](mailto:Karen.Schuchardt@pnl.gov), [Jim.Myers@pnl.gov](mailto:Jim.Myers@pnl.gov), and [Eric.Stephan@pnl.gov](mailto:Eric.Stephan@pnl.gov)

## Abstract

*Next-generation problem solving environments (PSEs) promise significant advances over those now available. They will span scientific disciplines and incorporate collaboration capabilities. They will host feature-detection and other agents, allow data mining and pedigree tracking, and provide access from a wide range of devices. Fundamental changes in PSE architecture are required to realize these and other PSE goals. This paper focuses specifically on issues related to data management and recommends an approach based on open, metadata-driven repositories with loosely defined, dynamic schemas. Benefits of this approach are discussed and the redesign of the Extensible Computational Chemistry Environment's (Ecce) data storage architecture to use such a repository is described, based on the distributed authoring and versioning (DAV) standard. The suitability of DAV for scientific data, the mapping of the Ecce schema to DAV, and promising initial results are presented.*

Index Terms—metadata, problem solving environments, scientific data management, self-describing dynamic schemas, WebDAV protocol, XML.

## 1. Introduction

Scientific problem solving environments are complex computing systems that seek to integrate the activities necessary to accomplish high-level domain tasks [1][2]. They may include components for managing scientific workflow, tracking data pedigrees, transforming and filtering data, analyzing and visualizing results, automating feature extraction, and annotating records. As described by Gallopoulos et al., they also “use the language of the target class of problems, so users can run them without specialized knowledge of the underlying computer hardware or software” [1]. Thus, at the cognitive level, a PSE encodes domain knowledge, and, to varying degrees, enforces or guides users toward best practices. This characteristic is a powerful benefit of PSEs, particularly for novices or occasional users.

Unfortunately, contemporary PSEs tend to embed domain knowledge into the design of persistent data objects and the data store itself, requiring early agreement about best practices, as well as a complete domain ontology. These undesirable impacts may result:

- As the scope of a PSE increases, the number of parties that must agree upon best practices and ontology, and the resultant data structures, become untenably large.
- As components are incorporated into a PSE, negotiation is required between the component developer and the PSE framework designers. Creating a component that fits within a PSE framework often makes the component unusable in other PSEs or as a stand-alone application.
- As best practices evolve or PSEs are extended to support users with different goals, the data structures and control flows must change. All components must be changed simultaneously and the existing data structures migrated.
- As PSE usage expands, the need for federated access to multiple data stores at multiple locations is necessary to provide multi-scale and/or cross-disciplinary capabilities. With current practices this is difficult and costly because of incompatible access mechanisms and non-integrable, non-discoverable schemas.

These four problems reduce the ease of PSE evolution, create undesirable coupling between components, and introduce up-front delays in creating and extending PSEs. Advances in data storage architectures will be required to mitigate these problems and enable next-generation PSEs. The work presented here is based on a concept for open, metadata-driven repositories whose schema can be dynamically extended and altered without requiring changes to existing PSE components. This concept differs in two respects from the use of metadata in digital libraries and scientific archives. First, we use the repository as the primary PSE persistence mechanism. Second, it is expected that no individual component, data store included, need understand or even be aware of the entire schema. This significantly reduces the coupling

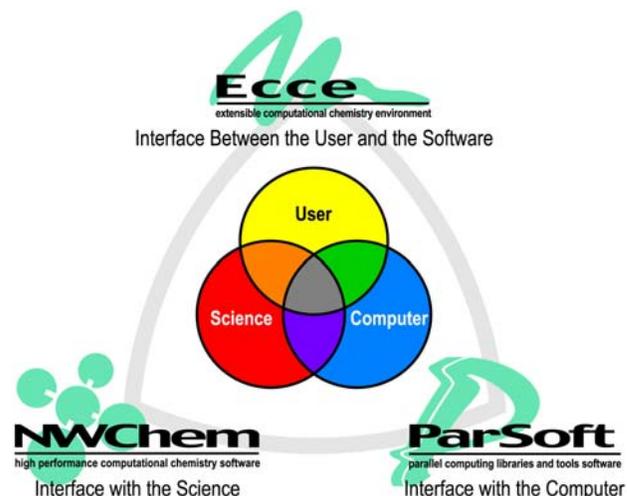
between components and between the data store and components, thus reducing the level of agreement necessary to create and evolve the PSE. This paper details our concept, presents results of an initial implementation of such an architecture within an existing PSE at the Pacific Northwest National Laboratory, and discusses some motivating scenarios made possible by this new design.

## 2. Background

The Pacific Northwest National Laboratory (PNNL) has several ongoing efforts in developing PSEs, collaboratories, and large-scale data management systems. These efforts focus in different scientific and engineering domains and have developed systems tailored for their respective communities with their differing requirements for security, computation, and data scaling. Unfortunately, the different design choices made with respect to the data management components have so far constrained the scope of applicability of otherwise generic components and pose significant barriers to the development of a single unifying architecture with best-of-breed capabilities.

In this paper, the context of PNNL's Ecce is used to explore these issues and to present an initial implementation of an architecture that addresses them. Ecce is one component of the Molecular Science Software Suite (MS3) [3]. 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 is a domain-encompassing PSE composed of a suite of tools. Ecce assists chemists with many tasks, including the management of projects and calculations, construction of complex molecules and basis sets, generation of input decks, distributed execution of computational models, real-time monitoring, and post-run analysis [4] [5]. Ecce and MS3 have been operational since 1997 and won an R&D 100 award from R&D Magazine in 1999.

Ecce was designed nearly eight years ago around object level integration. At the core of Ecce is an object-oriented chemistry data model that supports management and manipulation of computational data, experimental data, and metadata. Ecce designers elected to apply object database technology to the management of this data. Until recently, persistent data and the model itself were implemented using an object-oriented database



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

management system (OODBMS) [6] [7]. Persistent object classes, representing molecules, basis sets, projects, calculations, and jobs, provided the core for tool development. These object classes also provided the management of data, metadata, and complex relationships between data objects. Use of object-oriented design and the OODBMS allowed Ecce to provide a high degree of interaction between components. At the time of its development, the Ecce architecture represented an innovative approach to managing the complex computational chemistry research data [8].

Despite its success, the Ecce design has significant limitations when analyzed in Year 2001 terms. OODBMS systems have failed to mature and standardize as rapidly as expected. As described by others [9], it is nearly impossible to gain complete agreement between vendors on anything concerning object database systems. Other significant problems include proprietary binary formats, tight coupling between the programming language and the OODB, lack of application development tools, and a schema evolution process made painful by outdated schema/application compilation cycles. Object databases have design principles opposite from the web-based thin-client/fat-server architecture and cannot effectively leverage the plethora of new technologies being developed around this architecture.

Our vision for next-generation PSEs is one where independently designed and developed components are rapidly combined to deliver more powerful solutions and reach larger communities of researchers while sharing development costs among the interested parties. For example, Ecce is now adding support for the field of molecular dynamics. This change entails enhancements and additions to the object model and schema that,

without changes to the underlying data management system, would have amplified the issues previously mentioned.

Similar problems would be expected during the inclusion of third-party tools to compare theoretical and experimental results, to model chemical kinetics, or to add functionality related to biology or materials science. Within PNNL, two existing projects are targeted for integration with Ecce in the near term: a large-scale hierarchical data archive and an Electronic Laboratory Notebook system. These systems, which were developed in different languages with different object schemas and data management systems, are essentially third-party applications. Although a useful level of integration has been accomplished with the Electronic Laboratory Notebook, the use of independent data stores makes the integration brittle with respect to the evolution of either object model. Direct integration with either of these systems is undesirable due to the resulting tight coupling and impact on deployability and maintainability. Thus, an alternate strategy is required. The work to lower development costs and reduce deployment barriers for PSEs reported here is therefore motivated by practical as well as theoretical considerations. We sought to solve several pressing deployability and integration issues in a manner that would be widely applicable to PSEs in general.

### 3. Approach

A key observation leading toward an open PSE data management architecture was the realization 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—coupled with the issues previously discussed—leads to several design criteria:

- 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 (that is, 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 can also 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 object from the data storage mechanism via a standard protocol(s). Using a standard protocol for describing data management operations helps to maintain the schema independence previously described. Additionally, a standard protocol allows the selection of 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. They can also be deployed to a much broader range of users. Several Ecce-related scenarios, enabled by this design, will be described.

#### 3.1 Technology Selection

The architecture discussed above could be implemented using a variety of technologies. Data objects that support arbitrary metadata can be developed using the Common Object Request Broker Architecture (CORBA) [10]. Similarly, Version 3 of the Lightweight Directory Access Protocol (LDAP) allows extension of existing entries with new metadata through the use of the extensibleObject class. However, the combination of the Web's Hyper Text Transfer Protocol (HTTP) [11], the Extensible Markup Language (XML) [12], and the Distributed Authoring and Versioning (DAV) protocol, also known as WebDAV [13], provides the closest conceptual mapping to our design goals. DAV, an extension to http 1.1, was originally designed to support collaborative authoring [14]. It provides structured XML-encoded requests for manipulating MIME-typed "documents" (get, put, move, copy, lock) and associated metadata (propfind, proppatch). 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. DAV “documents” are not restricted to text-oriented formats and are more analogous to files or binary large objects. New properties can be added at any time, and applications can manipulate arbitrary subsets of properties. For example, an application can request only the values of properties 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 the scientific data management domain.

DAV currently supports only a simple, unordered container/contains relationship, but the wide range of data relationships used in PSEs (for example, 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. Extensions to DAV, such as DAV Searching and Locating (DASL), Advanced Collections, and Versioning that are currently under development promise additional PSE-relevant capabilities [15], [16], [17]. XML provides rich capabilities for schema description (XML Schema) and translation (XSLT), 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 (for example, the Chemical Markup Language (CML) [18], MathML [19], the Extensible Scientific Interchange Language (XSIL) [20]). Finally, 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.

## 3.2 Implementation

**3.2.1 DAV Server.** DAV is quickly gaining popularity in the Web industry. Before the end of 1999, the Apache Software Foundation, IBM, and Microsoft had already deployed DAV servers as extensions to Web servers. Client-side support is offered by the Microsoft Office 2000 suite and Java, C++, and Python tool kits. Database vendors are also moving to support DAV. More recently, Web development products have incorporated DAV capabilities including Macromedia Dreamweaver, Adobe Photoshop 6, GoLive 5, and several others. This broad acceptance of DAV is rapidly expanding the server-side options available and the emergence of optimized, high-performance implementations can be expected. In choosing a DAV server implementation for development use in this project, we emphasized cost, robustness, and protocol conformance over performance. The OpenSource `mod_dav` extension for the Apache Web Server fit these criteria. The `mod_dav` implementation uses file system files and directories to provide persistence for data objects and collections, respectively.

Metadata properties are stored in a *hash* table within a database manager (DBM) formatted file, one file per document or collection. Either Simple DBM (SDBM) or Gnu DBM (GDBM) may be used. SDBM imposes a 1-kilobyte (KB) size limit on individual metadata values, has a default initial size of 8 KB and requires fewer steps during the server build process. GDBM imposes no size restrictions, has higher performance, requires a few more steps during the server build process, and has a default initial database size of 25 KB [21]. With both implementations, manual garbage collection utilities must be used to reclaim space associated with changed or deleted metadata properties.

Under conditions expected to be representative of typical PSE requirements, the Apache Server and `mod_dav` module were tested for DAV protocol compliance, robustness, and performance. Several server configurations were used to assess the effects of key parameters, such as network connection, memory, and operating system features. All servers were built using Apache 1.3.11, `mod_dav` 1.1, and GDBM 1.8. Each was configured to use basic authentication, to accept persistent connections with limits of 100 connections per minute, 15 seconds between requests, and a minimum of 5 daemons. The test client machine was a 450-MHz Sun Ultra™ 60 with 512 MB RAM. Our client-side software consisted of internally developed C++ classes with 1500-byte packets to mirror our typical TCP packet sizes and the xerces 1.3 DOM parser for processing results.

As of this writing, no public protocol compliance test suites exist for DAV. Test programs were developed to test each DAV method (`put`, `proppatch`, `propfind`...). In addition, both the Microsoft Office 2000 tools and a Java DAV Explorer client [22] were used as interactive client-side applications test tools. As a result of this testing and the robustness and performance tests described next, we did not find any major DAV protocol compliance issues except for the few noted on the DAV development Web site [23]. These issues did not present any significant problems for the anticipated use.

Tests were performed to verify upper size limits and ensure the server behaved properly when encountering large metadata and documents. With `mod_dav` and GDBM, properties as large as 100 MB and documents as large as 200 MB were created repeatedly without problems. Document size restrictions are those imposed by the underlying file system. Although these tests involve property and data sizes much larger than those expected in DAV’s prototypical use in document management, no problems were encountered, convincing us that `mod_dav` would be suitable for our application. The maximum size of metadata properties is configurable and, as an initial (post-testing) value, we set a limit of

10 MB per property. It should be noted that storing an XML-based metadata property using mod\_dav currently requires double the memory of the property: one copy with the XML request body and another copy that is the key/value pair extracted from the body. Further, effective denial-of-service attacks can be created by repeatedly sending large XML request bodies. Thus, in a production system, the maximum should be set as low as possible for a given application.

When initially considering DAV as the basis for a PSE data management architecture, it was unclear whether overall performance of a request-response protocol such as http would be comparable to alternative strategies such as an OODBMS with a cache-forward architecture as used by Ecce. To assess the feasibility, tests that mimic typical PSE access patterns were developed. PSEs typically include capabilities to traverse through data sets and examine properties, query and replace properties, add new properties as tasks are performed, copy entire task sequences, and delete task sequences (Table 1). Additionally, to mimic storing and retrieving computational input and output files, the performance of get and put were tested (Table 2). All tests were performed during off-hours to minimize the effect of network traffic.

Table 1 includes both elapsed and CPU time to help determine whether performance costs were occurring on the client or the server side. Roughly, the CPU time represents client-side processing time while server processing time can be determined as elapsed time minus CPU time with some time allocated to moving the requests and responses across the network. Given the

relatively small sizes of the metadata and the 150-Mbit/s network connection, network transport has little impact on these tests, thus providing a reasonable assessment of server performance. For these tests, we created 50 documents, each with 50 properties of 1 KB in size and performed operations to query for selected data, traverse the data, copy it, and remove it. Server responses were parsed and moved into generic hierarchical object representations. As shown, metadata operations on individual objects are quite fast. However metadata operations on a large number of objects added up to several seconds. For these operations, the bulk of the time was spent on client-side processing. This percentage can be attributed to the current use of a parser based on the Document Object Model (DOM) [24] to parse the response and create custom data structures. Significant improvements can be expected by converting to a Simple API for XML (SAX) [25]-style parser. (SAX parsers do not build an in-memory representation of the entire XML document as DOM parsers do, eliminating significant overhead.) In addition, alternative server-side implementations that do not operate on many small metadata databases as mod\_dav does are expected to provide significant server-side performance improvements. In this particular test case, 50 separate database files were opened, queried, and closed. With data distributed across many documents and collections, copy and remove operations can be costly on the server side, but preliminary testing with journaling file systems show that significant performance increases can be expected for these operations as well.

Table 2 shows that our implementation of http/put performed comparably with a standard binary-mode ftp

**Table 1. Performance results of typical PSE operations – elapsed and CPU time**

	Get all metadata. Depth=0 <sup>(a)</sup>	Get selected metadata Depth=0 <sup>(b)</sup>	Get selected metadata for 50 objects depth=1 <sup>(c)</sup>	Get metadata for 50 objects <sup>(d)</sup>	Copy hierarchy with 50 objects totaling 4.5MB <sup>(e)</sup>	Remove hierarchy with 50 objects totaling 4.5MB <sup>(f)</sup>
elapsed <sup>(g)</sup>	0.068 s	0.055 s	2.732 s	3.032 s	3.482 s	1.782 s
cpu	0.04 s	0.03 s	2.04 s	1.93 s	0.14 s	0.01 s

- (a) Get all metadata on single document including system properties and 50 test properties each 1 KB.
- (b) Query for 5 of the properties on a single document.
- (c) Use depth=1 capability to query for metadata for 5 of 50 properties on 50 objects within a collection.
- (d) Query for 5 of 50 properties on 50 objects - one at a time.
- (e) Copy collection of 50 documents each containing 50 1 KB application properties.
- (f) Remove collection created by copy step.
- (g) Sun Enterprise 450 running Solaris 2.6 with 512 MB memory and 150-Mbit/s network connection. This machine served as Ecce's OODB server.

**Table 2. Performance of binary ftp vs http/put**

	ftp 20 MB mem to mem using /tmp	ftp 20 MB Local file to local file	ftp 200 MB Local file to local file	Put 20 MB Local file to local file	Put 200 MB Local file to local file
Enterprise 450 <sup>(a)</sup>	1.5 s	3.3 s	30 s	3.0 s	30 s

- (a) Sun Enterprise 450 running Solaris 2.6 with 512 MB memory and 150-Mbit/s network connection. This machine served as Ecce's OODB server.

client. It also demonstrates that network bandwidth is the primary driver for moving large amounts of data: our client and server did not introduce bottlenecks. As of this writing, no performance tests have been run when using alternative authentication mechanisms, such as public key certificates, and no tests of scaling through the use of multi-processor, multi-server load-balancing systems have been done. Because these issues are related to the Apache server, rather than the `mod_dav` module, the performance hit for secure communications and overall server scalability is expected to be similar to those reported for generic Web applications.

The `mod_dav`/GDBM/Apache server remained stable during approximately 6 months of testing using all of the scenarios described above. During this time, no loss of persistent data or any data transmission loss was experienced. Even without performance enhancements such as pipelining and event-based XML parsing, the performance, compatibility, and reliability tests provided confidence that a reliable, deployable system could be built with the current `mod_dav` implementation.

**3.2.2 Data Access Architecture.** Figure 2 portrays a high-level view of the Ecce data architecture. As shown, although the system uses the Apache/`mod_dav` server, the system can take advantage of any service that implements the DAV protocol. On the client side is a multi-layered architecture designed to accomplish several objectives: isolate data access to support plug-in protocol migration and enhancements in the future, encapsulate object access behind a factory/object layer for easy migration of existing object-based applications, and provide a generic data and metadata layer for flexible access to raw data for future development work. Existing Ecce applications can continue to work in terms of its rich set of C++ classes. Factory modules in the object layer encapsulate access to persistent data using implementations of the Data Storage Interface, which maps requests for manipulating data and

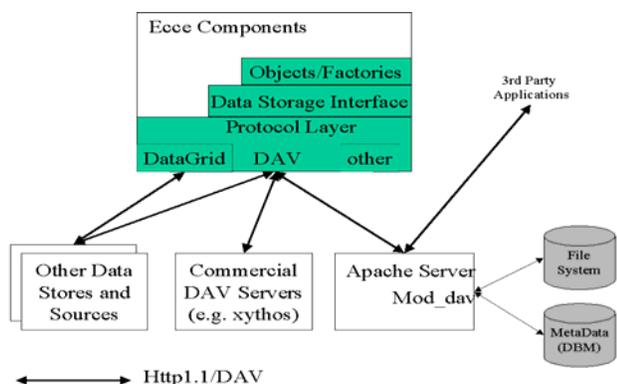


Figure 2. Data access architecture overview

metadata into protocol-specific operations. While DAV is the only protocol currently implemented, a separate data storage interface will reduce the changes required to provide native-protocol access to data grids or to incorporate high-performance extensions to DAV - for example, a GridDAV analogous to GridFTP [26].

The initial DAV client implementation, based on C++ http classes developed at PNNL and the Apache xerces 1.3 XML DOM parser, is blocking and supports persistent connections, but not pipelining. Further optimizations of this implementation, using a SAX parser for example, as well as the extension of the architecture to include a client-side cache, are anticipated. As previously noted, the data store is decoupled from Ecce and its only requirement is DAV compliance.

**3.2.3 Ecce Schema Mapping.** The replacement of the Ecce OODBMS data store with the new architecture has required examination of the use of persistent classes in Ecce and decisions about how to map their structure, content, and relationships into the DAV constructs of collections, documents, and metadata. Ecce had 70 classes “marked” for persistent storage, including relatively simple types, such as dates, and complex class hierarchies that include abstract classes for modeling experiments and calculations, output data properties, molecules, basis sets, and compute jobs. For brevity, the discussion of this mapping process is limited to a subset of the data model – the calculation model. A simplified version of the class model in Unified Modeling Language (UML) notation [27] is shown in Figure 3.

The inheritance in this model provides semantics through virtual methods, as well as through data derivation. Briefly, the model shows a study subject (Molecule) on which a task of an Experiment is performed, the results of which are a series of n-dimensional output Properties. The focus of the model is on simulated experiments or calculations. All the

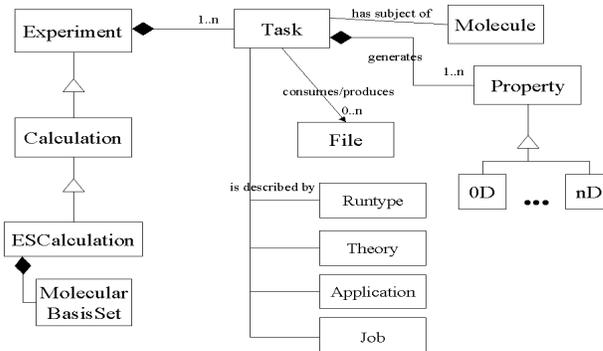


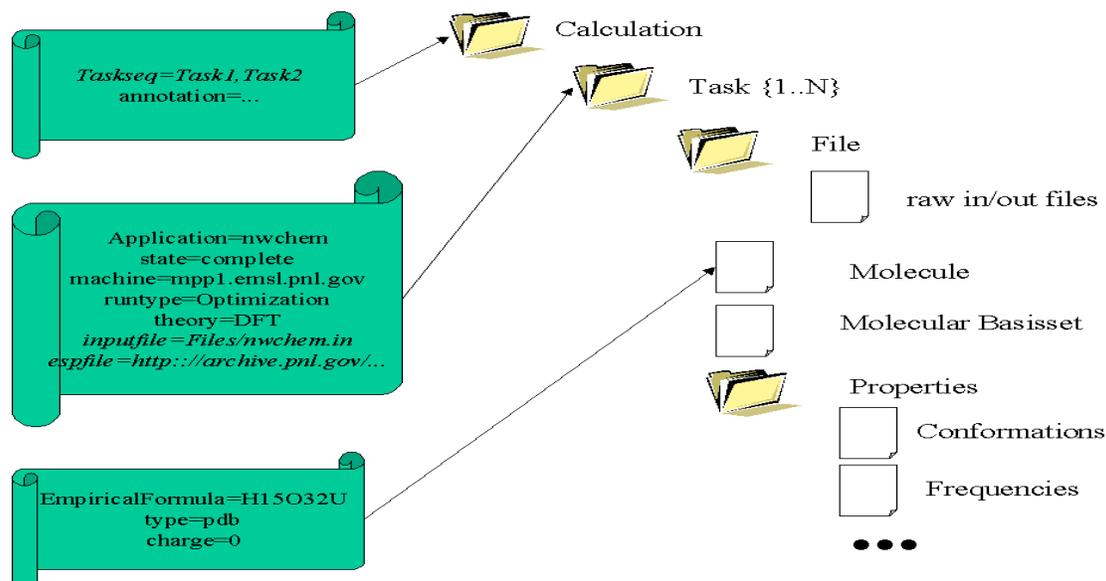
Figure 3. Simplified calculation object model

information needed to reproduce the calculation and provide historical context or post-analysis capabilities is captured. The mapping of the model to DAV can be somewhat simplified because the DAV structure does not need to explicitly capture the full inheritance semantics. These semantics can be applied in the object factory layer for applications in which they are important. For example, to ease the migration of existing Ecce applications that work directly with objects in Figure 3, the object/factory layer of Figure 2 provides the as was previously done through the OODBMS.

Figure 4 depicts how the model was mapped to DAV constructs. In general, objects recognizable by domain scientists were mapped to separate DAV documents. This strategy allows the lowest granularity of access to raw data, minimizing overhead for tools or agents that only care about certain subsets of data and reducing coupling at the data level. It also allows metadata attachment at the lowest granularity. Alternative strategies exist, but we believe they have significant drawbacks with respect to our objectives. For example, because DAV supports arbitrary XML-encoded metadata values, we could have chosen to include related objects within a single document. However, objects mapped as properties cannot themselves have DAV-accessible metadata properties. Objects mapped as properties also become accessible only through their relationship to the document's main object, severely limiting their ability to participate in multiple relationships and reducing their visibility to other applications.

In the initial implementation, hierarchical relationships were mapped into the DAV container/contains relationship provided by collections. Thus, the list of tasks in a calculation is located through the collection mechanism. This collection-based structuring provides convenience when viewing the data store through standard DAV browsers, but does bend our rule about schema independence. In future implementations, as shown in italics in Figure 4, we expect to implement relationships through properties, making the meaning of the relationship available to other programs and allowing the physical layout of objects in DAV to be adjusted dynamically and independent of the metadata. For example, an application or a DAV implementation might elect to store large documents on an archive system, or perhaps store all documents of a given type, such as 3D molecular structures, in a single hierarchy for easier algorithmic processing. Because the document will have self-describing structure, the DAV structure can be reorganized without breaking existing applications, as long as applications interpret the structure dynamically through the metadata. This “*virtual document*” approach increases the granularity of access and assures that all objects are independently accessible and can have their own metadata properties. It also enables the dynamic creation of relationships discovered and defined by third-party agents.

The data members of individual Ecce objects shown in Figure 3 were mapped to a combination of DAV document data and DAV document metadata properties.



**Figure 4. Calculation model mapped to DAV constructs. Scrolls represent metadata for documents/collections.**

Mapping decisions were based on assumptions about other applications that might want to discover, annotate, and manipulate the individual data members. Although these mapping decisions were somewhat arbitrary, the tendency was to decompose Ecce objects as much as possible to increase flexibility, stopping at the point where community standard data structures exist. For example, Ecce's Molecule object was mapped to a Protein Data Bank (PDB) [28], simple XYZ, or custom encoded molecular geometry with metadata properties encoding the format of the raw data, empirical formula, symmetry group, and charge state. Thus, applications could search the data store for DAV documents matching the formula metadata and render a 3D display of the molecule without understanding the rest of the Ecce schema. Where standards do not currently exist, plain text or XML markup (where appropriate) is applied to the data, as is done for the Molecular Basisset document.

For metadata properties, a single "ecce" namespace was defined. As conventions mature and usage becomes widespread, the project will migrate to community standard conventions (for example, CML and naming standards for computational science developed within the Global GridForum [29]).

**3.2.4 Data Migration.** Ecce has been operational for a number of years, and existing OODB data sets must be converted to the new storage system. We have conducted preliminary conversions of two of our larger databases, which contain a total of 259 calculations represented by about 420,000 objects with a combined size (excluding raw data files) of 35 MB on our OODB server. We found that the disk requirements increased by about 10% when using mod\_dav with SDBM and 25% when using GDBM. The bulk of the increase was due to mod\_dav: each document or collection may have an associated database.

With default sizes of 8 KB and 25 KB for SDBM and GDBM respectively, there is significant unused but allocated space. Some of the difference in size can also be attributed to the fact that binary formatted objects such as doubles are typically more compact than textual/XML representations of the same data. Finally, these particular data sets were on very small chemical systems with correspondingly small output dataset sizes. For studies on larger systems, the metadata databases will be a much smaller percentage of the total space used. While these differences can be explained, we were still somewhat surprised because our OODBMS also creates its own overhead, using hidden segments to optimize performance. Alternative back-end DAV solutions could be selected to provide more efficient storage, though at current storage costs this is not a pressing concern for Ecce.

## 4. Discussion

A public beta release of Ecce with the new storage architecture was released the first quarter of 2001, and design work for adding DAV capabilities to PNNL's electronic notebook have begun. The production release of Ecce (2.0) is scheduled for July 2001. As described in following paragraphs, we believe that porting Ecce to the new architecture meets our objectives: it satisfies our goal of creating a lightweight data storage architecture with dynamically evolving schema and loose coupling at the data access level. It will provide a useful platform for further research and development efforts.

All of the Ecce applications have been converted to the new storage architecture, enabling further performance assessments. Table 3 summarizes size, application startup time, and the operation of each tool loading its set of data for a typical calculation. The selected calculation is a Uranium Oxide surrounded by 15 water molecules

**Table 3. Ecce 1.5 vs. Ecce 2.0 beta Performance Summary for Ecce Tools (The client is a Sun Ultra 60. Times are elapsed time.)**

	<b>Builder</b>	<b>BasisTool</b>	<b>Calc Editor</b>	<b>Calc Viewer</b>	<b>Calc Manager</b>	<b>Job Launcher</b>
<b>Ecce 1.5</b>						
Size (res)	30 MB	20 MB	30 MB	30 MB	20 MB	19 M
Cold Start	1.6 s	5.0 s	2.4 s	1.5 s	2.8 s	0.9 s
Warm Start	1.2 s	4.6 s	2.2 s	1.1 s	2.7 s	0.8 s
UO2-15H2O <sup>(a)</sup>	0.5 s	2.14 s	7.6 s	4.4 s	NA	0.95 s
<b>Ecce 2.0</b>						
Size (res)	25 MB	14 MB	21 MB	25 MB	13 MB	12 MB
Start	1.1 s	1.0 s	1.0 s	0.9 s	2.0 s	0.42 s
UO2-15H2O	0.1 s	0.2 s	0.9 s	2.2 s	NA	0.48 s

(a) This is an example chemical system consisting of a molecule of Uranium Oxide surrounded by 15 water molecules, typical in size of those studied using ECCE.

(UO2-15H2O) for a total of 50 atoms and individual output properties up to 1.8 MB in size. Although enhancing performance was not a primary goal of the project, it was our goal to avoid a significant performance decrease that would compromise usability. As Table 3 shows, the overall performance actually improved—in some cases significantly. While this set of tests is small, we have qualitatively found that applications perform as well or better than their OODBMS-based counterparts overall. The typical workflow processes that a user performs within Ecce did not derive significant benefit from the cache-forward architecture of our OODB. If we do encounter areas of performance concern where a cache makes sense, one could easily be added to the layered client architecture of Figure 2.

Several additional possible optimizations have not been pursued, such as taking advantage of http 1.1 pipelining, making use of multiple simultaneous connections, or bundling requests where class usage patterns involve setting many data members (mapped to metadata on the DAV object) in rapid succession. Note that the test results reported here do not reflect the use of http 1.1 persistent connections. In the current environment, reconnecting each time was significantly faster than making use of persistent connections, an anomaly still under investigation. Overall, these directions, combined with anticipated enhancements in DAV server performance levels, provide a variety of options for substantially improving performance in subsequent Ecce releases.

In terms of deployability, the DAV-enabled Ecce represents a vast improvement. The client and server licensing costs are now zero, assuming use of a no-cost implementation of DAV, such as Apache and mod\_dav. Because DAV allows manipulation of individual objects and properties, the memory and processing requirements are much reduced in comparison to the OODBMS solution. Configuring and running Apache/mod\_dav is significantly simpler than installing an OODBMS. Also, because Ecce can share a DAV server, it is possible to have no server setup at all. This raises the possibility of small academic groups using a departmental DAV server as their data store, or outsourcing the server completely. Although commercial DAV services are aimed more at simple document and file sharing, we have already demonstrated running Ecce against a public DAV server hosted by Xythos [30]. That is, since Xythos' Web File System (WFS) 3.0 product is DAV-compliant, we were able to have the Ecce client use it as a database by simply configuring the client with the URL of a public WFS server maintained by Xythos at their site. For larger installations, the possibility for using multiple servers with standard Web load-balancing and fail-over services (a path not yet explored in detail) promises reliability and

scalability. The level of security can also be tailored to group needs; because DAV inherits the HTTP authentication, authorization, and encryption mechanisms, a variety of options exist. The standard HTTP libraries required to support the various Web security protocols are not yet included in Ecce. When this is done, selecting encryption of communications with the data store becomes a simple matter of Web server configuration. This broad flexibility makes it possible to tailor Ecce to the performance, storage, and management needs of individual groups.

As a testbed, Ecce now provides an unprecedented level of access to its data store, leading to a variety of possibilities. As DAV is an extension of HTTP, Ecce users can run standard Web browsers to “surf” the Ecce database and to view Ecce-generated images, subject to the same access controls applied when accessing the data through Ecce. Existing applets and applications can retrieve and render molecular structures and other data given the HTTP URL for that item within the Ecce data store. Relatively simple cgi scripts or servlets can quickly be developed to provide thin-client access to many of the features currently provided by heavy UNIX/Motif clients. DAV-enabled browsers would provide the additional benefit of allowing users to view all of the data members mapped as DAV properties as they navigate through the data objects.

Developers maintaining and enhancing Ecce have also benefited from the new data architecture. Web and DAV browsers become debugging tools. In-house developers are no longer burdened with a combined application/schema compilation cycle. Third-party developers choose whether to use the Ecce object schema or to develop a mapping of their own objects into DAV using generic XML parsing tools. The latter option will allow electronic notebooks to directly reference and display Ecce data. In addition, the notebooks will have the capability to add additional metadata, such as digital signatures and annotation relationships, to the data without affecting the operation of Ecce. This open data architecture also makes possible feature analysis applications or agents that can independently discover objects in the data store (3D structures, for example), apply feature analysis algorithms, and attach their discoveries to the objects as new properties. For example, an agent could use the molecular geometry, vibrational frequencies, electron distribution and other properties calculated via Ecce to determine thermodynamic properties of the molecule which could then be appended as new DAV properties of the molecule object. Although Ecce currently cannot make use of this additional data, we envision a modification that would allow Ecce, or any PSE, to present such metadata to the user as part of a query interface. This generic mechanism would make

metadata created by new applications immediately available for use in categorizing and selecting data sets within an existing PSE.

These lightweight integration scenarios can provide real benefits to users without system-wide agreement on a common schema. Moreover, the capability to move incrementally and partially towards a common schema in this open architecture is expected to actually promote more semantic integration. Since DAV supports "live" properties that are calculated dynamically, it is possible to imagine generating metadata on-the-fly to support new applications. Using XML stylesheet language translations (XSLT), a DAV server could be extended to translate properties for applications built using different schema. Thus, developers can encode the mapping between their object schemas external to their applications in a dynamically evolvable form. Although this paper has assumed that such mappings will involve data members encoded in metadata, we are investigating similar mechanisms that would allow XML description of the mapping between the (potentially) binary DAV objects. Ultimately, it may be possible to achieve any desired level of data interoperability between applications through the installation of XML mapping descriptions in a common DAV-based data store.

## 5. Conclusions

Full realization of this vision will require significant additional work. As noted earlier, many of the advanced features of DAV, including DAV Searching and Locating (DASL) and DAV Advanced Collections, are still being standardized, while features such as transaction support are not yet addressed. Development tools that simplify extracting metadata from binary data files are also needed, as are mechanisms to dynamically translate between metadata definitions. However, the growing acceptance of XML and DAV should quickly lead to a range of choices in these areas. Once developed, these tools will provide a rich, domain-independent foundation for developing flexible, scalable, evolvable PSEs.

The release of a DAV-based version of Ecce represents a significant advance for current Ecce users and a step toward a more flexible PSE architecture. The development of a new Ecce architecture to use open, metadata-driven repositories based on DAV has provided immediate benefits in terms of flexibility, reduced deployment and maintenance costs, additional security options, and improved data accessibility. Such schema-neutral repositories will be a critical component of next-generation PSE architectures that will enable dynamic collaboration across scientific disciplines and enhance information discovery. Using Ecce as a test bed, the plan is to continue to expand and explore the possibilities

inherent in open data architectures for integrating feature detection, data mining, and other agents, along with notebooks and domain applications. This approach is expected to significantly reduce the barriers to PSE development and evolution while enhancing capabilities and helping make the PSE a basic part of the scientific infrastructure.

## Acknowledgment

The Pacific Northwest National Laboratory is operated by Battelle for the U.S. Department of Energy. This work was supported in part by the U.S. Department of Energy under contract DE-AC06-76RLO 1830.

## References

- [1] S. Gallopoulos, E. Houstis, and J. R. Rice, 1994, "Problem-solving environments for computational Science," pp. 11-23, *IEEE Computational Science and Engineering*, Summer.
- [2] J. R. Rice and R. F. Boisvert, 1996, "From scientific software libraries to problem-solving environments," pp. 44-53, *IEEE Computational Science & Engineering*, Fall.
- [3] Molecular Science Software Suite. [http://www.emsl.pnl.gov:2080/mscf/about/descr\\_ms3.html](http://www.emsl.pnl.gov:2080/mscf/about/descr_ms3.html)
- [4] D. A. Dixon, T. H. Dunning, M. Dupuis, D. F. Feller, D. K. Gracio, R. J. Harrison, J. A. Nichols, and K. L. Schuchardt, 1999, "Computational Chemistry in the Environmental Molecular Sciences Laboratory," Plenum Publications, Book Chapter in "High Performance Computing."
- [5] D. R. Jones, T. L. Keller, K. L. Schuchardt, H. L. Taylor, and D. K. Gracio, "Extensible Computational Chemistry Environment Data Centered Framework for Scientific Research," 1999, Domain-Specific Application Frameworks: Manufacturing, Networking, Distributed Systems, and Software Development, Chapter 24, Vol. Three, No. 0-471-332801.
- [6] M. Atkinson, F. Bancillon, D. DeWitt, K. Dittrich, D. Maier, and S. Zdonik, "The object-oriented database system manifesto." In *Proceedings of the First International Conference on Deductive and Object-Oriented Databases*, pp. 223-40, Kyoto, Japan, December 1989.
- [7] The Committee For Advanced DBMS Function, Third generation database system manifesto, Computer Standards and Interfaces 13 (1991), pages 41-54. North Holland. Also appears in SIGMOD Record 19:3 Sept, 1990.
- [8] External Review Committee Report on the Extensible Computational Chemistry Environment, January 1996.
- [9] M. J. Carey and David J. DeWitt, "Of objects and databases: A decade of turmoil." *Proceedings of the 22nd VLDB Conference*, Mumbai (Bombay), India, 1996.
- [10] PRE - A FRAMEWORK for ENTERPRISE INTEGRATION. R. A. Whiteside, E. J. Friedman-Hill, and R. J. Detry, <http://daytona.ca.sandia.gov/pre/s-docs/Information/HICCS.html>
- [11] RFC 2616 Hypertext Transfer Protocol -- HTTP/1.1, <http://andew2.andew.cmu.edu/rfc/rfc2616.html>
- [12] XML Specification, <http://www.w3.org/TR/REC-xml>

- [13] RFC 2518 HTTP Extensions for Distributed Authoring – WEBDAV, <http://andrew2.andrew.cmu.edu/rfc/rfc2518.html>
- [14] R. T. Fielding, E. J. Whitehead, Jr., K. M. Anderson, G. A. Bolcer, P. Oreizy, and R. N. Taylor, “Web-based development of complex information products, communications of the ACM,” August 1998 (Vol. 41, No. 8), pp. 84-92.
- [15] DAV Searching & Locating – DASL, <http://www.webdav.org/dasl/protocol/draft-dasl-protocol-00.html>
- [16] WebDAV Ordered Collections Protocol, [http://www.ics.uci.edu/pub/\\_ietf/webdav/collection/draft-ietf-webdav-ordering-protocol-02.txt](http://www.ics.uci.edu/pub/_ietf/webdav/collection/draft-ietf-webdav-ordering-protocol-02.txt)
- [17] Goals for Web Versioning, <http://www.webdav.org/deltav/goals/draft-ietf-webdav-version-goals-01.txt>
- [18] P. M.-Rust, H. S. Rzepa, M. Write, and S. Zara, 2000, “A universal approach to web-based chemistry using XML and CML,” *Chem Commun*, pp. 1471-1472.
- [19] Math Markup Language, <http://www.w3.org/TR/REC-MathML/>
- [20] Extensible Scientific Interchange Language, <http://www.cacr.Caltech.edu/SDA/xsil/>
- [21] DBM Comparisons, [http://www.rz.uni-hohenheim.de/anw/prg/perl/nmanual/lib/AnyDBM\\_File.html](http://www.rz.uni-hohenheim.de/anw/prg/perl/nmanual/lib/AnyDBM_File.html)
- [22] DAV Explorer, <http://www.ics.uci.edu/~webdav/>
- [23] WebDAV mod\_dav, [http://www.webdav.org/mod\\_dav/](http://www.webdav.org/mod_dav/)
- [24] Document Object Model (DOM) Level 2 Core Specification, <http://www.w3.org/TR/DOM-Level-2-Core/>
- [25] Simple API for XML, <http://www.megginson.com/SAX>.
- [26] GridFTP: Protocol Extensions to FTP for the Grid. W. Allcock, J. Bester, J. Breshnahan, A. Chervenak, L. Liming, and S. Tuecke. Internet Draft. March 2001. <http://www.gridforum.org>.
- [27] Unified Modeling Language, <http://www.omg.org/uml>
- [28] Protein Data Bank Format, [http://www.rcsb.org/pdb/docs/format/pdbguide2.2/guide2.2\\_frame.html](http://www.rcsb.org/pdb/docs/format/pdbguide2.2/guide2.2_frame.html)
- [29] Global GridForum, <http://www.gridforum.org>
- [30] Xythos, <http://www.xythos.com/>