A Java-based Science Portal for Neutron Scattering Experiments

Sudharshan S. Vazhkudai* James A. Kohl* Jens Schwidder*

ABSTRACT

The Spallation Neutron Source (SNS) is a state-of-the-art neutron scattering facility recently commissioned by the US Department of Energy (DOE). The neutron beam produced at SNS will have an intensity that is an order of magnitude higher than existing facilities worldwide, enabling a significantly better understanding of and exploration into the structure of matter. The SNS is a billion-and-a-half dollar investment supporting research that impacts diverse science domains such as materials, chemistry, engineering, polymers, structural biology, and superconductivity. Thousands of scientists from around the world will annually perform experiments at SNS, ultimately producing petabytes of raw data that must be reduced, curated, analyzed and visualized. The SNS facility is developing a Java-based *one-stop shopping* web portal with access to the broad spectrum of data and computing services that will facilitate scientific discovery by enabling geographically dispersed users to seamlessly access and utilize the SNS facility resources. In this article, we describe the design and implementation of the SNS portal, focusing on several key architectural components, highlighting the diverse usage of Java in a production environment, ranging from enterprise level software composition to remote interactive visualization and integration with high performance distributed computing.

Categories and Subject Descriptors

D.2.11 [**Software**]: Software Engineering Software Architectures

General Terms

Design

Keywords

Neutron Science Portal, Java Web Portal, Service Oriented Architecture

*Computer Science and Mathematics Division, Oak Ridge National Laboratory {vazhkudaiss,kohlja,schwidderj}@ornl.gov

PPPJ 2007, September 5–7, 2007, Lisbon, Portugal.

1. INTRODUCTION

Located at Oak Ridge National Laboratory (ORNL), the Spallation Neutron Source (SNS) [14] is a large-scale leading-edge neutron scattering facility that hopes to fundamentally revolutionize the analysis and characterization of fine-grained atomic/material structure. These advanced new experimental capabilities will drive scientific discovery across a spectrum of domains, including materials, chemistry, engineering, polymers, structural biology, magnetism and superconductivity. Yet there are many challenges to performing state-of-the-art neutron science at this scale, toward which SNS must blaze a trail and break new ground for the neutron science community at large.

The SNS was recently commissioned by the US Department of Energy (DOE), in collaboration with six national laboratories, and will have at least eighteen distinct instruments, each facilitating a certain aspect of neutron scattering experimentation or research. These instruments will produce an order of magnitude larger data than existing facilities. Three SNS instruments are already online and in testing, and are already beginning to attract users, while remaining and additional instruments will continue to come online incrementally over the course of the next several years.

Traditionally, neutron scientists have conducted their experiments/raw data collection *in situ* at various neutron facilities, and then departed with their datasets in hand for analysis back at their home institutions. This was feasible due to the relatively small size of these datasets. However, at SNS, individual data file sizes will range from hundreds of megabytes to several gigabytes and beyond, making the relocation and storage of these large suites of experimental data somewhat costly. Further, because most science teams are collaborative in nature, across multiple geographically distributed institutions, it becomes increasingly difficult for each user to store and analyze a given experiment's data locally at their home site. The sharing of data and results can be prohibitively slow. In conjunction with these practical concerns, ad hoc data management solutions also imply a lack of reliable metadata and provenance information about the processing of an experiment's raw data and subsequent analysis results. A single unified and centralized data storage and management infrastructure is imperative for the full production scale of experiments that will be performed at SNS, to ensure flexible and efficient accessibility and the proper integrity of metadata management and tracking.

Another concern relates to the breadth and diversity of

analysis and reduction software for neutron science. Each new instruments brings its own associated set of data analysis and reduction processes, and requires specially catered and customized software tools, although these tools all tend to build on the same common functional and mathematical foundations. Until recently in the neutron science community, there have not been proliferate and widely accepted standards for analysis, reduction and visualization software, leaving many users to fend for themselves in terms of actually processing the data. The result has been several disparate, custom or ad hoc solutions, with much redundancy and overlap in the existing community tools. This begs the need for an overarching software infrastructure to provide unified access and integration of these many software tools.

Even given a new cohesive and integrated software analysis infrastructure, the majority of users, especially at academic institutions, would often be confined to run analyses just on local desktop computers. Only a handful of user groups can afford the high-performance computing (HPC) infrastructure truly required for timely processing of these massive new datasets. Access to computational resources, such as clusters with large in-core memory and high-throughput mass storage archives, is required to properly and competitively conduct the more sophisticated data analyses, or any associated simulation/modeling of experimental samples.

Based on a series of end-user interviews, in conjunction with community workshops of the Neutron Science Software Initiative (NeSSI) conducted by SNS [14], an extensive set of user requirements has been collected. Neutron science users fall under three main categories: *novice users*, who desire push-button interfaces; *seasoned users*, who will tweak the tools provided and use them in novel ways; and *expert users*, who desire direct shell access to their data workspace, with a rich environment to write their own tools and compose custom workflows. All users desire that the facility maintain, annotate and archive data, with a rich set of readily available Graphical User Interfaces (GUIs) to sift and search through data and browse their workspaces. The users don't care how or where the data is stored, as long as they have the ability to easily and quickly access it, for running any community- or facility-provided analysis tools, including composing custom workflows around them. Users also need the ability to perform complex interactive visualizations of raw and reduced data, with seamless access to an HPC environment for efficient data analysis, reduction and simulation.

SNS is taking an evolutionary step for the neutron science community by providing *one-stop shopping* through the SNS facility web site, with a diverse set of these key required services, including analysis/visualization software, data hosting/sharing and computing/storage infrastructure. To address the aforementioned requirements, SNS is building a comprehensive software solution, with a Java-based web portal and set of crucial back-end services. This portal approach is beneficial because it provides a means for abstracting and managing the complexity of the system, through a single point of entry, and a common framework for defining the various services and front-end user interfaces. Most users, regardless of their sophistication with computers, have some experience browsing the Internet in a web browser, so this general presentation layer is amenable to many. Java itself lends a well-explored collaborative approach to software development, with a variety of available development environments, tools and libraries. SNS can also capitalize on an existing investment in several Java-based neutron science tools, and experience with developing Java.

SNS users require that the diverse and sophisticated set of neutron science services be readily available "any time, anywhere", through the convenience of a standard web browser front-end. Because the user base covers a wide breadth of skills and experience, especially with respect to computational environments, it is crucial that the cost of entry be minimal for the resulting system, with a shallow learning curve and little or no required software installations on the client machines, yet with flexibility/extensibility for more advanced users.

Java is a natural technology on which to base this infrastructure, due to its portability and flexibility in inheriting from and extending various existing standard classes. Java is being used to implement many of the SNS software subsystems and services, as well as the encompassing portal presentation layer. Numerous existing web portals and enterprise software solutions commonly use Java or Java-based frameworks, due to their flexibility, potential for reuse and the ease with which web-based applications can be developed and deployed. However, applications in science and engineering pose new challenges beyond merely serving static or interactive web content.

The obstacles which SNS must surpass include:

- Gigabytes to terabytes of data from instrument data acquisition systems must be archived in a scalable storage system that is conducive to both long-term storage as well as efficient dynamic web-based accesses. These tens of thousands of datasets must be annotated and cataloged for scalable searching and querying.

- As dataset sizes grow, so does the complexity of remote visualization, including conducting interactive 1-D, 2-D and 3-D views driven remotely from large archived data though a web portal. This opens numerous issues in terms of both web client and server-side caching of data and geometry/image delivery to provide practical levels of interactivity.

- Seamless and transparent access must be provided to HPC cyberinfrastructure (massive supercomputers, high-speed optical networks, parallel and networked file systems, and archival mass storage) to enable users to remotely conduct and monitor long-running data analyses in progress.

- Remote execution of monolithic community-provided analysis and reduction tools, replete with their own independent GUI constructs, must be integrated with the internally developed software infrastructure, and displayed through the common web portal.

- A secure authentication and authorization layer must be provided to control access these services and the proprietary experiment data.

In the remainder of this paper, we present details of the design and implementation of SNS software components that address the aforementioned challenges, and discuss some of the many research issues involved in their realization. We illustrate scenarios where the use of Java has been apt in this endeavor, and situations with a genuine need to interface

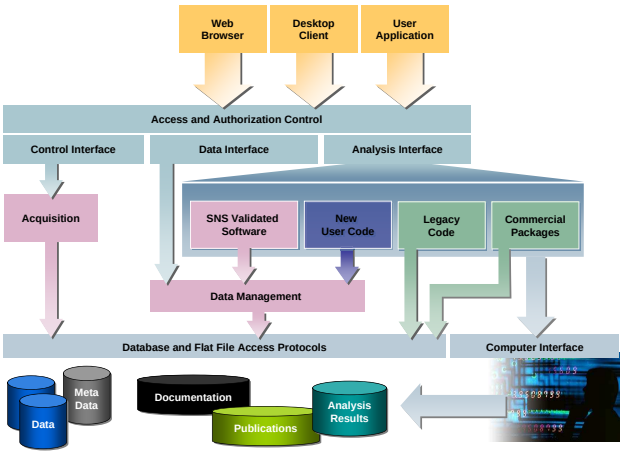

Figure 1: Overall SNS Software Architecture

with native operating system services, e.g. for simultaneous secure access to back-end Java services by *multiple* distinct users. We discuss our modular software build process and the use of Maven to bring together a number of diverse developmental efforts: code from our tightly-coupled internal development team, friendly developer contributions, and open-source third-party and legacy software. Our solutions highlight the diverse usage of Java ranging from enterprise software composition to integration with high-performance distributed computing.

2. SNS SOFTWARE ARCHITECTURE

To set the context for the remainder of the paper, this section briefly overviews the overall software architecture at SNS, as shown in Figure 1. The many functional blocks and systems are organized in a modular layered fashion, allowing a separation of concerns and extensibility/pluggability of various subsystems. This organization is intended to be very flexible, to support the wide range of use cases that are expected, for users with varying degrees of sophistication as well as programmatic access by other external software environments. At the top level, access and use of the SNS software services are made available from remote web clients (the most common usage, and the main focus of this paper), local desktop tools (for *in situ* use during actual experimentation) and other large-scale software frameworks (for post-mortem data acquisition/independent analysis approaches).

All of these access modes must first pass through an Access and Authorization security layer, to validate the identity and permissions of a given end-user to use the underlying software services and access specific experimental data suites. It is crucial to exclude unauthorized access or malicious external attacks, especially for the highly critical experimental control interfaces, but also to protect the high value of the proprietary information contained in as-yet-unpublished experimental results. As such, it is not sufficient merely to identify a specific user as having general access to the SNS software facilities, but the access to specific proposals/experiments and results data must be controlled on a per user basis (see Section 3). All requests/accesses to the internal software services must be properly authenticated/authorized before these operations can be executed, and external access is only allowed at appropriately exported interfaces within the internal service infrastructure.

There are three broad high-level functional interfaces exported through the top-level security layer: *Controls*, *Data* and *Analysis*. The Control interface is primarily intended for internal use for directly monitoring and controlling various aspects of in-progress neutron scattering experiments and samples. The main task of this interface is management of the data acquisition system, to oversee details of the live sample environment, including temperature, position, and the starting/stopping of data collection. The Data interface is a set of simple streamlined mechanisms for searching and directly accessing data files from the permanent data management system archives, including downloading, annotating and uploading/submitting new user-generated files. This interface is independent of the more specific and sophisticated data reduction and analysis functions, that are provided as part of the Analysis interface, which can likewise access data, in whole or in part, and download/upload various data files during data processing.

Most of the complexity in the SNS software architecture resides in the Analysis interface, where a variety of numerical algorithms can be applied in custom workflows, to systematically process (analyze, reduce and visualize) the experimental data. These algorithms operate on data at several levels of complexity, from raw neutron event data to histograms to reduced and energy/frequency domain representations. Similarly, these algorithms and analysis/reduction functions originate from a variety of sources, including internally developed and validated software, legacy/community and commercial monolithic analysis tools, and custom user-provided analysis codes. These different types of analysis functions, across a broad spectrum of software paradigms and runtime environments, must be seamlessly integrated into a unified analysis framework. This allows interchangeable cooperation among functions, in arbitrary custom pipeline combinations, for processing of experimental data. Some of these software libraries and programs can be decomposed or repackaged into individual software components that can be directly integrated and composed into contiguous multi-function analysis applications, potentially on-the-fly. Others codes remain independent monolithic programs that must be individually staged with proper input files, with subsequent results collected for the next stage in the workflow pipeline. Details of this complex "Application Management" task are briefly explained in Section 6.

Underneath the higher level Data and Analysis interfaces, there is a generic Java-based *Data Management* layer for accessing various archival and scratch data files, and a *Computing* layer for low-level handling and monitoring of available computational resources, for executing the requested data analysis pipelines. The base layer database archives are used to organize and coordinate access to all experimental data and analysis results, with the associated meta-data and provenance information, as well as provide user documentation, publications, experimental proposal details and user preferences.

The focus of this paper is primarily on the presentation of these various SNS facility services through a generic web browser "portal" client. This "SNS Portal" environment provides a unified and flexible user environment for coordinating access to these services, as described in Section 3. The majority of the portal environment software (both front-end

client and back-end services) is written in Java.

Java is generally a logical choice, as it enjoys a wide acceptance in science communities as a portable and standardized development language, and a commonly supported delivery mechanism for web-based content. It has been used in the development of other science portals such as the Earth System Grid (ESG) [9]. In addition, many Grid-related portals are based on Java technologies, and projects like Gridsphere [10] have done much to support the Java-based development of portals for Grid communities.

3. PORTAL INFRASTRUCTURE

The "SNS Science Portal" is the front-end of the multi-tiered SNS software architecture. The portal is intended to support thousands of users per year as the facility matures. It provides users with a rich interface to address their software, computing and data needs. In its current form, the SNS Portal is a Java applet and a suite of web services hosted in a Tomcat servlet container as a web application. Tomcat itself runs behind an Apache web server. In addition to Tomcat, several other internal software components and services are hosted in a JBoss application server, running on a separate database machine, to distribute the load and shield external access. The web services expose the functionality of SNS back-end software components through straightforward programmatic interfaces. In the following section, key aspects of the front-end portal software are highlighted.

User Interface: As a science portal, the goal of SNS is to provide its user base with a broad set of interactive functionality and controls, for a variety of data processing and analysis operations. The user interface enables workspace browsing, metadata/database searching, analysis/simulation tool setup, visualization of large datasets, and composition of data processing workflows. A Java Swing-based UI was chosen for the primary portal interface, given the breadth of available widgets and classes, and the need for rapid development and easy/portable deployment. Swing expedites implementation of detailed graphical interfaces, with a great degree of flexibility in UI design and manipulation options.

The portal front-end is written as a Java applet, which must be trusted using digital signatures to allow for loading/saving of files at front-end client computers. Yet SNS is committed to supporting both "lightweight" portlet or forms-based and "thick" applet-scale client interfaces. For instance, to be inclusive of code contributions from the neutron science and other user communities, third party tools are sometimes written as lightweight portlets, adhering to the JSR 168 standard to promote sharing (e.g., with OGCE [13]). Applets have been prototyped to coexist with portlets within a JSR compliant portal framework (such as Gridsphere or Jetspeed). The SNS portal also includes traditional web content serving, for basic facility information, online instrument status, etc. This information is best served as RSS feeds within a portlet.

Workspace Browsing: Much of an SNS user's online experiences are confined to activities within their workspace, and the experimental/analysis data therein. A user can manage numerous files within the workspace, from experiments conducted across many facilities, all viewed via the single unified interface. The SNS portal presents a "file explorer" view of the workspace, as shown in Figure 2. A user might also desire to utilize additional views, such as one organized by instrument usage proposals. The portal remains agnostic as to how data is stored in the SNS archives – the current implementation is as a filesystem. However, the back-end *Workspace* interface has been designed to abstract the low-level data management implementation from the portal. This abstract workspace service interface implements a basic set of file operations and data extraction capabilities.

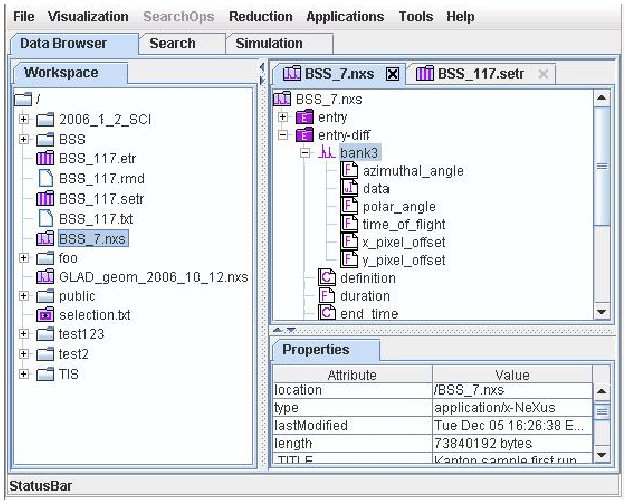

Figure 2: Workspace browsing in the SNS portal

Within the workspace browsing GUI, the viewing of multiple file formats is supported. Each file format is associated with a specific front-end viewer and a cooperative back-end web service for appropriately serving data file contents. For example, a "NeXus" file viewer displays a hierarchical view of the common SNS data file format (discussed in Section 4), while Text and XML viewers simply display the plain text contents of files. A "Properties" panel displays relevant standard file information, including any additional format-specific file metadata.

Security Infrastructure: SNS security infrastructure comprises of authentication followed by an authorization layer. SNS has the rather restrictive requirement that authentication be little more than simple user name and text password-based. This is to minimize complexity and ensure the comfort zone of our user base, as well as provide a scalable solution for the large number of distinct transient users annually (e.g. it is not practical to issue secure electronic id cards or one-time passwords just for a week or two of SNS portal use). To this end, the portal utilizes a user name/password-based authentication system called XCAMS, as generally supported by ORNL for informal external users. XCAMS is connected to an Apache module that provides the portal with information about the authenticated user in the HTTP header of all service requests. The portal security stub pulls this information to instantiate a user principal object that the back-end services can use to identify access permissions. Once authenticated to the portal, users must be authorized to use specific resources, including data, computing infrastructure, databases and—in the future—for externally controlling the instruments themselves.

At the inception of SNS software architecture, the decision was made to use UNIX user accounts and standard filesystem access control lists (ACLs) to meter authorized access to data and compute resources. (This decision was motivated

by the need for interactive shell access by advanced users, in addition to portal-based usage.) The portal must therefore map the authenticated user principal with back-end user accounts to reconcile access to resources. Because the portal server runs inside a Java Virtual Machine (VM), which itself runs as user *tomcat* on the back-end web server machine, the JVM is oblivious to the real UNIX user accounts and cannot negotiate access. Specifically, there is no inherent JVM mechanism for assuming any given user's identity (i.e. "setuid"), let alone for multiple simultaneous users in different threads. Any back-end action on behalf of a portal user must be performed *as that user* to adhere to file/group permissions access control. For instance, users from one proposal group cannot access data from another proposal, and data analyses must be executed as "themselves" in the workspace (this also enables the facility to monitor and track resource usage).

A special native authorization layer interfaces the portal servlets with underlying operating system services, e.g. to negotiate with Data Management to access data or with Application Management to secure access to computing resources (see Section 6). In addition to UNIX accounts, user credential and certificate mechanisms are required by many HPC resources and distributed computing environments. A portal "Simulation" prototype provides authorization through X.509 certificates to execute jobs on clusters (e.g., the "TeraGrid," a national cyberinfrastructure testbed [15]). The following sections further discuss the design and implementation of additional SNS back-end infrastructure.

4. DATA AND METADATA MANAGEMENT

By 2008, data generated at SNS will be on the order of one Terabyte per day contained in over 20,000 files. By 2011, the cumulative data store is expected to reach over one Petabyte. Thousands of users will access the datasets, including downloading, data mining, and conducting real-time analysis, visualization, and sophisticated metadata searches. Data will be accessed through a mix of portal-, web- and shell-based tools and services. The SNS data management architecture must be flexible enough to address this diverse usage, and scale as SNS ramps up its data production.

At the core of this data management architecture is the SNS data repository, which is simply a networked file system. Several alternatives were considered for the data storage organization, such as SRB (Storage Resource Broker) [20] and SRM (Storage Resource Manager) [21], which are used in other science communities. However, the SNS requirement to support both shell and web-portal users precludes the use of these (and other) more sophisticared solutions. Many of these data management solutions cannot be directly mounted as file systems for shell access.

Data is organized by experimental runs, which are stored within project/proposal directories that are nested within each instrument folder. Datasets are stored with basic UNIX file access privileges so that only users belonging to the proposal team can access them. Ultimately, all data files are owned by SNS, and are eventually made world-readable via a directory link in the "public" folder when the proposal becomes public/has been published.

SNS users also have file workspaces at the facility where their proposal and public data is linked in from the main data repository. In addition, users can create scratch analysis data and other personal data in their workspaces, enabling access through the portal from anywhere. The data management system also provides user workspace management services such as creation of user accounts, populating user workspaces with links to accessible data, etc. Many other user workspace administration activities are initiated in response to this SNS proposal submission/creation.

Data on spin storage is archived in HPSS online archival storage [19]. In the long run, all data cannot be maintained on spin storage, so SNS is developing a database catalog system atop both the file system and archival storage. An active "window" of recent datasets will stay on disk file systems, while older data is accessible only from archival storage. These database catalogs and the general data management system provide Java interfaces via portal clients, using remote method invocation and web services. The data management system currently supports transfer protocols such as ftp and GridFTP [17], and hsi (Hierarchical Storage Interface, to access HPSS) [19].

In addition to storing and serving data to clients, the data management system is also responsible for capturing metadata about SNS data. The system applied for this purpose is called ICAT, which uses a common database schema to define neutron science data (adopted by both SNS and ISIS, a neutron facility in the United Kingdom (UK)). The schema comprises attributes covering proposal information, experiment samples, instrument parameters, etc. The ICAT software is written using Enterprise Java Beans (EJB) that interface with an Oracle database at the back-end and present a web service and remote method interface to the portal front-end. The portal UI presents a guided search interface to the user based on the services provided by the metadata management component. Users are initially presented with all the data they are allowed to see. They can then trim this set of results adding further constraints (such as selecting data from a specific principal investigator or experiment, or with specific keywords in the title, etc.)

A note on the fundamental dataset building block is in order here: SNS datasets are NeXus [12] files. NeXus is a neutron-community hierarchical data format standard that describes the dataset in terms of targets, detector counts, pixel banks and x-y offsets. The data management system provides specialized NeXus file format services, e.g. serving file contents as XML to portal clients. To this end, we have adapted an existing Java NeXus [12] library implementation.

Java has proven itself an excellent choice for the data management system, by incorporating existing software from the Neutron Science community, such as JNexus and ICAT. These Java-based tools integrated easily within the SNS infrastructure, along with standard data transfer tools such as ftp and GridFTP, and provided a service layer atop the underlying storage fabric (file and archival system) for web clients. The flexibility and convenience of composing the disparate Java interfaces expedited the development process.

5. REMOTE VISUALIZATION

A variety of existing neutron science visualization and analysis tools are available for use at SNS, including ISAW [11], DAVE [1], IDL [3], Matlab and many others. However, many of these tools are commercial products or contain proprietary interfaces and formats. Because the ISAW system was available as open source and was already written in Java (with the additional benefit of a friendly working relationship between SNS and the ISAW developers), it was a clear

technology choice for the integration of some basic neutron science "views" into the SNS Portal. A custom "subset" of the ISAW classes was extracted and packaged by the ISAW developers for inclusion in the SNS software infrastructure.

The challenge to *using* these handy ISAW view classes was in wrangling and extending the associated internal data management. Like most monolithic tools, ISAW assumes that it can read data arrays into memory as needed from available input files on the local filesystem. Clearly, when running the ISAW viewer classes as part of a remote front-end portal applet for the SNS, this is not logistically possible! Fundamentally, there must be a separation between the local client data processing and the actual back-end data access, which is implemented by replacing the core Java data object classes with remote-capable counterparts. Data must be loaded into the applet's memory by invoking a specialized "data extraction" servlet, via URL from within the Java applet. The servlet runs back on the portal web server machine, which has local (mounted) filesystem access and so can open up the desired data files. The servlet extracts the requested data array (or the desired subregion/elements of it), and then transmits these data back to the front-end applet as the results/output of the servlet invocation.

For the purposes of the SNS Portal ISAW viewers, the size of the multidimensional numerical data arrays that need to be sent to the front-end applet can be quite large, on the order of several hundred megabytes each. Therefore, an efficient custom data delivery protocol was created to directly transmit the numerical values, without the overheads (or benefits) of full Java object serialization. Every byte has an impact in terms of network bandwidth, so eliminating the additional Java state in a full numerical object class can help reduce the latency in delivering these raw data. In a rather primitive but ultimately concise approach, the arrays of numerical values are output in simple ASCII format (rather than the *most* efficient binary format, to guarantee integrity across machines with heterogeneous data formats) and then reassembled with minimal meta-data back into new numerical Java arrays at the client side applet. This stream could be further optimized by compression, however the current implementation does not yet do this. Data arrays are organized into appropriate new applet side data classes, which are then able to implement the interfaces required for driving the ISAW viewer classes. These special applet variations of the ISAW data classes are extended with additional "servlet-friendly" methods that provide meta-data on the origins of the data, and allow a variety of new operations relating to updating the currently loaded array subregions (see below).

Unfortunately, due to the minimalistic client operating environment expected for the diverse spectrum of potential SNS users, this servlet-based data linkage cannot assume a significant amount of available/allocated memory in the Java Virtual Machine (JVM) plug-in being run by the client's web browser. So, aside from having to deal with the significant level of indirection or "distance" (latency) between the data and the applet, this portal infrastructure must also deal with serious remote visualization and data caching issues. Entire data arrays cannot be fully loaded all at one time into the front-end client applet, but must be incrementally or interactively loaded as needed to drive the various viewer classes. This creates problems both in terms of viewer application interfaces and local client data management.

The basic assumption/expectation of the ISAW viewer classes, to have fully in-core data residing in its internal data classes, is no longer valid. This creates many complications in trying to *emulate* an in-core data array at the front-end data classes, by encapsulating a complex front-end data caching scheme. In the worst case scenarios, based on strict data usage and access patterns, some viewers required additional "callback" hooks to allow automatic pre-fetching of specific data elements prior to their use by the viewer. (Fortunately, these special hooks could be added to the main ISAW source tree without significant perturbation.)

In other more moderate cases, incremental data access patterns allowed specific data subregions to be retrieved using a straightforward data caching algorithm, from *inside* the front-end (impostor) data class methods. In all cases, there is a small latency penalty to be paid while the required data elements are downloaded on a "cache miss," which manifests as an occasionally "pause" during what would otherwise be smooth scrolling or animation. However, by choosing an appropriately small "cache block size" this latency can be minimized, in part depending on the available network bandwidth between the client and the back-end server.

These cache miss latencies could be further attenuated by applying some level of "local disk caching" at the front-end client machine. Given the restrictive JVM default of local client memory, relative to the many hundreds of megabytes required to store typical SNS data arrays, a great deal of network bandwidth and latency could be avoided by caching even a moderate amount of data subregions locally on a client filesystem. With the current algorithm, every cache miss triggers a full data replacement in the limited JVM in-core memory space. By storing/retrieving some data *locally*, many data requests could be handled more efficiently, with minimal impact to the user's available disk space. The only additional requirement for this local client caching would be that the portal applet must be signed, however this is already required for other tangential logistical reasons.

Once the above fundamental data delivery capabilities were woven into the original ISAW viewer classes, and these classes integrated into the SNS Portal infrastructure, several additional functional extensions were made to these Java Swing viewer GUIs. In many cases, the original viewers were custom designed for specific types of neutron science data, and as such assumed particular data array dimensionality and axis arrangements. Yet for the SNS Portal's extended application of these data viewers, it is beneficial to be able to *select* the desired data dimensions to view in a 1-D, 2-D or even 3-D viewer, by extracting specific lower-dimensional slabs or slices from data arrays of a higher dimensionality. Specific axes of a data array can be swapped, to provide various orthogonal "probes" or "slices" through a higher-dimensional array.

To support these orthogonal slicing and rotational operations, with appropriate *navigation* and zooming through the source data arrays, the GUIs for each ISAW viewer were extended by retrieving and extending the "built-in" ISAW viewer control classes, thanks to some handy existing accessor methods provided by ISAW. Additional Java sliders and selector elements were applied to "scroll" the selected data region across a higher-dimensional array, and choose the data axis, by label, to be applied for each given viewer plotting axis. For example, when viewing a 3-D data array in the 2-D image viewer, the slider chooses which 2-D slice

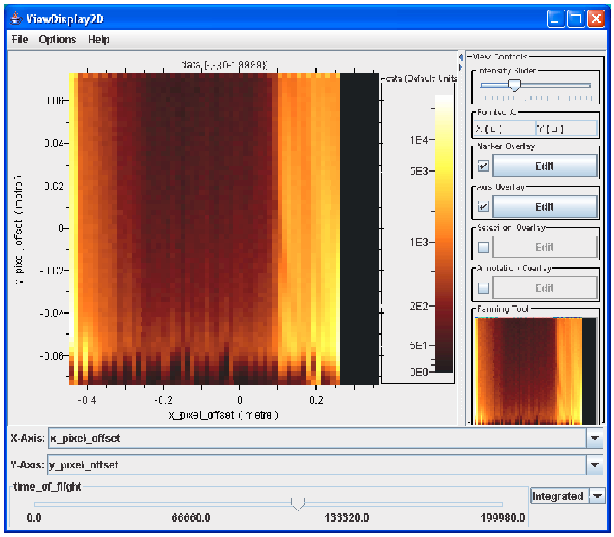

Figure 3: SNS Portal's Extensions to the ISAW 2-D Image Viewer

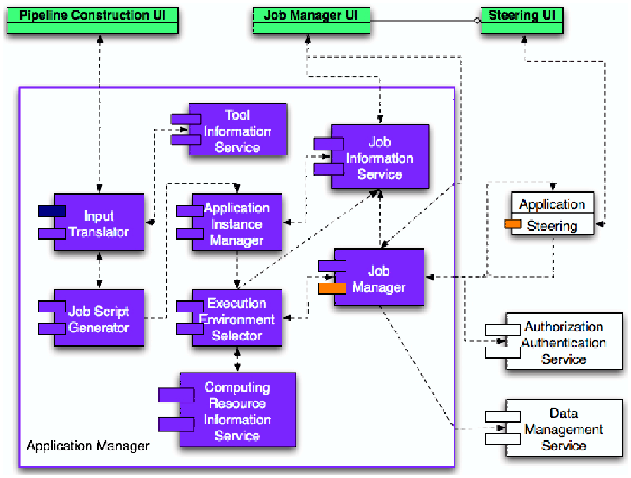

Figure 4: Application Manager

of the larger volume is displayed, as can be seen in Figure 3. Swapping the 2 plotting axes among the 3 available data axes allows all 6 variations/orientations of the image slice to be selected. These successful GUI extensions to the original ISAW viewers warrant integration back into the original ISAW source tree, as part of a future collaboration.

One other significant challenge was encountered in integrating the animated 3-D Volume Viewer into the SNS portal. This viewer uses a Java wrapper library onto the OpenGL standard native graphics library, called "JOGL". Though this native JOGL solution works reasonably well in a local non-applet execution of ISAW, it creates a number of complications for remote applet-based execution. Whereas a simple ISAW install script can place the appropriate JOGL libraries in accessible filesystem locations for local ISAW use, there is not a good standardized solution for on-the-fly JOGL installation or usage within a standard Java applet in a client web browser.

One solution is to manually install the JOGL libraries on each client machine, which is error prone especially for the non-computer-expert user (and violates a fundamental "user friendliness" requirement for the SNS). Other common JOGL solutions involve the use of specialized applet loader classes, which can automatically detect the native client operating environment and install the appropriate JOGL shared object libraries. However, to utilize such a solution would require that the *entire* SNS Portal applet be loaded through this custom applet loader classes. This was not deemed a wise decision from a software reliability and maintenance perspective. It is hoped that upcoming new Java standards will integrate native OpenGL support, and that ultimately this "temporary" snafu will be alleviated.

6. APPLICATION MANAGEMENT

The Application Manager (AM) is responsible for orchestrating the execution of data analysis applications and instrument simulations that are submitted via the SNS Portal. These various computational jobs are constructed in the form of data pipelines that include both approved facility and open community or commercial software packages, each of which can be executed either interactively or in batch mode. The overall AM design is illustrated in Figure 4, which shows the design layout of the internal functional blocks and their relation to other elements of the overall software architecture (as defined in Section 2). Only a brief overview of the many complexities of the AM design will be covered here.

To specify and submit pipelined analysis or simulation jobs, users interact with the Pipeline Construction User Interface (PCUI) to indicate the sequence of tools, each with their own input/output data files, configuration files and control parameters, to describe the desired overall pipeline invocation. A custom tool parameter GUI is automatically generated for each tool in the pipeline, based on meta-data information obtained from the Tool Information Service (TIS) database. An example of such an interface is provided in Figure 5.

The TIS runs as a portal service and provides an XML-based description of all the input parameters for each given tool. This rich meta-data describes the names, command-line invocation syntax, suggested widget elements for the GUI, and tooltip and description information. There can even be complex relationships described among tool parameters, such as required versus optional inputs, and even condition inclusion or exclusion of sets of parameters based on other dependent parameter's settings. The Java applet parses the TIS XML using the Commons-Digester library and produces a "parameter tree" that contains all the parameter meta-data, including the organization of parameters into high-level functional "groups." This tree is traversed to construct the tool GUI, and verify the set of inputs before submission for execution.

The Input Translator (IT), another back-end portal service, is presented with the full pipeline specification, consisting of an ordered list of tools with their associated pipeline connections and input parameter sets. The IT also obtains tool meta-data information from the TIS, on the actual location of these tools on various computational resources including their invocation syntax and preferred execution environment, and resolves specific details for the potential

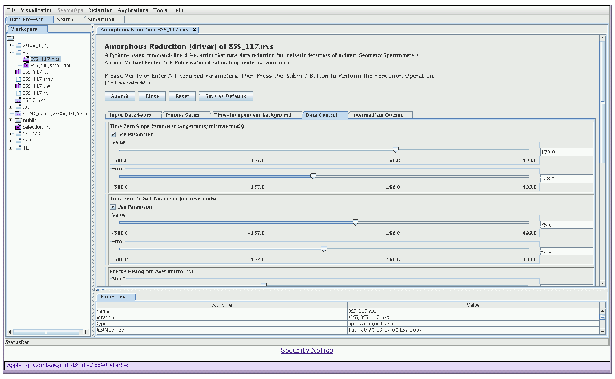

Figure 5: Automatically Generated Tool GUI

execution targets. The Job Script Generator (JSG) massages this information into a set of executable command-line scripts and identifies the data dependencies among them. The scripts and dependencies are passed on to the Application Instance Manager (AIM), which analyzes the data dependencies and appropriately submits scripts with satisfied ("ready to execute") data dependencies for scheduling and execution.

The Execution Environment Selector (EES) is primarily a *scheduler* and takes each "ready" script, along with any preferred execution locations and other data locality requirements, and identifies the best resource for its execution. In its decision making process, the EES makes queries to the Computing Resource Information Service (CRIS) to obtain dynamic information on current resource loading and behavior. The CRIS is implemented as a stand-alone persistent Java database service that collects information from multiple execution environments using the Ganglia cluster monitoring tool [2] and presents it as XML summaries. The CRIS provides its services to the EES via Java RMI.

The Job Manager (JM) then obtains user authentication information for the given job script on the target resource (from queries to the Authorization and Authentication Service discussed in Section 3) and actually schedules the job. The JM handles the mechanics of negotiating access to the resource location, staging the input data files and execution script, and submitting the job to its target resource's scheduling queue. Data files for processing are obtained via interaction with the Data Management Service (see Section 4). Output data is stored back into the data management system, in a scratch area of the user's workspace, for visualization or other exploration of the results. Jobs are monitored using the Job Information Service (JIS) with a job-id and other job related information that can be used to query the status of the job.

The target locations for running SNS analysis and simulation jobs can be any of the available instrument-dedicated analysis computers, internal ORNL institutional clusters, or the national cyberinfrastructure testbed, the TeraGrid [15]. In the current implementation, several execution modes are provided through back-end Java portal services. First is a simple remote shell-based execution through SSH. Second, an execution interface has been implemented around Slurm [4], which is a job execution tool for cluster management that does its own localized scheduling and rudimentary load balancing. Mechanisms have also been developed to allow computation-intensive simulation jobs to be launched from the SNS Portal on the TeraGrid using Globus [18]. If jobs are run internally on ORNL machines, then the output data can be easily captured onto local filesystems, and subsequently accessed on the portal server machine via standard file system mounts. However, if jobs are run on external locations such as TeraGrid, then GridFTP is used to move the resulting data files back into the SNS data management system (or other local scratch space).

6.1 SNS Remote Display Conduit

Many of the SNS analysis and reduction functions, and most instrument simulation jobs, are basic command-line tools or libraries that do not have extensive interactive GUIs. However, some community standard neutron analysis tools and other commercial tools provide (or require) the use of an independent custom GUI panel to operate. Often such tools cannot be separated out and directly integrated into the SNS software infrastructure, such as the ISAW viewers have been (see Section 5).

For these more challenging cases, the most convenient and desirable scenario is that these individual tool GUIs be made available directly to the end-user through the portal. When the user submits a pipeline that contains one of these GUI tools, the appropriate user interface should "magically" appear at the right moment, to let the user control the tool and complete the desired analysis step. This requires much additional infrastructure to coordinate the staging and interactive remote display of the tool GUI to the end-user.

The SNS Remote Display Conduit currently applies a specialized extension of the "WeirdX" system by JCraft, Inc. [5], to allow remote display of X Windows programs through a client's web browser. The SNS extension to this Java-based X server is called "WeeerdX," which integrates directly into the SNS software infrastructure to allow a seamless display of back-end analysis tool GUIs on the user's client machine.

This solution is superior to other remote desktop clients, such as the various incarnations of VNC, because WeeerdX allows the full flexibility of stand-alone "rootless" windows *and subwindows* on the client desktop. Most existing remote desktop clients do not support rootless windows, or if so, do not support rootless popups or subwindows. WeeerdX provides a more natural "look and feel" versus these encapsulated remote desktop clients, improving the seamless usability by the end-user. WeeerdX also communicates with the back-end using the high-level X event stream, rather than channeling raw pixels from the back-end desktop. This provides the potential for a logical and practical compression of the X event stream to help improve the responsiveness of the GUIs.

However, great complexity exists in the WeirdX/WeeerdX solution, in trying to cover and maintain support for the immense collection of potential X server protocols, features and extensions. While the current WeeerdX implementation is moderately complete and functional, there are many unimplemented features that must be added before this system will be considered fully functional.

7. MESSAGING AND LOGGING

Messaging: Many SNS software components send and receive notifications for events. One example is the completion of an experiment run in the data acquisition system (DAS) that triggers the archiving of a dataset in the data manage-

ment system. These events need to be distributed in a scalable and extensible fashion so that future components can produce and receive messages without changes to the existing systems. The Java Messaging Service (JMS) API provides a solution for the above requirements. It specifies an API for common messaging semantics like publish/subscribe (topics) and point-to-point communication (queues). The JMS API has been implemented by a number of different providers.

The SNS messaging system uses JBossMQ, the JMS implementation in JBoss 4.x. JBossMQ serves as the message broker that client components connect to in order to send or receive messages. It was a convenient choice since we were already using JBoss. However, other JMS implementations like ActiveMQ [6] could have also been used. The SNS software components take advantage of the different messaging semantics JMS supports. Most messages are published using topics so that multiple clients can subscribe to events. Some client systems that need to send messages, like the DAS system for instruments, are not written in Java. For non-Java clients, we created a simple web service that allows injecting messages into the JMS based system. The client can use simple HTTP POST requests to submit messages. The message body uses a custom XML format that is used to generate Java SNS message objects. So far, we are not faced with non-Java clients that need to receive messages. However, we anticipate this need in the future.

Logging: The SNS software requires logging for a variety of purposes (debug logs, access and usage logs to meet auditing requirements, etc.). The logging mechanism is also used for runtime monitoring. The SNS software is distributed across many machines, which necessitates distributed logging. The Java components of the SNS Portal use the Log4j API to generate log events. Log4j is a flexible Java API that is extensible and can be configured in various ways. Using Log4j, we are able to direct log events to various targets from local log files to databases. Log information that needs to be retained for auditing—like access logs—are sent to an SQL database. Information can also be sent through JMS, which allows any number of client components to monitor log events within the portal software. Critical log events in some components are also sent out via email to administrators. Basic debug information and log events are written to local files as a backup mechanism in case the distributed logging fails. Further, all of the above can be configured without any changes to the component producing the log events. A central Log Manager, under development, will provide access to logs collected in databases and allow browsing, filtering, and searching through a web-based interface.

8. PROJECT MANAGEMENT

The SNS portal development effort consists of a core development team that is in-house. In addition, there are other external collaborators from universities and non-profit organizations. Since the team is distributed across different sites, it was important for us to find ways in which different aspects of the portal can be developed independently of each other, and in parallel. This is accomplished using a combination of Subversion, Trac [16], Maven [7] and Continuum [8].

We use a Subversion repository (SVN) in combination with a ticket management system, Trac. We have setup several SVN and Trac combinations for the different aspects of the portal to help streamline the notification messages and have individual roadmaps for different Portal components. In order to manage and build our Java projects we decided to use Maven 2. Maven is a project management tool from the Apache Software Foundation based on the concept of a project object model (POM). Our motivation for using Maven was the easy sharing of build libraries through internal Maven repositories. Projects can elegantly define dependencies between each other through the POM. We created Maven repositories for the following purposes: snapshots, releases, and to deploy a number of third party Java libraries that were not available through other Maven repositories. Source code inherited from other Java projects were converted into Maven projects and added to SVN. For all other external dependencies, we included the binary versions in our internal repositories to make them easily available for our development. For each external dependency, we created a POM that documented the source of the dependency and why it was included. Another desirable side effect of using Maven is profile management. For instance, during the development cycle, we are required to build and deploy the portal software on different machine configurations. Maven allows us to configure different profiles with settings specific to our development, test, or production setups. One example is the use of different authentication tools based on the machine the software is deployed on.

We needed a Continuous Integration (CI) system that would allow us to automate basic processes and centralize common operations. This can significantly reduce the learning curve for a developer to setup the project environment on his local development machine. We used Continuum as our CI system due to its closeness to Maven. Each Maven project is added to Continuum for automatic builds every hour. In addition to the basic builds, additional goals have been configured that allow developers to deploy snapshots of artifacts to the internal repository and make them available to the rest of the team. The use of Continuum ensures that the deployed libraries correspond to a specific revision in SVN. The main projects are also configured with goals that allow push button deployments to development, test, and production machines. Further, the use of Continuum for build and deployment also obviates the need for every developer to require direct write access to Maven repositories.

9. CONCLUSION

SNS is a state-of-the-art neutron scattering facility that will revolutionize the way neutron science is conducted. Such a facility requires robust software, catering to diverse user requirements including data access, data analysis, visualization and secure remote access. In this article, we have presented the SNS software architecture and the portal development. SNS portal is evolving as a one-stop shop for all of the above mentioned user needs. We have discussed the design rationale and the Java-based implementation of the internals of the SNS portal. We have shown how Java has served as an excellent candidate for the portal web application, its front-end interface and the back-end abstractions to mask disparate implementations. Further, we have discussed in detail, the design and implementation of the SNS software components such as data management, application management, remote visualization, messaging, logging, authentication and authorization. A sizable portion of the code

in all of the above has been in Java, dealing with the conjoined use of databases and file systems for data management, high performance distributed computing and access to clusters and national testbeds for application management, different authentication modes and communication between these components. Our experience building the science portal, the back-end software components and catering to a large user base supports our choice of Java-based tools for a diverse set of tasks, ranging from software composition to viz caching to distributed computing.

10. ACKNOWLEDGMENT

This work was supported by the U.S. Department of Energy, Office of Science, under contract No. DE-AC05-00OR2275 with UT-Battelle, LLC.

11. REFERENCES

[1] Dave - data analysis and visualization environment. http://www.ncnr.nist.gov/dave/.
[2] Ganglia monitoring system. http://ganglia.sourceforge.net/.
[3] Idl: The data visualization and analysis platform. http://www.ittvis.com/idl/.
[4] Slurm: A highly scalable resource manager. http://www.llnl.gov/linux/slurm/.
[5] Weirdx - pure java x window system server. http://www.jcraft.com/weirdx/.
[6] Apache activemq. http://activemq.apache.org, 2007.
[7] Apache maven. http://maven.apache.org/, 2007.
[8] Apache maven continuum. http://maven.apache.org/continuum/, 2007.
[9] Esg: Earth system grid. http://www.earthsystemgrid.org/, 2007.
[10] Gridsphere portal framework. http://www.gridsphere.org/gridsphere/gridsphere, 2007.
[11] Isaw. http://www.pns.anl.gov/computing/isaw/, 2007.
[12] Nexus. http://www.nexus.anl.gov/, 2007.
[13] Ogce. http://www.ogce.org/index.php, 2007.
[14] Spallation neutron source. http://www.sns.gov, 2007.
[15] Teragrid. http://www.teragrid.org, 2007.
[16] Trac. http://trac.edgewall.org/, 2007.
[17] J. Bester, I. Foster, C. Kesselman, J. Tedesco, and S. Tuecke. GASS: A data movement and access service for wide area computing systems. In *Proceedings of the Sixth Workshop on I/O in Parallel and Distributed Systems*, 1999.
[18] I. Foster and C. Kesselman. The globus project: A status report. In *Proceedings of the IPPS/SPDP 98 Heterogeneous Computing Workshop*, 1998.
[19] M. Gleicher. HSI: Hierarchical storage interface for HPSS. http://www.hpss-collaboration.org/hpss/HSI/.
[20] A. Rajasekar, M. Wan, and R. Moore. MySRB & SRB - components of a data grid. In *Proceedings of the 11 th International Symposium on High Performance Distributed Computing*, 2002.
[21] A. Shoshani, A. Sim, and J. Gu. Storage resource managers: Essential components for the grid. In J. Nabrzyski, J. Schopf, and J. Weglarz, editors, *Grid Resource Management: State of the Art and Future Trends*, 2003.