Tech-X Corporation www.txcorp.com

5621 Arapahoe Ave, Suite A
Boulder, CO 80303
phone: 303-448-0727
fax: 303-448-7756

User-Centric Job Monitoring (UCM)

Final Report

November 6, 2009

Contract: DE-FG02-05ER84170

David A. Alexander, Principle Investigator

alexanda@txcorp.com

303-448-7751

TABLE OF CONTENTS

Section **Page**

I. EXECUTIVE SUMMARY

The User Centric Monitoring (UCM) project was aimed at developing a toolkit that provides the Virtual Organization (VO) with tools to build systems that serve a rich set of intuitive job and application monitoring information to the VO's scientists so that they can be more productive. The tools help collect and serve the status and error information through a Web interface. The proposed UCM toolkit is composed of a set of library functions, a database schema, and a Web portal that will collect and filter available job monitoring information from various resources and present it to users in a user-centric view rather than and administrative-centric point of view.

The goal is to create a set of tools that can be used to augment grid job scheduling systems, meta-schedulers, applications, and script sets in order to provide the UCM information. The system provides various levels of an application programming interface that is useful through out the Grid environment and at the application level for logging messages, which are combined with the other user-centric monitoring information in a abstracted "data store". A planned monitoring portal will also dynamically present the information to users in their web browser in a secure manor, which is also easily integrated into any JSR-compliant portal deployment that a VO might employ. The UCM is meant to be flexible and modular in the ways that it can be adopted to give the VO many choices to build a solution that works for them with special attention to the smaller VOs that do not have the resources to implement home-grown solutions.

II. OVERALL PROJECT ACTIVITIES FOR ENTIRE PERIOD

The UCM project was success in that it resulted in a system that is functionally for the STAR group and represents an example for the Open Science Grid for systems for smaller Virtual Organization (VO) to implement in order to give their users monitoring information. The STAR computing team was heavily involved in the design so that the system would best serve their needs, which was our first priority on the way to a general small-VO solution. Therefore, even though we have shown the generality of the solutions they end up in the reference implementation a tailored solution for the STAR research group.

At the end of the project, we believe that we accomplished the goal of creating a set of tools that can be used to augment grid job scheduling systems, meta-schedulers, applications, and script sets in order to provide the UCM information. The system does provide the intended flexible levels of application programming interfacing that are useful through out the Grid environment and at the application level for logging messages. The user-centric monitoring information in a abstracted in a "data store" and we have provided a very general messaging transfer system to get the monitoring information from the grid sites to a portal server. These features make the system a great start for the smaller VO to build the system that suits their needs.

We have tested the UCM system in practice at BNL and at PDSF at Nersc. This has been invaluable in ironing out some of the implementation details and some usability improvements. We did not quite reach the ultimate goal of getting all STAR scientists using a system that reduces their work flow load, but the system has benefited the data production teams at STAR.

On the technical side, we have produced an end-to-end UCM system where the broker is one host, jobs are submitted to an Open Science Grid site on another host, and finally have messages from the application that composes the jobs send information to the data store located on a yet another host. This has been a successful collaboration with the "Center for Enabling Distributed Petascale Science" project (see http://www.cedps.net) where our work was synergistic with and based-on what they learn from their investigations into the troubleshooting information needs of Grid such as the Open Science Grid. In this collaboration, we have also investigated making links between UCM information and CEDPS information that is collected from the GRAM and local scheduler systems. So for example, in our portal we have shown case studies where had a link from the UCM jobs to the corresponding CEDPS information about that job. In this case study, we ran a grid job at PDSF that is composed of a simple application that sends a few messages to the UCM store which was a database running at one host machine (osp.nersc.gov) that was local at Nersc, but external to the grid cluster (pdsf-grid.nersc.gov). This host also had a CEDPS database, where grid scheduler software (GRAM) and cluster scheduler software (SGE) were collected from jobs run on the grid cluster. Our portal then could connect to both databases and first show the UCM

information and then if the user was interested (for example, if there was a job that failed), then they could click on a "show CEDPS" link to display the information from the CEDPS database that corresponds to that specific job. This exercise forced us to find a common identifier to connect the information in the two databases. We found a convenient identifier to be the ID returned from the GRAM server. This could be collected in the UCM system by the broker and is also in the CEDPS information because it is collected directly from the GRAM server.

The final architecture the UCM system is shown in Figure 1.

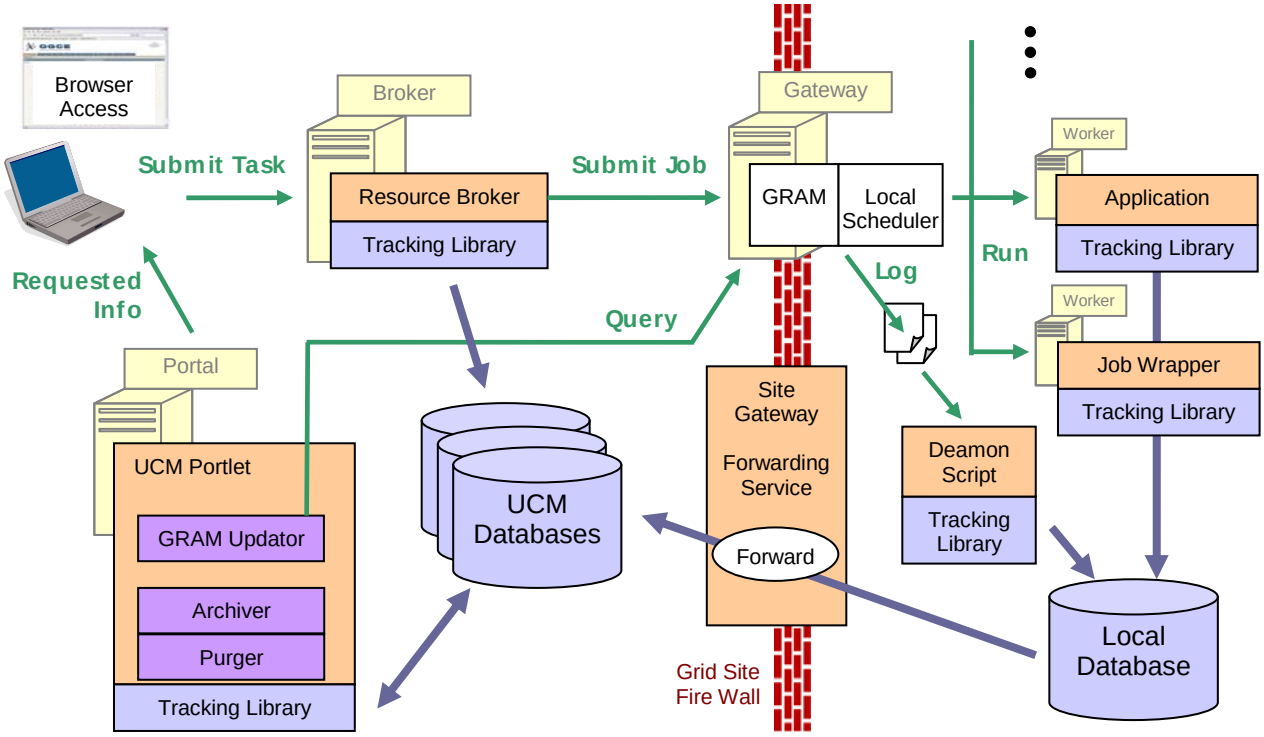

Figure 1. The architectural reference for the design of the UCM toolkit. Our prototype BNL implementation of the UCM architecture will have a central database located at the same site as the application run site, but in a Grid environment we must take into account a "forwarding" mechanism that will collect the information to a centralized database (or federation of databases).

In the target architecture, which is admittedly from the Open Science Grid point of view, there are three main information production points: the broker, the site grid scheduler (or combination Grid & local scheduler), and the application. At the broker point, we envision the broker using the Tracking Library directly to populate task and job

related information to the system. At the site level, there are some options in getting job submission status information from the GRAM scheduler, but we are currently focusing on daemon-style parsing the logging of the GRAM in the latest Globus source that is being written in the CEDPS format. Thirdly, at the application level, the library can be used directly by the application or by a job wrapper or by a pilot job script. Finally, once the data is collected into the store, the idea is to also provide tools to display and view the information from a web portal.

The following sections of this report discuss our progress in two areas: (1) implementation of the Tracking Library including nailing down a minimal interface for collection in the three main points of the architecture; and (2) implementation of the basic Web portlet for accessing the UCM information.

TASK 1. Determine UCM Properties

Objective

Continue prototype work with partners to determine the best collection of UCM information. Determine how it should be structured based on the way it will be accessed, and how and when it is collected. This will include properties from dispatchers, resource management systems, applications, and external sources.

Accomplishments, Results, and Analysis

The work on UCM properties was for the most part finished early on in the project and done in collaboration with our subcontracting partners at STAR/BNL. We have made minor refinements in the taxonomy of UCM properties of interest and this in large part has come recently in the form of indentify two crucial properties that are carried through the Grid system similar to as a traveler would carry a passport to identify themselves. The two properties (broker job ID, and broker task ID) have been dubbed the “magic two” because as monitoring messages are generated by the broker and application the carry this information so that the data store can sort messages to the right user/job/task collections. As a review from our previous report, we have categorized the UCM properties into three categories: user task level, job level, and job event level. Figure 2 shows a list of the prototype properties that we currently using.

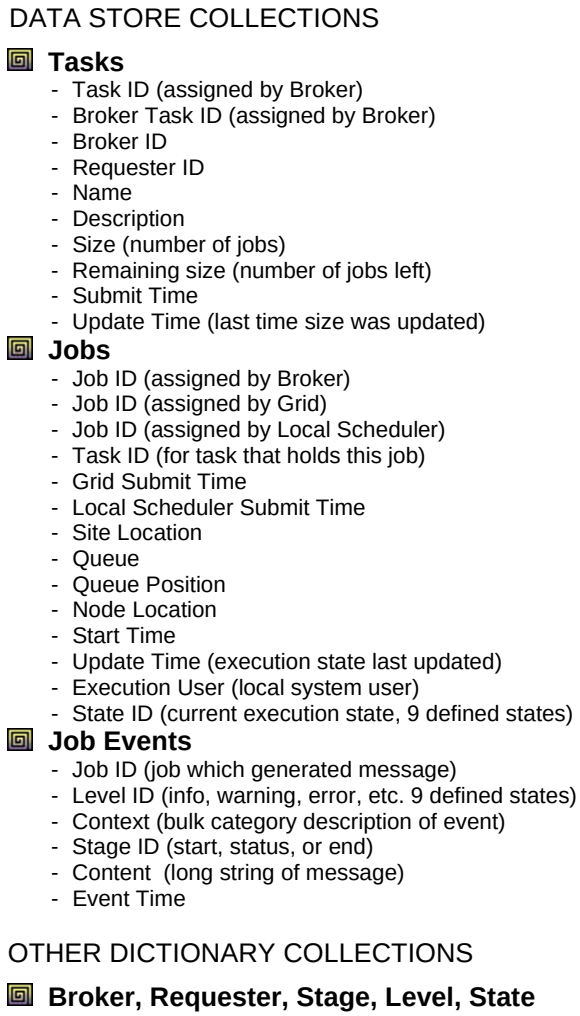

Figure 2. The current UCM properties that have been identified as being important to collect in the data store collections (which correspond to tables in a database configuration). The properties are broken down into three categories and auxiliary categories hold more detailed properties for performance during access.

TASK 2. Design Dispatching Database and Tracking Database Structures

Objective

Create schemas for the Tracking Database(s). Focus is on organizing and archiving the UCM information efficiently. We believe that the best way to proceed is to have one database schema even if more than one database ends up in the final design.

Accomplishments, Results, and Analysis

As reported previously, we have identified database schema. This schema is shown in Figure 3.

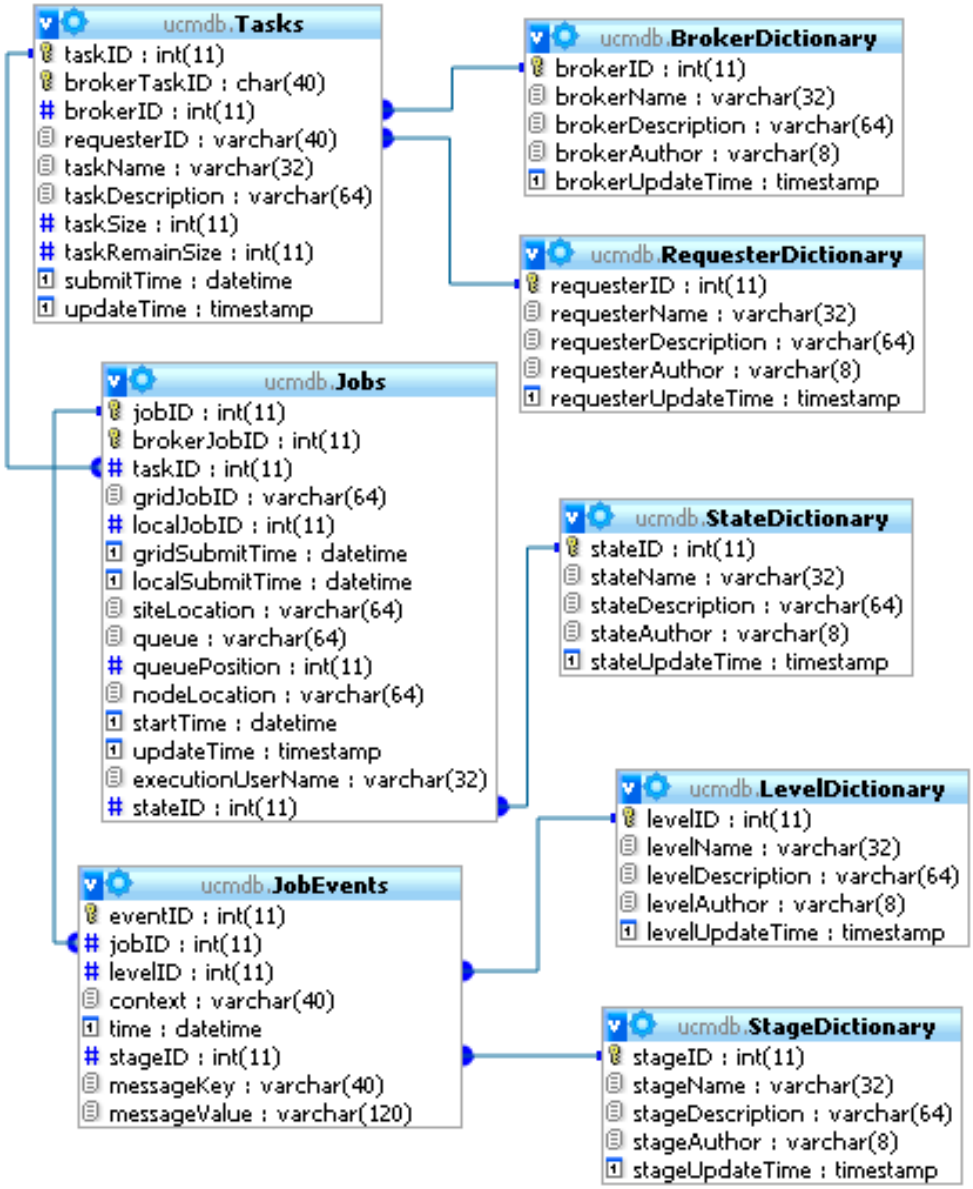

Figure 3. A three level DB schema coinciding with the three-level structure of the UCM properties has been developed. The three main tables are "Tasks", "Jobs", & "JobEvents". Each entry in the Jobs table has a task ID and each entry in the JobEvents table has a job ID. Auxiliary table provide "Dictionary" lookups and allow for integer references to be used in the main tables for performance.

During our last workshop with our BNL collaborators (see https://ice.txcorp.com/trac/ucm/wiki/BoulderWorkshop), we have indentified some of the remaining changes in the schema that were to be done to address the scaling issues that will inevitably be present in a VO-wide implementation of the UCM system. The important table structure changes were implemented which include:

- JobEvents should be changed to Events
- jobID in Events needs to be indexed key
- NEW TABLE NAMES
    - Task
    - Jobs_requesterID_brokerTaskID
    - Events_requesterID_brokerTaskID
        - drop RequesterDictionary? tables
        - keep "requesterID" as varchar, but change to (32)
        - remove requsterID from Tasks_requesterID table

TASK 3. Design UCM Tracking Library API

Objective

Define the Interface UCM Tracking Library. Determine the best ways to write UCM information into databases from applications and meta-scheduler.

Accomplishments, Results, and Analysis

After complete the initial version of the Tracking Library as detailed in the previous report, we started testing it in real-world grid submission. This gave us the opportunity to investigate potential problems with firewalls and scaling. With these tests, we concluded that the original design was too difficult to implement in practice because of the communication required from outside (the portal running at Tech-X) to the inside of the grid site (the database running at osp.nersc.gov). We did set up a secure shell tunnel to make a demo work for the SuperComputing (SC08) conference; however, in general the tunnel is not stable and robust. The redesign discussed in the introduction should solve this problem by limiting the communication to “syslog-ng” messages in an outward bound manner. This does require firewall rules adjustment, but they are on the portal host and not the grid site, so they are logically more in the control of the VO and not as large of a hurdle to overcome. In our testing case, the portal is running at Tech-X and we have already adjusted the firewall to allow these log messages from Nersc.

This redesign has prompted a corresponding refactoring of the Tracking Library. While the API has not changed drastically, the implementation has been broken into two parts so that we can incorporate the syslog-ng logging.

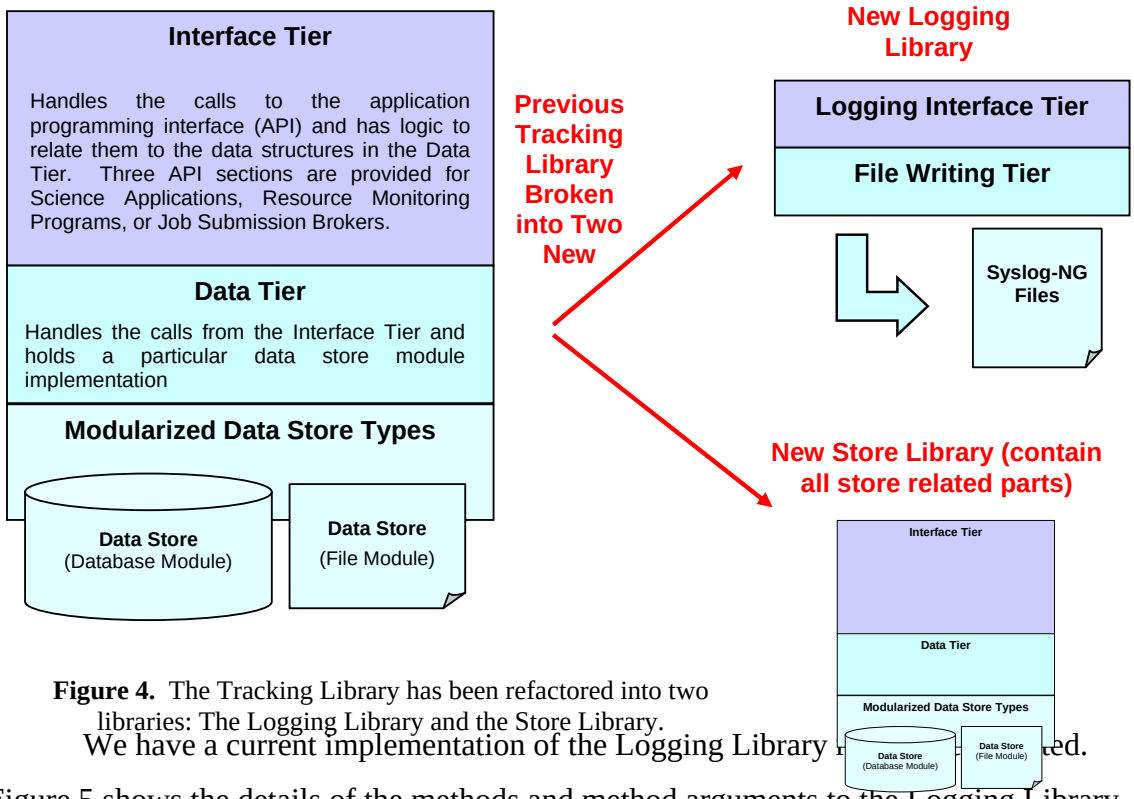

Figure 4. The Tracking Library has been refactored into two libraries: The Logging Library and the Store Library.

We have a current implementation of the Logging Library [illegible]ed. Figure 5 shows the details of the methods and method arguments to the Logging Library API portion. There are basically two important methods: the constructor of a TxLogEvent object and a set of methods to log events called logEvent. This API is intended to be used by both the broker and the application.

Public Types

enum	**Stage** {
	START = 1, **STATUS** = 2, **END** = 3, **START** = 1,
	STATUS = 2, **END** = 3
	}

enum	**Level** {
	LEVEL_TRACE = 1, **LEVEL_DEBUG** = 2, **LEVEL_INFO** = 3, **LEVEL_NOTICE** = 4,
	LEVEL_WARNING = 5, **LEVEL_ERROR** = 6, **LEVEL_CRITICAL** = 7, **LEVEL_ALERT** = 8,
	LEVEL_FATAL = 9, **LEVEL_TRACE** = 1, **LEVEL_DEBUG** = 2, **LEVEL_INFO** = 3,
	LEVEL_NOTICE = 4, **LEVEL_WARNING** = 5, **LEVEL_ERROR** = 6, **LEVEL_CRITICAL** = 7,
	LEVEL_ALERT = 8, **LEVEL_FATAL** = 9
	}

enum	**Stage** {
	START = 1, **STATUS** = 2, **END** = 3, **START** = 1,
	STATUS = 2, **END** = 3
	}

enum	**Level** {
	LEVEL_TRACE = 1, **LEVEL_DEBUG** = 2, **LEVEL_INFO** = 3, **LEVEL_NOTICE** = 4,
	LEVEL_WARNING = 5, **LEVEL_ERROR** = 6, **LEVEL_CRITICAL** = 7, **LEVEL_ALERT** = 8,
	LEVEL_FATAL = 9, **LEVEL_TRACE** = 1, **LEVEL_DEBUG** = 2, **LEVEL_INFO** = 3,
	LEVEL_NOTICE = 4, **LEVEL_WARNING** = 5, **LEVEL_ERROR** = 6, **LEVEL_CRITICAL** = 7,
	LEVEL_ALERT = 8, **LEVEL_FATAL** = 9
	}

Public Member Functions

	TxEventLog (const char *envBrokerJobID=0, const char *envBrokerTaskID=0, const char *defaultContext=0) *Constructors: Connects event log to a store, which events are written to.*
	TxEventLog (const int brokerJobID=-1, const char *brokerTaskID=0, const char *defaultContext=0)
	~TxEventLog () *Destructor: Sets job state to finished, decrements task remaining job count.*
void	**logEvent** (const std::string &eventMsg, **Level** level=LEVEL_INFO, **Stage** stage=STATUS, const std::string &msgContext="") *Log a simple message event.*
void	**logEvent** (const std::string &userKey, const std::string &userValue, **Level** level=LEVEL_INFO, **Stage** stage=STATUS, const std::string &msgContext="") *Log a key-value pair type event.*
	TxEventLog (const char *envStoreInfo=0, const char *envBrokerJobID=0, const char *envLocalUser=0, const char *envGRAMLocalUser=0) *Constructor: Connects event log to a store, which events are written to.*
	~TxEventLog () *Destructor: Sets job state to finished, decrements task remaining job count.*
void	**logSystemEvent** (**Stage** stage, **Level** level, const std::string &systemContext, const std::string &systemMsg) *Log an event not associated with the job.*
void	**logUserEvent** (**Stage** stage, **Level** level, const std::string &userContext, const std::string &userKey, const std::string &userMsg) *Log an event associated with the current jobID.*

Figure 5. The Tracking Library API is broken down into three parts for each of the broker, resource monitor, and application parts of the data production points in the architecture.

TASK 4. Implement the UCM Tracking Library

Objective

Code the Interface UCM Tracking Library. Determine the best way to create and maintain multi-language versions of the library.

Accomplishments, Results, and Analysis

The implementation of Logging Library was completed, so that there is syslog-ng file writing code in addition to the API detailed in Task 3 above. We are now in the process of testing the implementation and will be moving onto the Store Library implementation. The Store Library, re-uses much of what was previous the full Tracking Library, but will has some new methods to process the messages coming from the syslog-

ng pipeline. On the UCM host, we will also wrote a collector that uses the Store Library to pack the information of the messages into the store.

The full Logging plus Store library class list is shown in Figure 6 below.

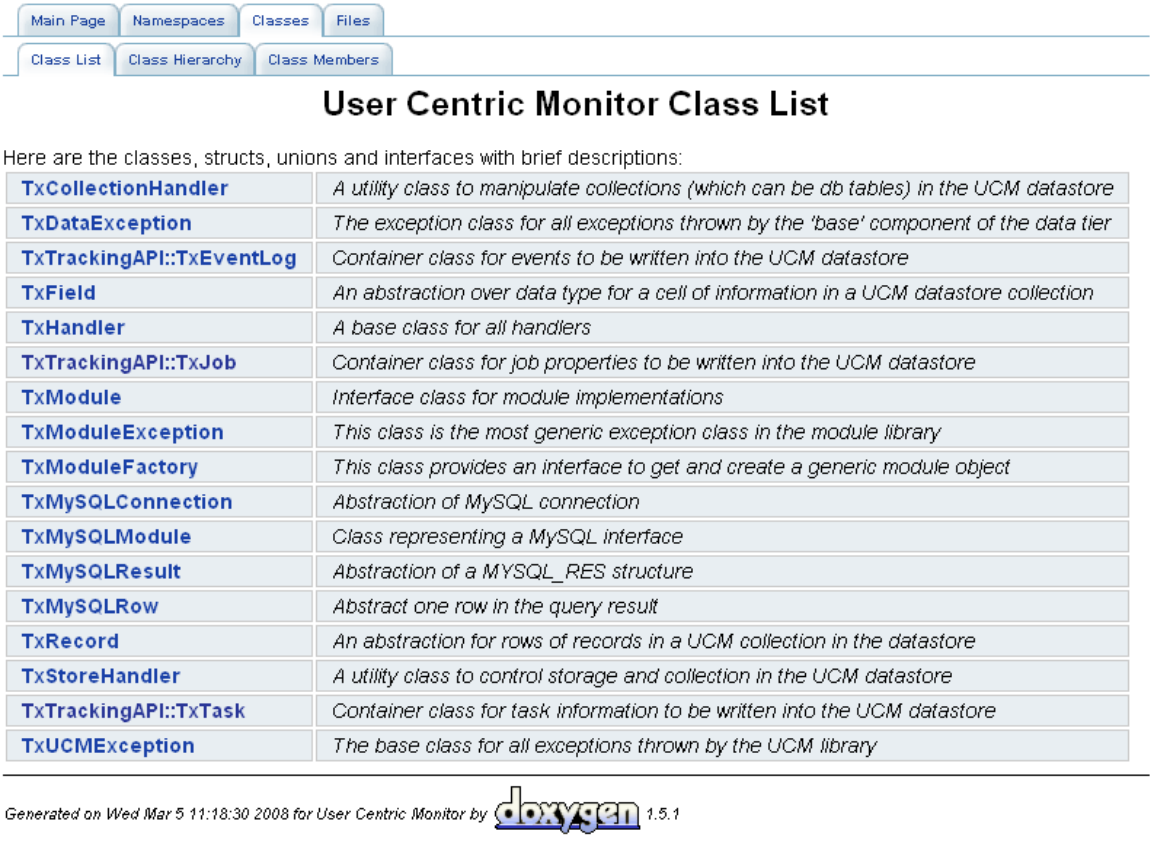

Figure 6. The class list for the UCM Tracking Library includes the *API Tier* (TxTrackingAPI namespace classes like TxTask, TxJob, & TxEventLog) along with other *Data Tier* classes (TxCollectionHandler, TxRecord, & TxField).

TASK 5. Apply UCM Tracking Library in Reference Implementation

Objective

Apply the UCM tracking library in root4star and the SUMS system to log UCM data.

Accomplishments, Results, and Analysis

To be able to use the UCM Tracking Library implementation within the STAR-BNL community the original implementation was adjusted. First, the C++ class

TxEventLog was converted to a pure abstract interface (i.e. all of its methods were made purely virtual, see: Figure 7)

Public Types

```
enum   Stage { START = 1, STATUS = 2, END = 3 }
enum   Level {
         LEVEL_UNKNOWN = 0, LEVEL_TRACE = 1, LEVEL_DEBUG = 2, LEVEL_INFO = 3,
         LEVEL_NOTICE = 4, LEVEL_WARNING = 5, LEVEL_ERROR = 6, LEVEL_CRITICAL = 7,
         LEVEL_ALERT = 8, LEVEL_FATAL = 9
       }
enum   State {
         UNKNOWN = 0, UNSUBMITTED = 1, STAGEIN = 2, PENDING = 3,
         ACTIVE = 4, SUSPENDED = 5, STAGEOUT = 6, CLEANUP = 7,
         DONE = 8, FAILED = 9
       }
```

Public Member Functions

```
virtual          ~TxEventLog ()
virtual void     setEnvBrokerTaskID (const std::string &envBrokerTaskID)=0
     void        setEnvBrokerTaskID (const char *envBrokerTaskID)
virtual void     setEnvBrokerJobID (const std::string &envBrokerJobID)=0
     void        setEnvBrokerJobID (const char *envBrokerJobID)
virtual void     setBrokerTaskID (const std::string &brokerTaskID)=0
     void        setBrokerTaskID (const char *brokerTaskID)
virtual void     setBrokerJobID (int brokerJobID)=0
virtual void     setRequesterName (const std::string &requester)=0
     void        setRequesterName (const char *requester)
virtual void     setContext (const std::string &context)=0
     void        setContext (const char *context)
virtual void     logStart (const std::string &key, const std::string &value)=0
     void        logStart (const char *key=TxUCMConstants::appStart, const char *value="application started")
virtual void     logJobSubmitLocation (const std::string &url)=0
     void        logJobSubmitLocation (const char *url)
virtual void     setJobSubmitLocation (const std::string &url)=0
     void        setJobSubmitLocation (const char *url)
virtual void     logTask (unsigned int size=1)=0
virtual void     logTask (const std::string &taskAttributes)=0
     void        logTask (const char *taskAttributes)
virtual void     logJobSubmitState (State state)=0
virtual void     setJobSubmitState (State state)=0
virtual void     logJobSubmitID (const std::string &ID)=0
     void        logJobSubmitID (const char *ID)
virtual void     setJobSubmitID (const std::string &ID)=0
     void        setJobSubmitID (const char *ID)
virtual void     logEvent (const std::string &logMsg, Level level=LEVEL_INFO, Stage stage=STATUS, const
                 std::string &msgContext=TxUCMConstants::defaultContext)=0
     void        logEvent (const char *logMsg, Level level=LEVEL_INFO, Stage stage=STATUS, const char
                 *msgContext=TxUCMConstants::defaultContext)
virtual void     logEvent (const std::string &userKey, const std::string &userValue, Level level=LEVEL_INFO, Stage
                 stage=STATUS, const std::string &msgContext=TxUCMConstants::defaultContext)=0
     void        logEvent (const char *userKey, const char *userValue, Level level=LEVEL_INFO, Stage
                 stage=STATUS, const char *msgContext=TxUCMConstants::defaultContext)
virtual void     logEnd (const std::string &key, const std::string &value)=0
     void        logEnd (const char *key=TxUCMConstants::appEnd, const char *value="application ended")
```

Figure 7. TxEventLog C++ abstract interface

Secondly, the implementations of the former concrete methods of the class TxEventLog were moved to the new class TxEventLogFile that is derived from the TxEventLog interface. (see: Figure 8)

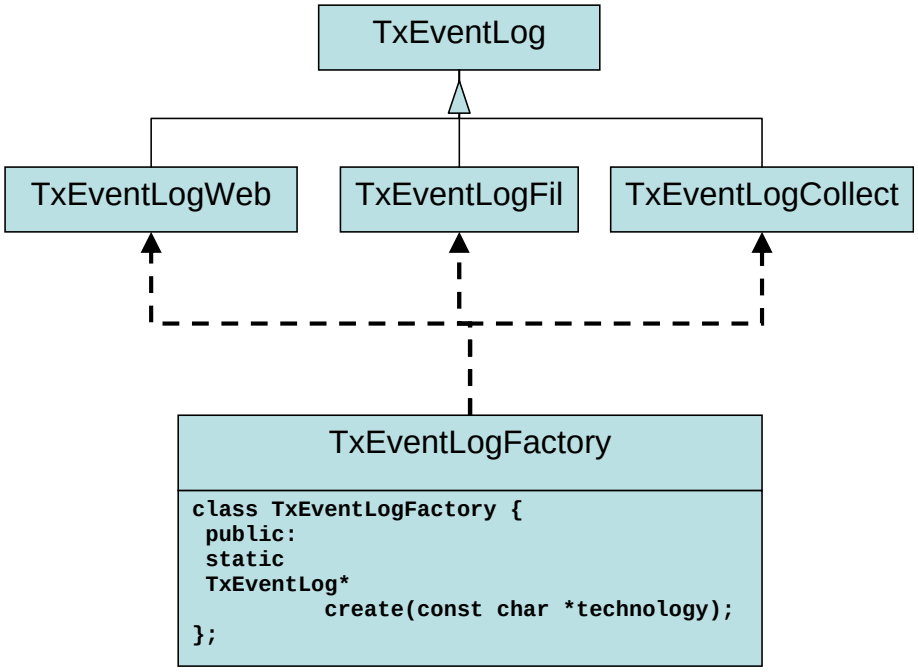

Figure 8. The C++ class diagram: TxEventLog C++ interface and its implementations via TxEventLogFactory

Finally, to provide the concrete implementations for the different run-time environments the UCM library were complemented by the TxEventLogFactory class.

This allows to adjust the job tracking facility to the job current run-time environment, for example, GRID mode, cloud node, local UNIX farm, remote Unix farm etc. (see: Figure 9). Since the class name exposed to the end-user code (TxEventLog) and its interface remained intact, no correction of the existing STAR offline code was required to deploy the new version of the UCM library.

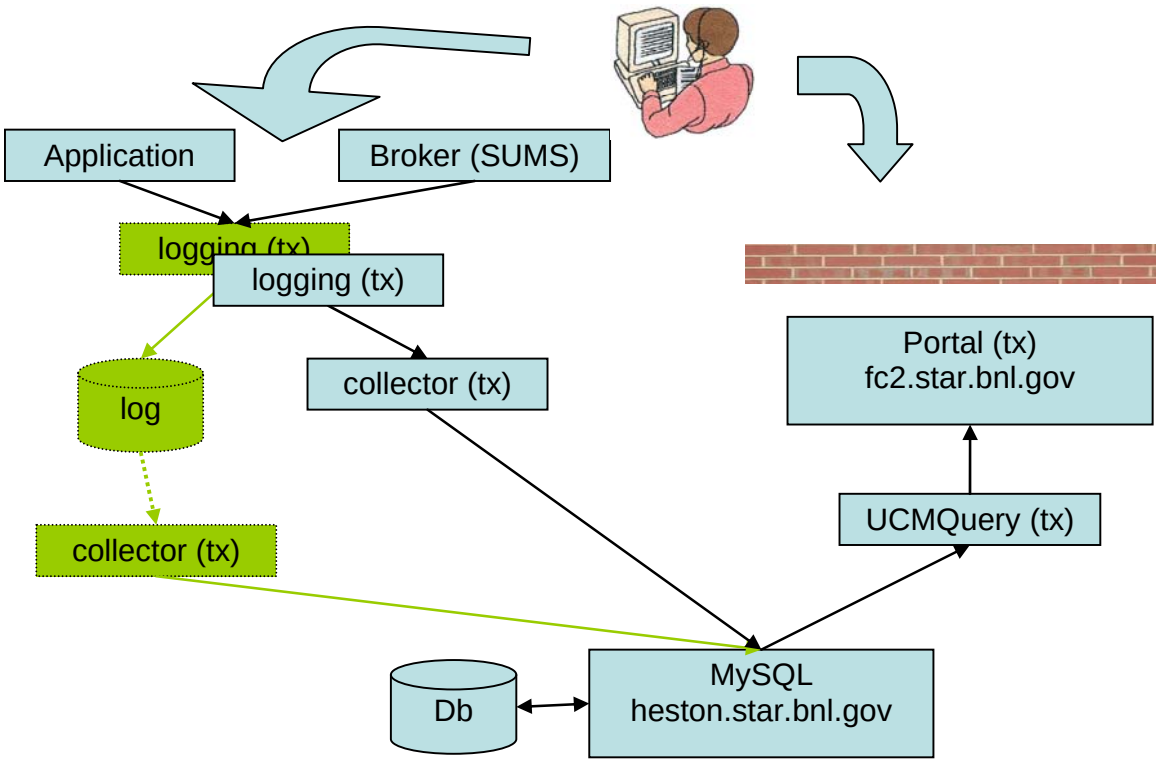

Figure 9. Interaction of the different components of UCM library within STAR run-time environment (green color shows another implementation of TxEventLog interface) (tx) is to mark the original developed code.

To be able to use the UCM C++ library from java-based code, the final implementation stayed away from the initial java wrapper approach (based on Babel) and reverted to a much lighter approach SWIG based wrapper . The C++ TxEventLog abstract interface was converted automatically to the Java code using the SWIG converter. To accomplish this automation, a simple SWIG interface descriptor (see Figure 10 & 11) was created and a simple make file was used (see Figure 12). This allowed the user jobs and SUMS to share one and the same C++ implementation via the common shared library. The initial Babel wrapping was composed of a 3x20 MB shared libraries and a 6 MB JAR file (which exceeded by far the size of the SUMS package as a whole) while the size of the SWIG wrapper is not significant. Since the SWIG approach was automated and made part of the STAR library build, all platform dependencies for the shared library portion are taken care off during STAR software package deployment.

```
/* logging.i */
%module logging
%{
#include "TxEventLog.h"
#include "TxEventLogFactory.h"
%}

%include "TxEventLog.h"
%include "TxEventLogFactory.h"

%pragma(java) jniclasscode=%{
  static {
    try {
        System.loadLibrary("logging");
    } catch (UnsatisfiedLinkError e) {
      System.err.println
            ("Native code library failed to load. \n" + e);
      System.exit(1);
     }
  }
%}
```

Figure 10. SWIG definition of the TxEventLog C++ interface

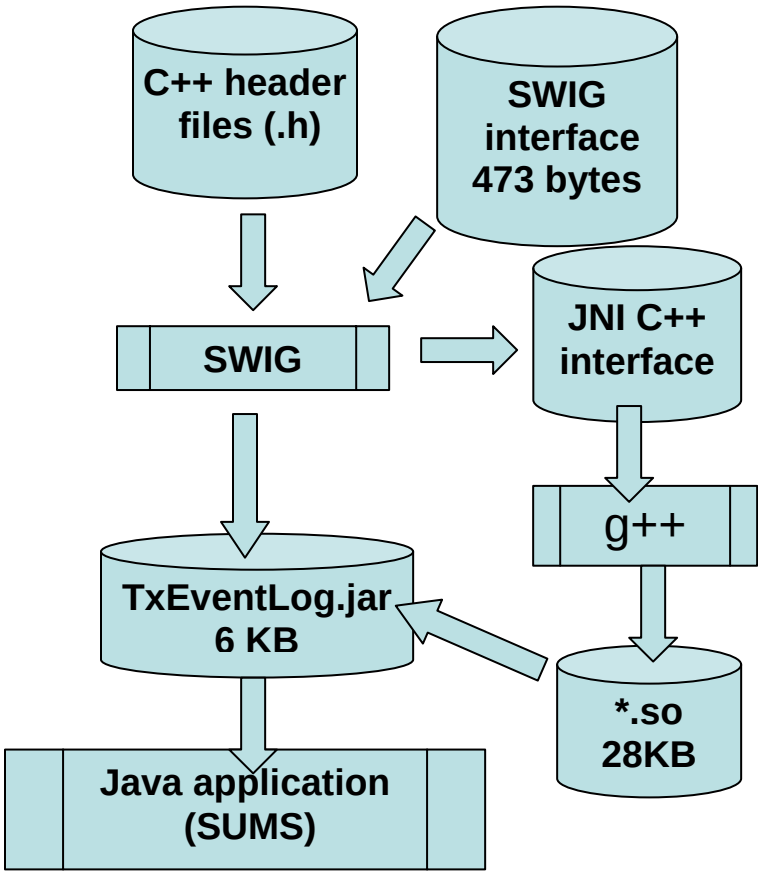

Figure 11. Using SWIG to generate the Java UCM logging interface from the C++ header file (time to complete 0.25 sec)

```
.SUFFIXES: .cxx

INCDIRS   = . StRoot/StStarLogger/logging $(OPTSTAR)/include

CFLAGS    = -Wall -c  $(INCDIRS:%=-I%)
CXXFLAGS  = $(CFLAGS)

MYMOD = logging
MYJAR = tx-ucm

UCMINC = $(STAR_USM_MODULE_DIR)/TxEventLog.h
STAR_USM_BASE_DIR = StRoot/StStarLogger/logging

SRCCPPS +=TxEventLog.cpp  TxEventLogFactory.cxx
SRCS += $(SRCCPPS:%=$(STAR_USM_BASE_DIR)/%)

INCS = $(STAR_USM_BASE_DIR)/TxEventLogFactory.h (STAR_USM_BASE_DIR)/TxEventLog.h
INCSI = $(STAR_USM_BASE_DIR)/TxEventLogFactory.h $(STAR_USM_BASE_DIR)/TxEventLog.h

SWIGI = $(STAR_USM_BASE_DIR)/$(MYMOD).i

OBJS  = $(filter-out %.cxx,$(SRCS:%.cpp=%.o))
OBJS += $(filter-out %.cpp,$(SRCS:%.cxx=%.o))

JAVAS= $(MYMOD).java
JDEPS= $(MYMOD)_java_wrap.cxx
JOBJS= $(MYMOD)_java_wrap.o  $(JSRCS:%.cxx=%.o)

all:  $(MYJAR).jar

$(MYJAR).jar: lib$(MYMOD)_java.$(SOEXT)
      javac -d ./ *.java
      jar -cf $@ com

$(MYMOD)_java_wrap.cxx: $(SWIGI) $(INCSI)
      swig $(INCDIRS:%=-I%) -package "com.txLogging" -java -c++ -o $@ $<

$(MYMOD)_java_wrap.o: $(JDEPS)
      $(CC)  -c $(INCDIRS:%=-I%)   $(JINC) $<

$(MYMOD)_main.class:  $(JAVAS) $(UCMINC:%.h=%.java)
      $(JAVAC) $?
```

Figure 12. Fragment of Makefile to use SWIG to generate the Java UCM logging interface from the C++ header file.

The SUMS job description language was complemented with one additional option to allow users to activate / de-activate the tracking library and require its concrete implementation at run-time.

The dedicated log4cxx-based UCM logger was added to the STAR run-time environment as the top level user interface. To send the custom message to the UCM tracking DB, the end-user code should forward the message to the UCM logger (for example with LOG_UCM macro). The STAR logger documentation linked from http://drupal.star.bnl.gov/STAR/comp/ was adjusted to reflect this fact.

All UCM tracking components were committed to the STAR CVS repository and deployed as the standard components of the STAR production environment. In the other words it is available for all STAR users. All nightly tests do use the new DB schema with the dedicated tables for each job/task. Scalability testing beyond data production (user jobs especially) was not carried out at the time of this report. However, the STAR UCM job tracking infrastructure has been exercised for a period of 3 months un-interrupted, i.e. there was no interruption of service. During this period, the MySQL "logger" DB has accumulated about 40.000 tracking tables with the total volume of 4Gb. Disruption of the neither the DB server nor the job processing has been recorded.

The Tomcat based portal was installed on the dedicated fc2.star.bnl.gov PC and tested. However, the service was shutdown by Cyber Security due to insecure version of the Tomcat portal provided as a part of UCM portal distribution. The portal could be made to be compliant with security standard by an upgrade in the Tomcat container version, which would need to be tested with the GridSphere code. We anticipate that this would be straightforward and leave it up to partners at BNL to do.

TASK 6. Develop Site Tracking Agent

Objective

Develop an agent to serve information from the Tracking Database that is populated by the scripts and applications running at the job execution site where the Site Tracking Agent is located. This agent will run on any OSG site as an edge service. This agent could parse scheduler logs for job status information if there is GRAM servers do not provide such information properly.

Accomplishments, Results, and Analysis

After many discussions and research in solutions to funnel application tracking information out of a grid site and into a VO-centric store, we have reached some basic conclusions about the appropriate technology and that has been to move towards syslog-ng messaging. This also integrates our project well with the CEDPS troubleshooting project and makes going back and forth between the types of information and connecting the two sources will be easy and straightforward, which is valuable.

In this light, the basic architecture has shifted. Instead of writing our own "Site Agent" we settled on using syslog-ng to act as the forwarding agent. Figure 7 shows the message pipe line between the broker host submitting the job or the grid site running the jobs and the final message destination host, which we call the portal or UCM host since it holds the UCM data store and portal.

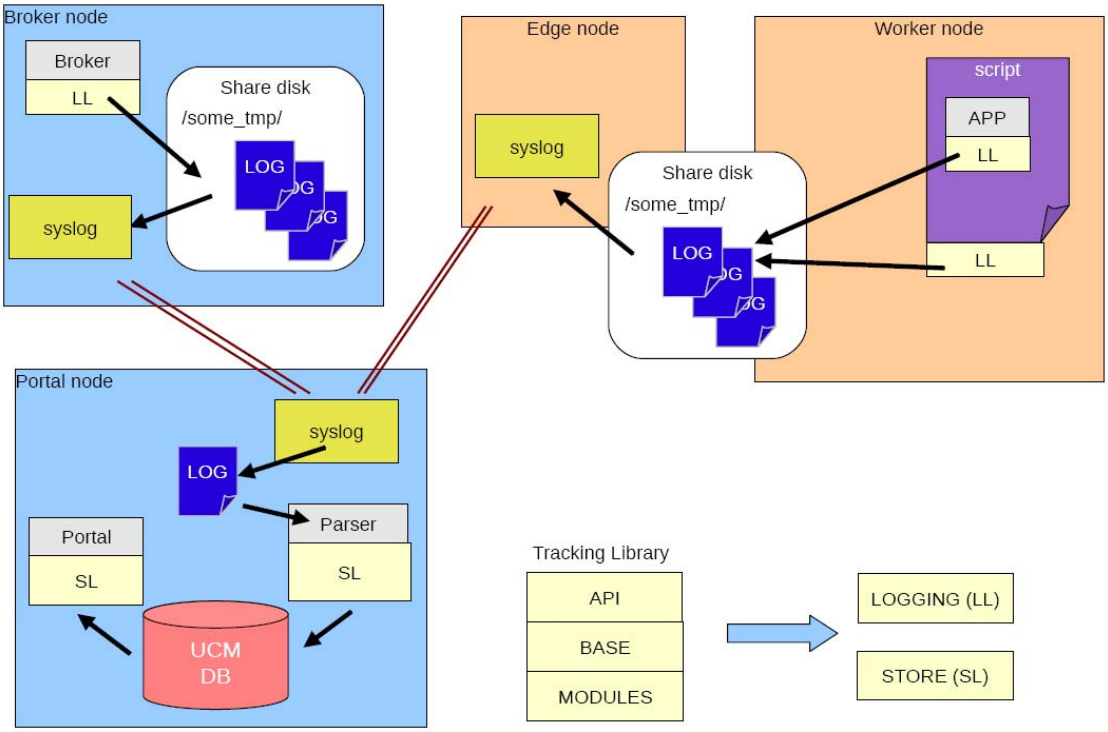

Figure 7. The new architecture using syslog-ng as a message forwarder from an "edge node" that shares disk space with the grid site worker nodes.

TASK 7. Define UCM Service Interface

Objective

To determine the best interface for client applications to get monitoring information and then develop it.

Accomplishments, Results, and Analysis

One of the original interpretations of security was that a Grid Service would be needed to serve the UCM information. However, after the bulk of the Phase II

investigations, we have found that security can be handled by the portal and that a Grid Service is not needed. We have therefore put much of the effort that was budgeted for Task 7 into the portal work described in Task 9. Also, security is described in a recent paper on our portal given at the Grid Computing Environment Workshop at SC08 (see paper included in appendix).

TASK 8. Construct UCM Service

Objective

Extend prototype to implement UCM Access Interface defined in previous task. The UCM Service will access the Dispatching Database and Site Tracking Agent as well as external services to collect UCM data.

Accomplishments, Results, and Analysis

As was written above in Task 7, a Grid Service is not needed to serve UCM data to the users and in terms of getting information from the Site Tracking Agent, we now have in the design syslog-ng so the messages are received on the portal/UCM host by a syslog-ng receiver, which is software we do not need to write. In the last period of performance, we wrote a "collector" or "parser" to read the message on the UCM host and pack them into the data store. These were tested by submitting jobs at NERSC and collecting the monitoring information on the portal located at Tech-X. This worked as planned and the portal was also able to get CEPDS troubleshooting information from the NERSC database as well.

TASK 9. Develop UCM Portal for Reference Implementation

Objective

Develop client library in Java (e.g. base class) for accessing the UCM Service. The library later can be used by various client applications (e.g. Web portal and command line tools) to retrieve UCM information. Implement JSR168 compliant portlet based on GridSphere for UCM information presentation. Construct UCM Web portal for the reference implementation.

Accomplishments, Results, and Analysis

Efforts originally planned for Task 7 were applied to the portal and the increased effort resulted in a very well implemented portal that exceeded our expectations and what we promised in the proposal for the project. The result is that we implemented a fully functional Web Portlet within the GridSphere Portal framework. We have published a paper detailing the work that address this task in a portal specific workshop at the last SC08 (Super Computing) Conference. A copy of the paper is attached in the appendix of this report.

One of the most important hurdles accomplished was to add a security mechanism to the portal that would work with DOE certificates. Figure 8 shows how this is displayed to the user and the details of the information are in the paper in the appendix.

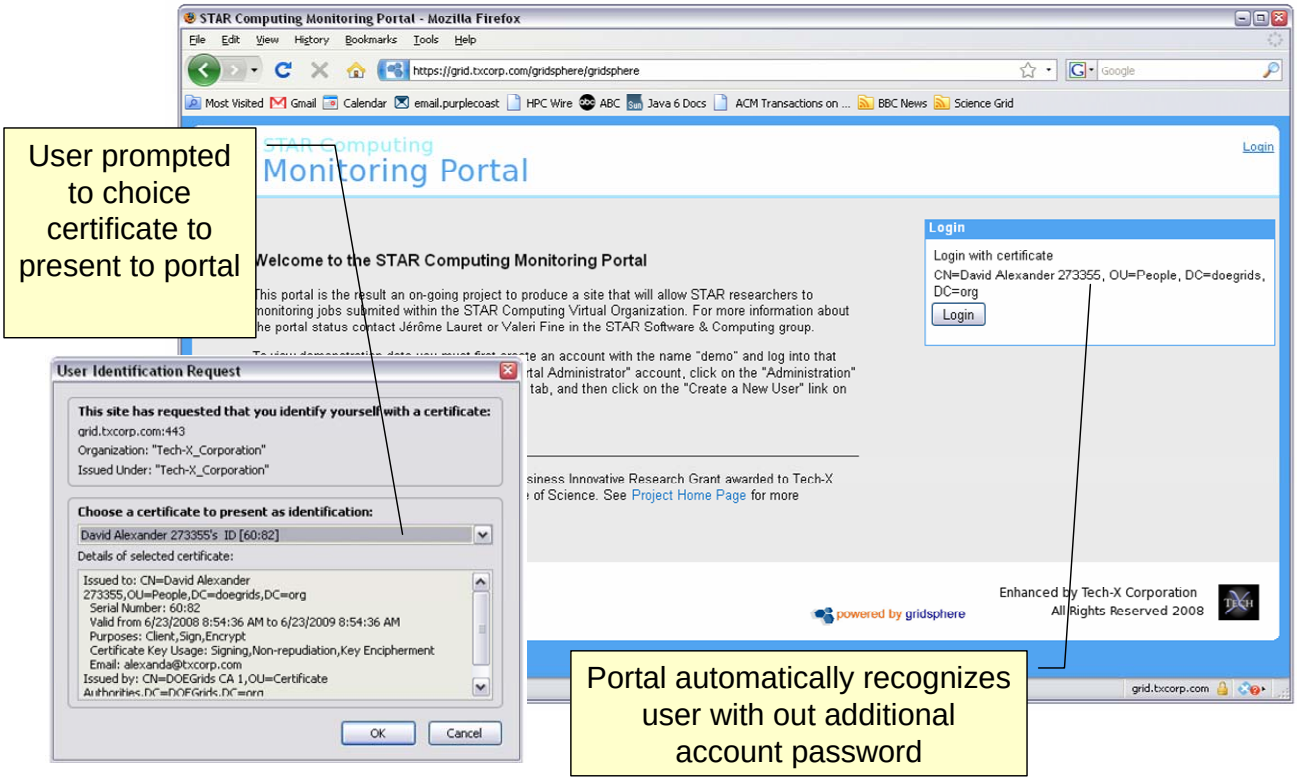

Figure 8. New certificate-based security added to the portlet. Login mechanisms gather DOE certificate from the browser and uses it to connect to portal, so that the portal can subsequently use the certificate subject information to identify user with the virtual organization.

The bulk of the remaining work for Task 9 can be divided into two basic parts of code: a database reader and a combination of JavaScript and Java Server Page code that displays the UCM information. Further work was necessary to integrate the resulting Portlet into a test Portal that we built so that we could test it in a full Portal environment where a user would get a proxy and submit a job to a Grid site.

Figures 9-12 show the current state of the UCM information service Portlet. The main view is a list type view with lines at the top level being tasks. Each line with a "+" or "-" symbol on the left is clickable to expand and collapse the level of information presented. So task lines can be expanded to see the contained jobs and each job line can be expanded to see the contained messages. In this manor the user can drill down into the information as needed.

A few other features are important to note. One is that only the tasks/jobs/events that are identified as coming from the user are shown to begin with. This greatly simplifies the organization of the information for the user and reduces the complexity of the noise that they have to search through to get the information that they need. Another is that tasks lines have overall progress bars. This is a built-in functionality in where the "progress" is simply the number of the jobs that are not done over the total number of jobs defined in the task. Yet another feature is that each line along the thread that contains a job that has an error status is color coded red so that at a glance the user can determine the gross failure status.

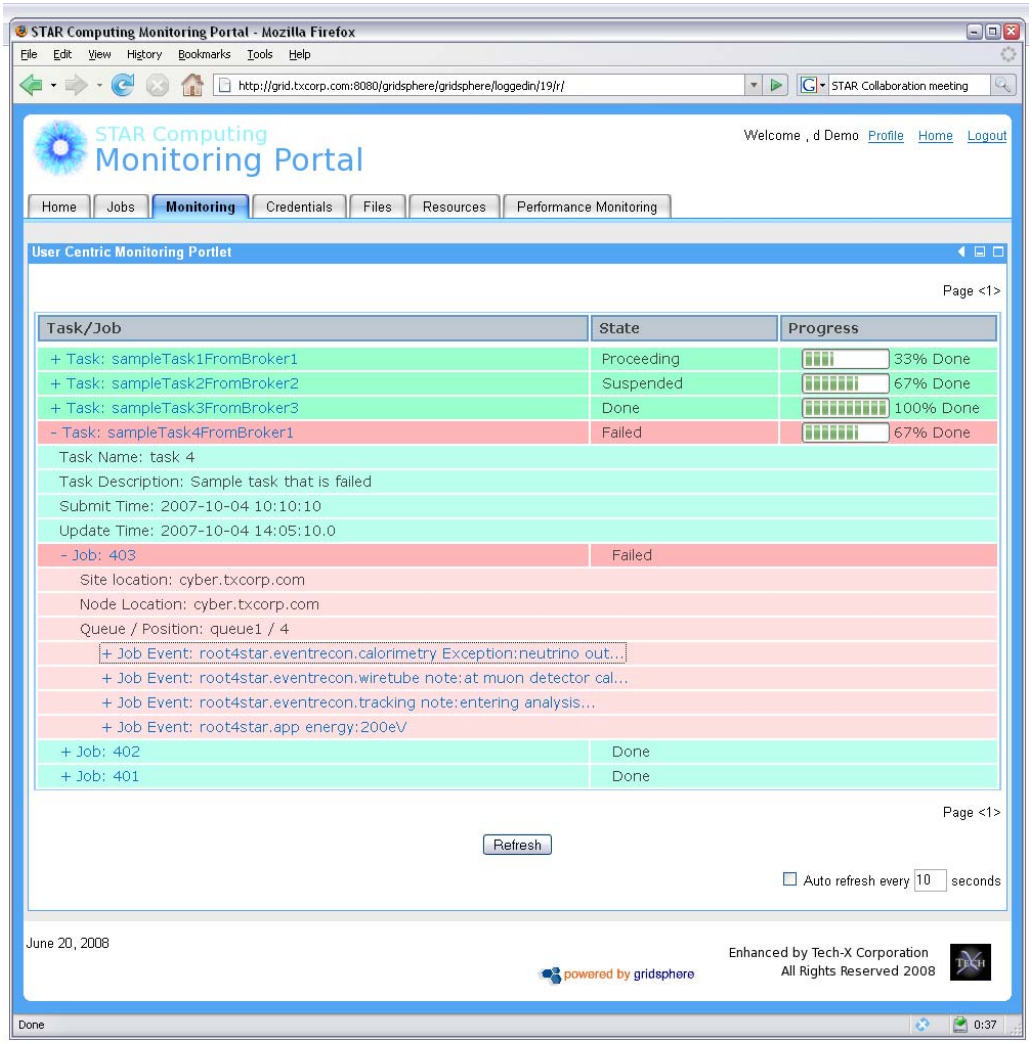

Figure 9. The UCM Portlet. There and task information lines, job information lines, and event information lines.

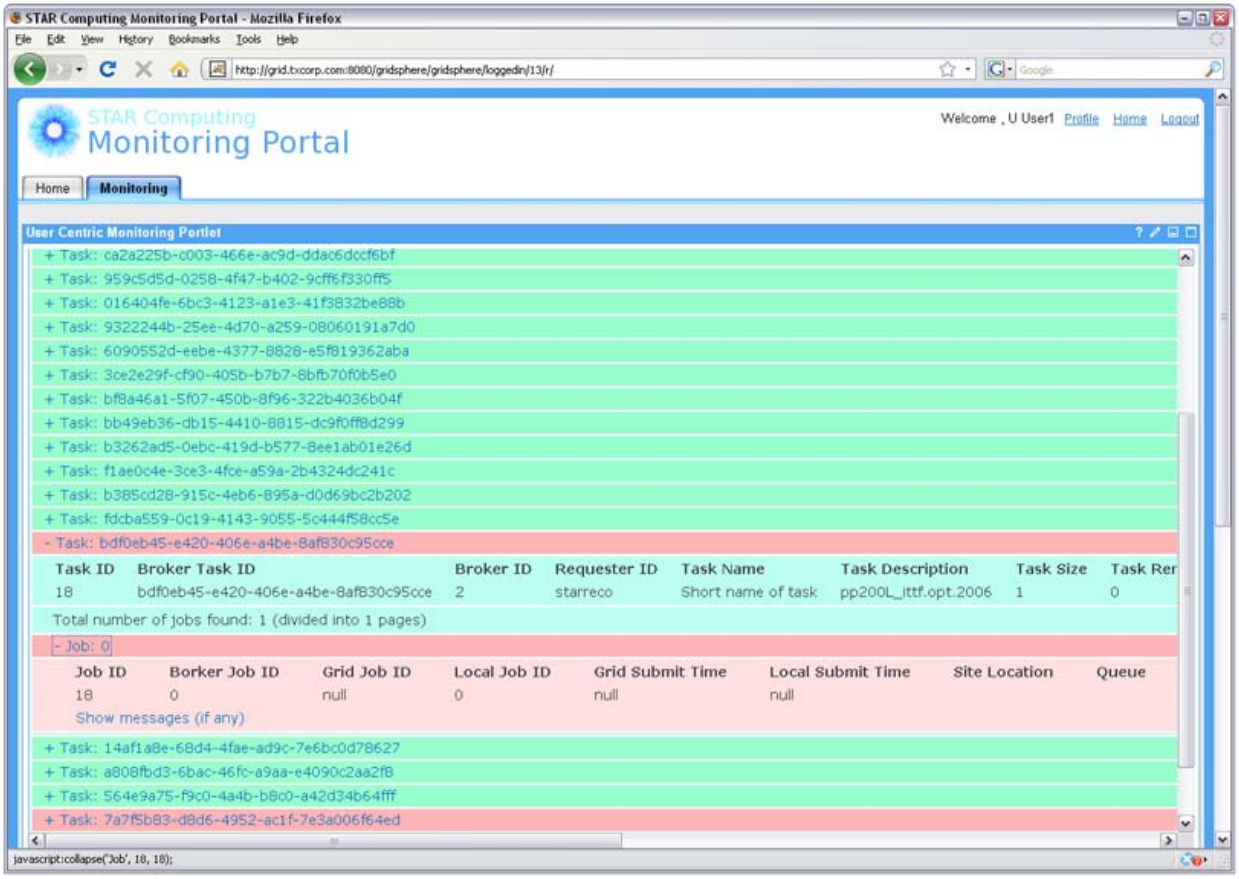

Figure 10. The Tracking Library has been integrated with root4star and has been tested in place with the UCM database at BNL along with the portal shown in this figure.

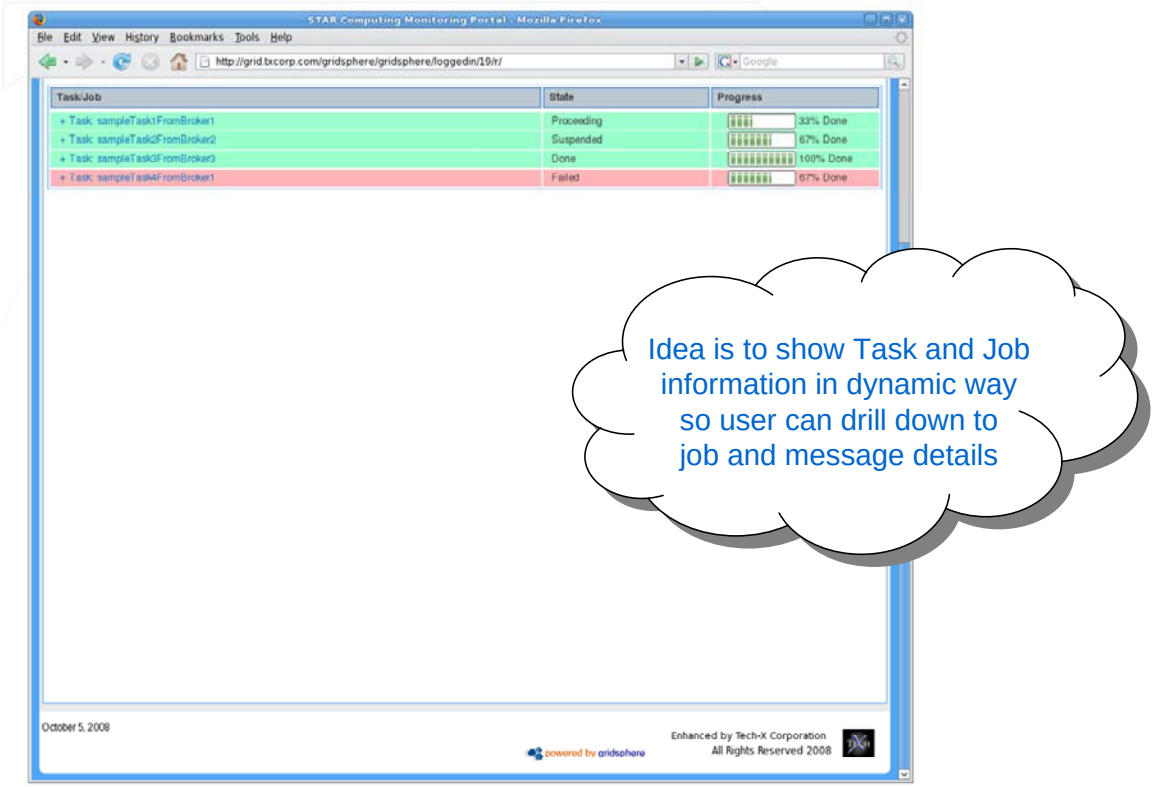

Figure 11. The users are first presented with a list of their tasks or jobs.

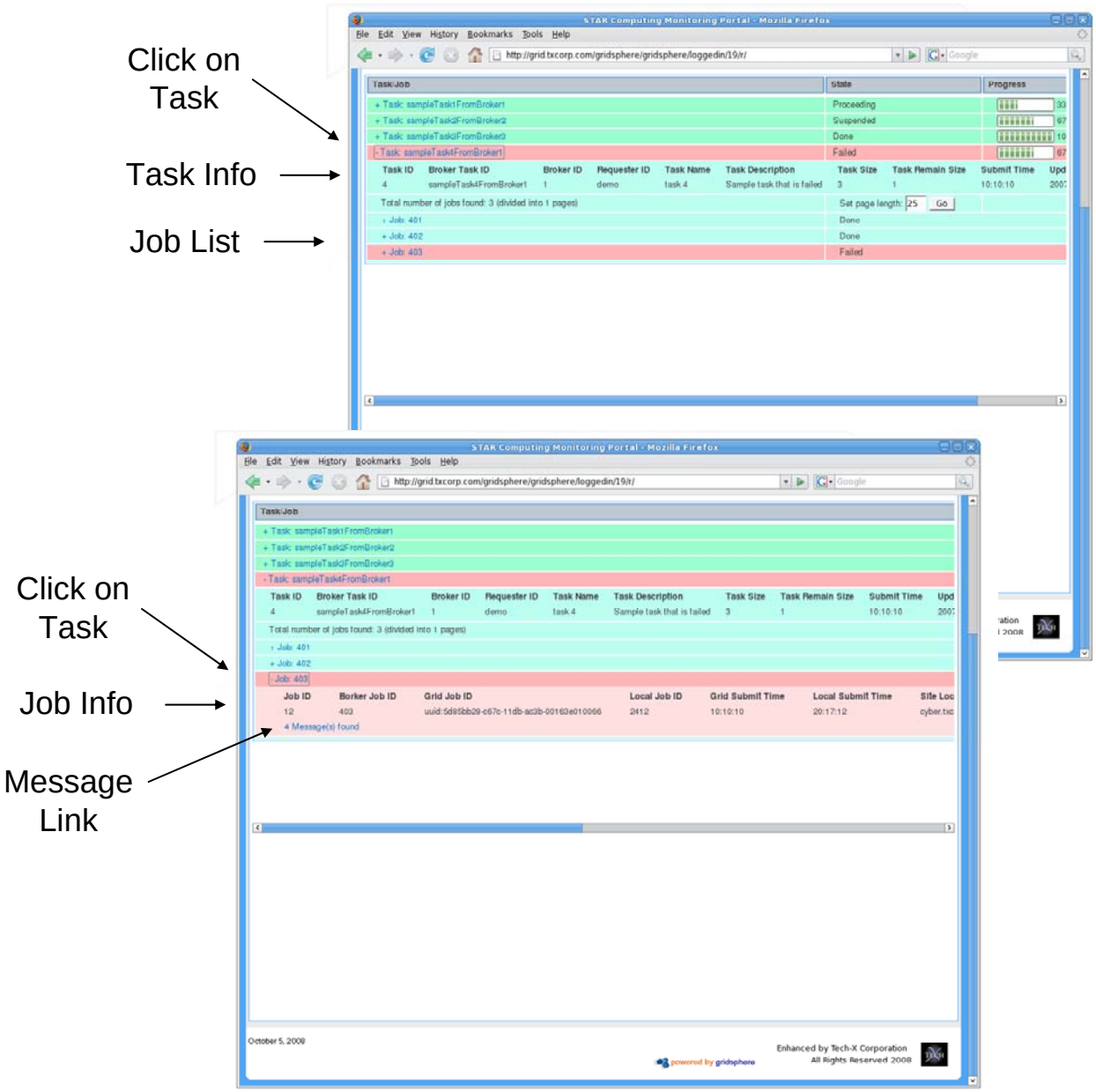

Figure 12. User's can click on specific jobs or tasks to drill down into monitoring information about their grid submissions.

Figure 13 below illustrates that as the portlet is implemented, it can be integrated easily into a number of portals with various portlets of choosing by any VO. Tabs can be created and then portlets can be added to the tab layout. The UCM and CEDPS portlets

are one of a number of portlets that could potentially be used. For example, the Vine Toolkit provides a number of job submission portlets for use with Globus based grids.

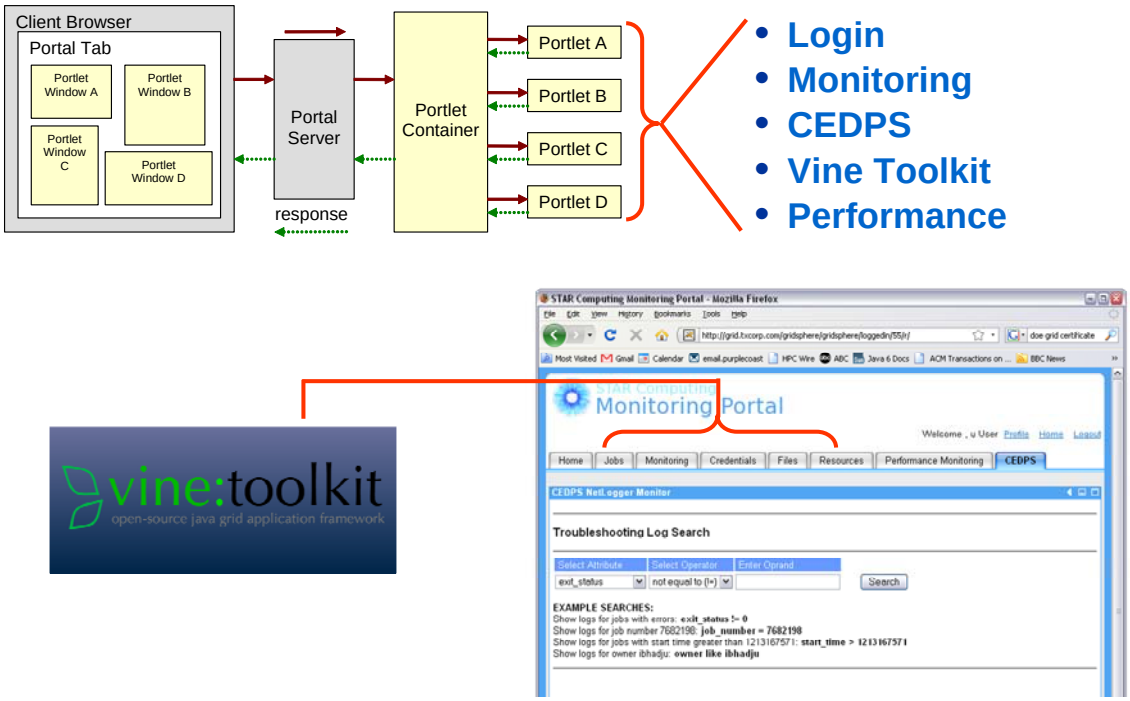

Figure 13. The portal solution was chosen because of its extensive nature where the UCM portlet can be contained in a portal along with other portlets.

IV. APPENDIX (UCM GRID PORTAL PAPER)

Below is a copy of the paper titled, "Automatic Certificate Based Account Generation and Secure AJAX Calls in a Grid Portal" with authors Mark L. Green, David A. Alexander, Roopa Pundaleeka, and James Matykiewicz all from Tech-X. This paper was refereed and published in conjunction with the Grid Computing Environment Workshop at SC08.

Automatic Certificate Based Account Generation and Secure AJAX Calls in a Grid Portal

Mark L. Green, David A. Alexander, Roopa Pundaleeka, James Matykiewicz
Tech-X Corporation
5621 Arapahoe Avenue Suite A
Boulder, CO 80301
+1 (303) 448-7751

mlgreen@txcorp.com, alexanda@txcorp.com, roopa@txcorp.com, james@txcorp.com

Abstract

Virtual organizations are interested in providing secure grid-related services to individual scientist users through portals and invest significant time and effort in managing them. Systems are often in place for users to request and receive Grid certificates in many grid infrastructures as a basis for their identity in the grid. These identities are, however, typically disconnected from web portal accounts and this presents an administrative maintenance problem. Furthermore, integrating certificate identities with grid portal identities is complicated when dynamic AJAX technology is used in the portal to connect to services outside the portal. The User Centric Monitoring project has developed a solution for automatic generation of web portal accounts that can be synchronized to a pre-existing list of grid certificate identities.

Categories and Subject Descriptors

H.3.5 [**Information Storage and Retrieval**]: Online Information Services – *web-based services.* C.2.4 [**Computer-Communication Networks**]: Distributed Systems – *Distributed applications, Distributed databases.* H.5.2 [**Information Interfaces and Presentation**]: User Interfaces – *User-centered design*

General Terms

Management, Design, Experimentation

Keywords

Grid computing, Web 2.0, Java Portal, AJAX.

1. Introduction

With the advent of Web 2.0, virtual communities and organizations are growing at a large rate. In order to integrate these communities seamlessly within web portals there is a strong need to manage large numbers of user identities and accounts. Traditionally, this is done in an ad-hoc fashion by the system administrator and requires a significant amount of knowledge and labor. Web portals (Open Grid Computing Environments [1], GridSphere [2], JBoss [3], etc.) provide basics frameworks for account management within the browser interface. Furthermore, virtual communities generally require some type of identity credentials such as X509-based certificates. These two discrete identity management systems present a significant administration and synchronization problem.

In the development of the User Centric Monitoring (UCM) project [4], we came up against this very issue. The UCM project aims at providing job and application monitoring services directed to the Solenoidal Tracker At RHIC (STAR) [5] physics experiment users. The STAR organization manages their user identities with the Virtual Organization Membership Service (VOMS) [6]. In trying to provide the STAR users with a portal, we needed a manageable way to automatically create and update portal accounts based on the identities listed in VOMS. Furthermore, it is important that STAR users do not have to memorize addition portal logins. It was during the course of developing a UCM portal for the STAR users that we developed a solution to this general problem that alleviates the burden of account generation from the portal system administrator.

Upon secure access to the UCM portal subsequent services need to be accessed. If these services are accessed through the web portal via portlets, then the entire browser page is redrawn and the user looses any potential for a dynamic application-like interface. In the STAR UCM portal it is crucial to provide a multi-portlet page that allows the user to dynamically interact with numerous attributes, information trees, and graphics. These dynamic interfaces can be provided with technologies such as AJAX [7]. However, with AJAX, the browser is bypassing the portal and any security layer that it would provide. Therefore, in addition to providing a basic certificate authentication layer, we have found a solution that allows AJAX calls to leverage this security layer thereby providing both the desired dynamic and secure features.

M Green, D Alexander, R Pundaleeka, J. Matykiewicz, Automatic Certificate Based Account Generation and Secure AJAX Calls in a Grid Portal, International Workshop on Grid Computing Environments November 16,, 2008, Austin, TX, USA. Work Supported by the U.S. Department of Energy Small Business Innovative Research Grant #DE-FG02-05ER84170.

2. Background on Grid Portlet Security

A common solution for providing Grid security for portal operations is to use the MyProxy [8] credential management service. MyProxy provides a solution for delegating credentials to a Grid portal to allow it to authenticate to Grid services on the user's behalf, allowing users to submit compute jobs, transfer files, and query Grid information services from a standard web browser.

The typical use case for MyProxy with a Grid portal is shown in Figure 1. The first step is for the user to store a Grid credential on a MyProxy server that the portal can access. Some portals are configured to use specific MyProxy servers, whereas others allow the user to specify a particular server. This first step requires that the user have Grid middleware installed on their machine and to use the command-line tool `myproxy-init` to load the credentials. The second step requires the user to log into the Grid portal in order to use the services that it provides. This login is normally a user name and password that is a separate account specific to the portal and is stored in the database associated with the portal.

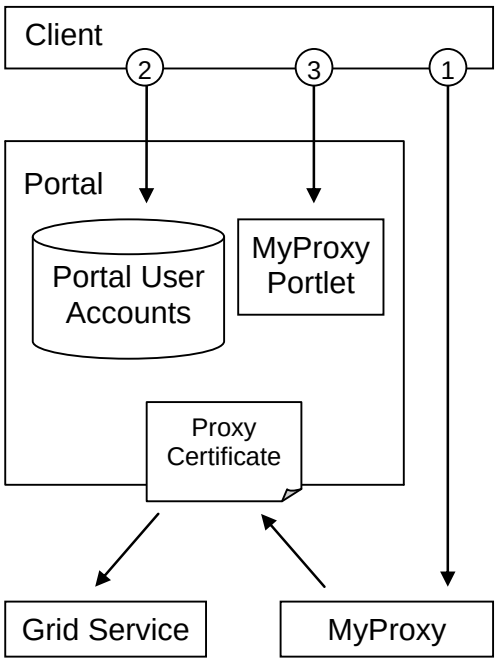

Figure 1. The typical use of MyProxy in a Grid portal requires the user to (1) use a command-line tool to load their credentials into a server; (2) log into the portal with a Grid portal account; and (3) use a portlet to get a proxy certificate with the same password that was used to load the certificate into the server in step one.

These user accounts are either maintained by a portal administrator or a registration portlet is used to allow users to set up their own accounts. The third step involves another portlet that collects a passphrase from the user that the portlet can subsequently use to contact the MyProxy server and retrieve a proxy certificate.

The account generation scheme described in this paper leverages the fact that the user often first has to use the browser to obtain a Grid certificate, which is certainly the case for the U.S. Department of Energy Grid certificates.

3. The UCM Portal

The UCM Portal utilizes Gridsphere portal framework version 3.1 to manage and contain the portlets discussed in the paper. Gridsphere 3.1 provides facilities for managing user accounts, basic security, portlet content and layout, and it is JSR-168 compliant [9]. The UCM related portlets are also JSR-168 compliant to insure portability and maintainability when migrating the portlets to new (portal) frameworks. We have taken advantage of the open-source code of Gridsphere 3.1 to customize the UCM portal with content, including project specific messages or logos, and layout of portlet controls (features).

We created a source repository for the UCM portal that includes Gridsphere 3.1 source code and build (Apache Ant) files. In addition, the UCM portal source repository includes a binary distribution of Apache Tomcat as the Gridsphere container. This developer "sandbox" approach allows rapid changes of portlets and Gridsphere source code to provide customized look and feel to the UCM portal.

Gridsphere 3.1 manages portal content and layout using XML files. Tech-X has developed new content and layout XML files, which overwrite the default files distributed with Gridsphere for certain customizations. Changes to Gridsphere source code include: modifying the login portlet to use Grid certificates, discussed in detail in section 5; improvements to logging controls; and extracting the database. These changes were to improve logging control and extract the database schema to manage it independently, especially when migrating portal user information. Gridsphere 3.1 provides solid performance for JSR-168 compliant portlets 'out-of-the-box'.

The most relevant portlet for this paper is the Certificate Login Portlet, which we describe in detail in the section 5 below. The Application Monitoring Portlet is another portlet, for which we describe as an example of how we can securely use AJAX with a portlet is section 6. First we describe the UCM portal deployment and configuration. The UCM deployment may be of general interest because it is compact (resulting in one tar file that is unpacked and one script that is subsequently run).

4. Portal Deployment and Configuration

In dealing with large and diverse user communities a flexible authentication and authorization model that is scalable, extensible and does not require excessive administration is required. We have developed a solution with an initial implementation that integrates X.509

identities, a selection of Virtual Organization based member lists (see Figure 2), and the Gridsphere 3.1 portal framework.

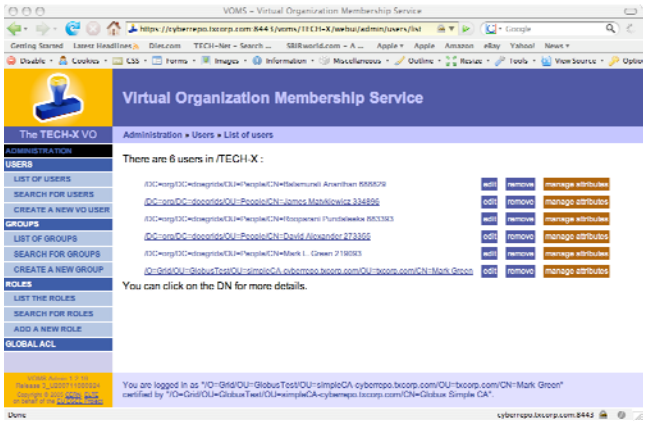

Figure 2. The Tech-X Corporation VOMS member listing hosted by the cyberrepo.txcorp.com server, which is accessed by the UCM Portal server grid.txcorp.com.

The UCM Portal is designed to be easily deployed and configured by administrators that do not necessarily have extensive system administration experience. We have developed a simple `ucmportal.properties` file that contains the required system definitions for deploying the Portal successfully. An example listing of the relevant parts of this `ucmportal.properties` files are listed below in order to clarify the following implementation discussion:

```
# ------------------------------------------------
# Portal Server Host Name
#
host.name=cyber.txcorp.com
host.httpport=8080
host.httpsport=8443
host.country=US
host.state=CO
host.city=Boulder
host.organization=Tech-X Corporation
host.unit=User Centric Monitoring

# ------------------------------------------------
# Database Connection Information for UCM DB.
#
ucmdb.name=ucmdb
ucmdb.host=cyber.txcorp.com
ucmdb.port=3306
ucmdb.username=ucmuser
ucmdb.password=strongpassword

# ------------------------------------------------
# Database Connection Information for Gridsphere
DB.
#
gsdb.name=gridsphere
gsdb.host=cyber.txcorp.com
gsdb.port=3306
gsdb.username=gsuser
gsdb.password=strongpassword

# ------------------------------------------------
# VOMS Connection Information.
#
# TECH-X VO VOMS
voms0.hostcert=/etc/grid-security/hostcert.pem
voms0.hostkey=/etc/grid-security/hostkey.pem
voms0.certtype=PEM
voms0.capath=/etc/grid-security/certificates
voms0.connection=https://cyberrepo.txcorp.com:8443
/voms/TECH-X/webui/admin/users/list
```

5. UCM Portal Login Components

In this section, we describe the login portlet solution that was put into place in the UCM project. As we will show this solution is very general and could be reused in almost any VO's grid portal system. In our first prototype solution, we did not disable the normal or default Gridsphere Login Portlet access to the grid portal. Instead, we added a secure (HTTPS) instance of Tomcat to host the new login portlet that is certificate-based. This way, if any validation step fails in the certificate connection, the system can fail-over to the default Gridsphere Login Portlet (note: this Login Portlet as well as the entire Gridsphere Portal can be deployed using HTTP or HTTPS solely based on the administrators discretion). However, subsequent portlets can rely on the success of the login, such as the shared secret security function of the UCM database Servlet discussed in Section 6.3.

Figure 3 describes the UCM Portal authentication and credential management components, each step in the authentication and authorization process shown in the figure is described below.

1) The user has joined a supported Virtual Organization and has loaded the corresponding X.509 certificate into a browser.

2) The UCM Portal HTTPS Tomcat server is automatically configured using the Java *keytool* and the *ucmportal.properties* file parameters *host.name*, *host.httpport*, *host.httpsport*, *host.country*, *host.state*, *host.city*, *host.organization*, and *host.unit*. The ucmportal.sh script parses these values and generates a Java `truststore` for utilization in Tomcat along with all Tomcat configuration files required for starting the UCM Portal. This configuration is only done during the initial UCM Portal setup procedure and does not have to be repeated. In addition, the administrator has the opportunity to import as many trusted Certificate Authorities root certificates into the `truststore` at this time. We now have the capability of requesting the users credential and subsequently obtain their identity. If this process fails we redirect to step 4 and otherwise continue.

3) During the UCM Portal configuration process the administrator has the opportunity to allow Portal access by acceptable Certificate Authorities. Upon obtaining a users identity we validate the X.509 certificate root Certificate Authority (CA) against our acceptable list. If the root CA is accepted we proceed to step 5, otherwise we are redirected to step 4.

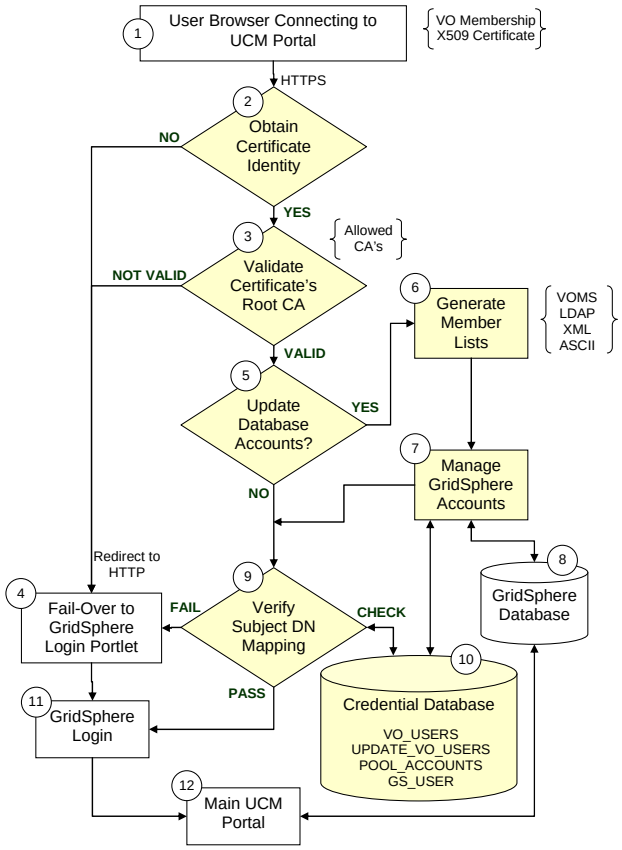

Figure 3. The UCM automatic account management is designed to integrate seamlessly with GridSphere account management and fail over to regular GridSphere login. The new components are shaded, whereas the normal GridSphere components are clear.

4) The failover Gridsphere username and password Login Portlet is invoked in the event of credential-based authentication failure, then proceed to step 11.

5) An automated process for initializing and maintaining the Gridsphere accounts utilizes the *update_vo_users* database control table, shown in Figure 4. This table grants the UCM Portal administrator the ability to configure when account updates are initiated, by default 0, 1, 3, 6, 12, or 24 hour increments are provided. The administration may select 1 hour if the VO has a volatile member list that changes quite often, otherwise a 24 hour increment will essentially update once per day. If an update is required we proceed to step 6 otherwise to step 9.

6) There are several ways to generate an acceptable member list the UCM Portal has the capability of using VOMS, LDAP, XML, or ASCII providers for this task. Essentially, any provider that returns the current member list with the users X.509 subject Distinguished Name (subject DN) will suffice. The *ucmportal.properties* previously listed defines the required VOMS parameters *voms0.hostcert*, *voms0.hostkey*, *voms0.certtype*, *voms0.capath*, and *voms0.connection* for the Tech-X VO VOMS. An unlimited number of providers can be configured in this file.

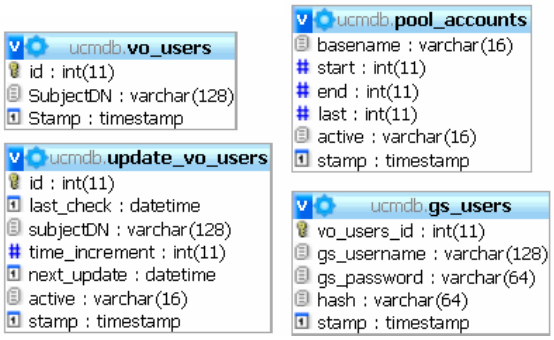

Figure 4. UCM Portal Database Credential Tables where created to manage the dynamic GridSphere accounts needed to map certificate-authenticated users.

7) After generating a list of allowable members the UCM Portal infrastructure will either generate, update, activate, or de-activate the Gridsphere accounts automatically. If the supported UCM Portal member listing (as noted before can be obtained from several sources) no longer includes one of the active accounts generated by the Portal the account is preserved but inactivated by changing the gridsphere database sportletuserattrbutes attribute "gridsphere.user.disabled" value to "TRUE". This de-activation will prevent the member from accessing the Portal through the credential-based and username/password pathways. If a new member subject DN is identified, then a gridsphere account will be generated using the UCM database pool_accounts table parameters and populated with the members relevant information such as username, password, and email address. We should also note that the ability to change the members automatically generated account password is restricted, as the current password is not automatically given to the member. During the automatic update of the supported UCM Portal member listing all email addresses at updated in the gridsphere database so that contact emails can be sent by the administrator if desired. The UCM database table *pool_accounts* provides the control for this process utilizing the *basename*, *start*, *end*, *last*, *active*, and *stamp* fields. The UCM Portal administrator can configure the format of the Gridsphere username generated by modifying the active table record, the default username account is *basename_number*.

8) The initial Gridsphere database is generated during installation and in general is only configured for

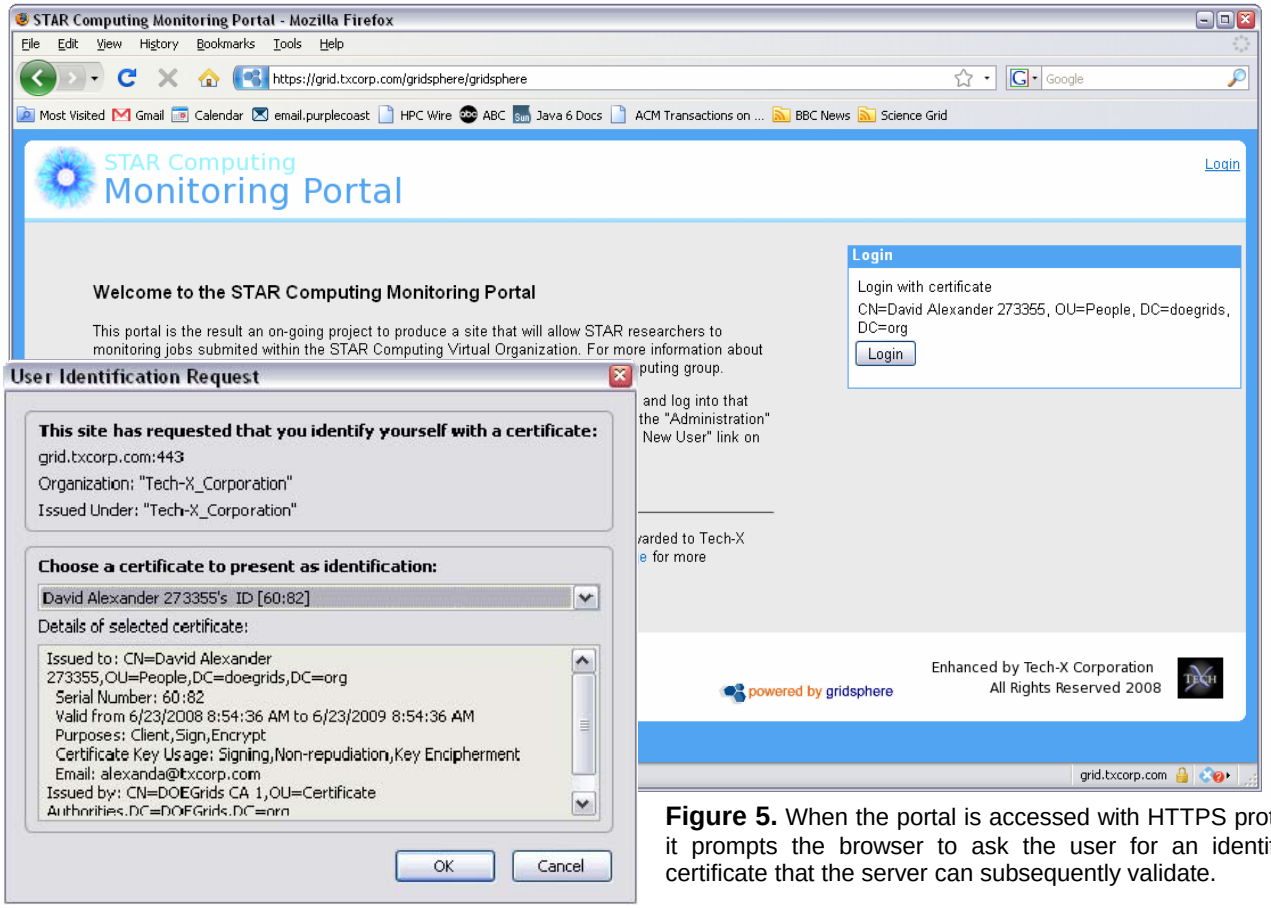

Figure 5. When the portal is accessed with HTTPS protocol it prompts the browser to ask the user for an identifying certificate that the server can subsequently validate.

administration access, thus contains only one account. We directly modify the Gridsphere database during the automated account generation process so that the administrator does not have to create each individual account through the user interface. Thus, hundreds of accounts can be generated and managed quite efficiently as the UCM Portal supported providers expand.

9) Based on the users certificate identity (subject DN) provided from step 2 we verify whether a Gridsphere account exists by querying the UCM credential database (see Figure 4), specifically a simple table join of *vo_users* and *gs_users* will return the appropriate information needed (i.e. SELECT vo_users.id FROM vo_users, gs_users WHERE vo_users.id = gs_users.vo_users_id AND vo_users.SubjectDN LIKE 'target subject DN' LIMIT 1). If this process fails the user is redirected to step 4 otherwise continue to step 11.

10) The UCM Portal credential database tables are listed in Figure 4. Specifically, the *gs_users* table contains the automatically generated Gridsphere account information, *vo_users* table contains the subject DN mapping information, *update_vo_users* table provides the account update and generation control information, and *pool_accounts* provides the account generation naming and limits information.

11) The standard Gridsphere login infrastructure is used to continue the authentication and authorization process for both the username/password and credential-based Portlets.

12) Finally, after successfully passing the Gridsphere login the user is presented the UCM Portal Welcome Portlet or on failure is redirected to the UCM Portal Login Error Portlet.

The end user experience of the above steps during login depends on the browser, but basically involved some sort of dialog windows popup prompting the user with an acknowledgement that a server is requesting a certificate for authorization or a choice of certificate to present to the server. Figure 5 shows the view that the *Firefox* user would see including the certificate choice dialog and the portal welcome page with the user's credential information after login.

6. UCM Portal Application Monitoring Components

The application monitoring portlet was designed to provide users with job and simulation monitoring information. The details of the monitoring functionality of this portlet are described in a previous paper [4]. The important feature of this portlet example for the discussion

of this paper is that it has AJAX code within the portlet that asynchronously and securely contacts a Servlet. The page happens to be part of Java Server Pages compilation and the Servlet happens to further contact a MySQL [10] database in our case, but these are details that are not important to this paper and we only show them for completeness.

6.1 Monitoring Portal Architecture

The UCM project uses a typical portal architecture for the containment of the monitoring portlet user interfaces, but also includes a database query Web Service (Servlet) component running in same instance of the Tomcat Servlet container. Figure 6 shows the overall architecture for the UCM portal.

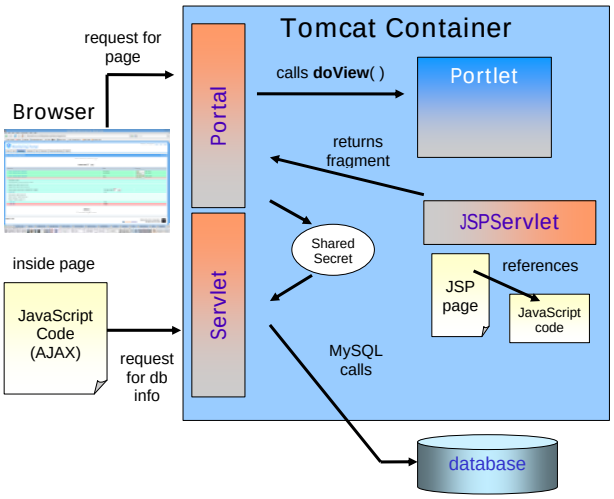

Figure 6. This figure shows the architecture associated with AJAX communication to auxiliary Servlet functionality of a portal such as querying a database.

Unlike the login portlet discussed in the previous section, the monitoring portlet heavily relies on not only this database query Servlet, but also the Java Server Pages Servlet to generate the default page and include the Java Scripting described in section 6.2. The shared secret functionality of the database Servlet is discussed in section 6.3

A Portal is a web application that typically provides services such as personalization, single sign-on, and content aggregation from different sources. Portlets are the visible active components that users see in a portal page. A Portal or a Portlet Container is essentially an aggregation of portlets within one web page. BEA, IBM, Oracle, etc. provide commercial Java Portals, while GridSphere, JBoss, Pluto, Liferay, Stringbeans, etc. are some of the open source portals. JSR 168 provides standards for building portlets that can, run on portals provided by any vendors, and most Java Portal Web Servers support the JSR 168 specification. [11]

Implementing a JSR 168 portlet begins by extending GenericPortlet, which is an abstract class that provides a default implementation for the portlet interface. The GenericPortlet subclass should override at least one method:

1. **processAction**: for handling action requests.
2. **doView**: for rendering requests when in View mode.
3. **doEdit**: for rendering requests when in Edit mode.
4. **doHelp**: for rendering requests when in Help mode.
5. **init** and **destroy**: for managing resources held for the life of the portlet.

The JSR 168 specification defines three portlet modes: *View*, *Help*, and *Edit*. All portlets must support the *View* mode by overriding the *doView()* method, which renders requests by generating markup to display the portlet when the portal is loaded on the browser.

One of the most expensive actions in a portal environment is to refresh the whole page. When the user clicks a link or takes some action on the page, the portal processes the request for the target portlet and also the doView() methods for each portlet on the page. It then aggregates the results and sends the entire HTML document down to the browser. In our implementation, this is done once when the portal is loaded, and all subsequent user requests are rendered asynchronously using AJAX, thus reducing the overhead.

We use a JSP page to display the contents of the portlet when the portlet's *doView()* method is called. All actions in the JSP page are associated with JavaScript functions which make an asynchronous request to a Servlet, which fetches data from a MySQL database and returns the results to be rendered on the browser dynamically, without having to refresh the whole page.

6.2 Web Application Features

One of our goals during the UCM Application Monitoring Portal development was to be able to design and develop Portlets that could be rendered independently without having to refresh the whole portal page, using AJAX.

Using JavaScript technology, the HTML page can asynchronously make calls to the server from which it was loaded and fetch content that may be formatted as XML documents, HTML content, plain text, or JavaScript Object Notation (JSON). The JavaScript technology may then use the content to update or modify the Document Object Model (DOM) of the HTML page.

There must be an associated web application, which in our implementation is the database access Servlet. This

Servlet, for example, could be used to retrieve data from another back-end store or service. In our case, this Servlet queries a remote MySQL server and creates the response in HTML format. When the Servlet returns, the ready state is set to 4, and the response will be handled by the return handler in the JavaScript asynchronously. The JavaScript can render the HTML DOM to produce the desired display on the portlet web page without having to refresh the portal page.

The key enabler of AJAX is the `XMLHttpRequest` object, implemented by most browsers. When used within a JavaScript on an HTML page, the `XMLHttpRequest` object can make asynchronous calls to the same HTTP server for other content by:

1. Creating `XMLHttpRequest` object compatible to the browser
2. Calling the `open()` and `send()` methods on the object
3. Providing handling for the `onreadystatechange` event the request object will return when content becomes available

There are quite a few JavaScript toolkits available such as DOJO, Yahoo UI, etc. which provide AJAX capabilities with in-built functions for various browsers. But they also come with a huge set of widgets and utilities which we do not currently require, and would increase the footprint of our portal package. Hence we decided to use the basic JavaScript functions to make the AJAX calls.

6.3 Security Features

The means for the security in the UCM AJAX-Servlet connection rests on being able to share a security context between the Portal and Servlet. This is possible because they are able to access the same UCM credentials database. The credentials database contains an identity string and an MD5 "message digest" or 128-bit hashed version of this string base.

In normal operation (see Figure 7), the user would log into the portal and then the monitoring portlet display would be sent the user's browser. Along with the web page with the monitoring default display, the portal would also send AJAX code in a separate JavaScript file. Included here would be an MD5 hashed digest of their identity plus a secret that is only shared between the Portlet and Servlet. This MD5 was generated at the time that their identity was loaded into the UCM Credential Database. This is done when the database is synchronized with the virtual organization information, which depending on the configuration of the Portal can be a short interval (each time the user logs in) or a longer interval (for example, once a day).

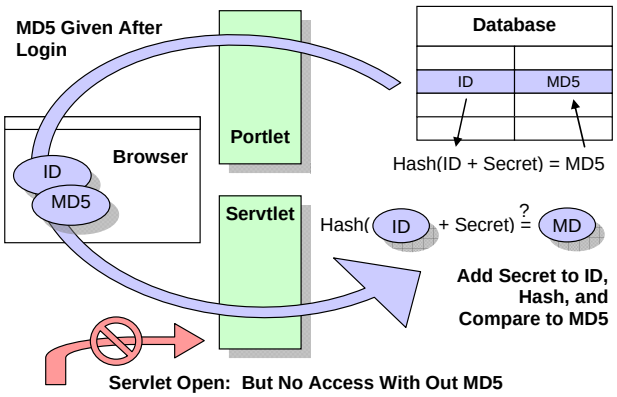

Figure 7. A shared secret allows secure access of Servlet-based Web Service functions that are called from the AJAX code within the Web browser.

Now that the AJAX code has the identity and the MD5 digest, it can then access the Servlet with this combination on subsequent operations that are initiated by the user in the browser. The Servlet can then on each operation check the validity of the combination by taking the identity, adding the shared secret, hashing the result, and comparing it to the MD5. Malicious attempts to access the Servlet, which is exposed to the internet, can be denied based on the lack of presented the correct MD5.

While not bullet proof, this level of security is sufficient for the UCM project. The possible failures include intruders listening on the traffic between the client and Portal (the so called man-in-the-middle attack). The initial login traffic is encrypted, but for UCM we have not encrypted all traffic to maximize the performance the information flow. One could encrypt all traffic for even better security, but there would be a performance penalty for this. Having the hashed value plus the ID unprotected does expose some small level of risk, but this can be mitigated by choosing lengthy secrets, changing them frequently, or using a new secret every session. Our current prototype is a simple case in that the Servlet and Portal are co-located. As we discuss in Section 7, there are other tools that might be considered if additional off-portal services are accessed.

7. Related Work

The share secret approach here has some similarities to the OAuth standard [12]. OAuth is an open protocol to allow secure API authorization in a simple and standard method from desktop and web applications. Using the API, the user might log into a Consumer website asking it to perform some service for them using their protected data that is on another site, the Service Provider. The Consumer sends the user to the Service Provide with identifying information about it and what it is trying to access. At the Service Provider, the user signs in and is asked the

Consumer is allowed to access the protected data. If the user agrees, they are sent back to the Consumer with a special Single-User Token. Finally, the Consumer takes the Single-User Token and contacts the Service Provider to exchange it for a Multi-Use Token that it can use to access the protected data. The underlying mechanism of hashed digests of user information plus shared secrets are similar to both the project described here and OAuth. However, OAuth is a standard that has clear advantages when the services are in separate domains. In the current UCM prototype, there is only a need for one service that is co-located with the Portal and OAuth did not make sense to use. However, with future expansion of the Portal or in other projects OAuth might make good sense.

Other related work includes efforts to place virtual organization information into login exchange such as the Shibboleth [13] single sign-on and federating software. In this framework extra information about an individual's access rights to services is appended to basic identity information, principally through the use of OASIS' Security Assertion Markup Language [14]. Using assertions with the VO information self contained would give architecture presented in this paper a more "real-time" quality since it would eliminate the need to synchronize with an external source such as a VOMS system. The UCM system does have flexibility of when this synchronization happens, but if the information is passed into the system via the assertion, then the database credential information is somewhat obsolete. The STAR experiment does not use SAML-enhanced assertions, so this approach was not used for the UCM application. It is also of note, that UCM performance was anticipated to preclude use of such assertions at the Servlet-Service, although scaling tests to large number of users are pending.

8. CONCLUSION

In this paper, we have shown a web portal that allows the use of dynamic AJAX-based portlets that connect to backend Servlets all within an X509-certificate-based security context. This security context is manageable for the VO because of the infrastructure that is able to automatically synchronize the generation and updates of grid portal accounts mapped to a set of grid certificate user identities. In our case, we use a VOMS list of certificate holders, but in general the system can be extended to most any type of identity list.

The result is a grid portal that has a simplified user interface in that they can present a certificate that is already likely to be loaded into their browser and automatically be recognized without having to remember a separate portal login. Additionally, for the administrator of the system it is also simple in that they only have to maintain a list of virtual organization providers and do not have to worry about generating individual portal accounts.

9. ACKNOWLEDGMENTS

The work reported here is supported by the U.S. Department of Energy SBIR Grant #DE-FG02-05ER84170.

10. REFERENCES

[1] Jay Alameda, Marcus Christie, Geoffrey Fox, Joe Futrelle, Dennis Gannon, Mihael Hategan, Gopi Kandaswamy, Gregor von Laszewski, Mehmet A. Nacar, Marlon E. Pierce, Eric Roberts, Charles Severance, Mary Thomas: The Open Grid Computing Environments collaboration: portlets and services for science gateways. Concurrency and Computation: Practice and Experience 19(6): 921-942 (2007), also see http://www.ogce.org

[2] Jason Novotny, Michael Russell, Oliver Wehrens: GridSphere: a portal framework for building collaborations. Concurrency - Practice and Experience 16(5): 503-513 (2004), also see http://www.gridsphere.org

[3] See http://jboss.org

[4] D Alexander, R Pundaleeka, S Tramer, J Lauret, V Fine, Portlets for User Centric Job and Task Monitoring for Open Science Grid Virtual Organizations, International Workshop on Grid Computing Environments 2007, Nov., Day, 2007, Reno, NV, USA.

[5] Information about the STAR nuclear physics experiment can be found at http://www.star.bnl.gov/

[6] VOMS information can be found at http://edg-wp2.web.cern.ch/edg-wp2/security/voms/

[7] AJAX technology is a web development technique for asynchronously accessing server data while allow the user to interact with an existing web pages and updating parts of the page when the data is retrieved. See http://en.wikipedia.org/wiki/Ajax_(programming)

[8] Jim Basney, Marty Humphrey, Von Welch: The MyProxy online credential repository. Softw., Pract. Exper. 35(9): 801-816 (2005)

[9] The "JSR 168: Portlet Specification", http://jcp.org/en/jsr/detail?id=168, defines the interoperability between portlets and portals.

[10] MySQL is a highly-available, light-weight database server. See http://www.mysql.com/

[11] General information about JSR 168 compliant portals can be found at http://en.wikipedia.org/wiki/JSR_168

[12] OAuth, http://oauth.net/

[13] Shibboleth, http://shibboleth.internet2.edu/

[14] SAML, http://wiki.oasis-open.org/security