BNL-101061-2013-CRAD

FINAL REPORT

FOR CRADA NO. C-08-11

BETWEEN

BROOKHAVEN SCIENCE ASSOCIATES

AND

TECH- X CORPORATION

Project Entitled: CWS4DB: A Customizable Web Service for Efficient Access to Distributed Nuclear Physics Relational Databases

Brookhaven PI: Jerome Lauret

Submitted by: Michael J. Furey
Manager, Research Partnerships
Brookhaven National Laboratory

DISCLAIMER

BNL-101061-2013

FINAL REPORT

FOR CRADA NO. C-08-11

BETWEEN

BROOKHAVEN SCIENCE ASSOCIATES

AND

TECH-X CORPORATION

Project Entitled: CWS4DB: A Customizable Web Service for Efficient Access to Distributed Nuclear Physics Relational Databases

PROTECTED CRADA INFORMATION

This product contains Protected CRADA Information which was produced on 6/7/13 under CRADA No. C-08-11 and is not to be further disclosed for a period of five years from the date it was produced except as expressly provided for in the CRADA.

Brookhaven PI: Jerome Lauret

Submitted by: Michael J. Furey
Manager, Research Partnerships
Brookhaven National Laboratory

DISCLAIMER

FINAL REPORT
CRADA BNL-C-08-11

TITLE: CWS4DB: A Customizable Web Service for Efficient Access to Distributed Nuclear Physics Relational Databases

BROOKHAVEN PI(s): Dr. Jerome Lauret
RHIC/STAR Experiment, Physics Department
Phone: (631) 344-2450
Fax: (631) 344-3257
Email: jlauret@bnl.gov

INDUSTRY PARTNER: Tech-X Corporation

OBJECTIVE:

Tech-X proposes to develop a system providing an efficient access for the High-Energy and Nuclear Physics data stored in distributed heterogeneous relational databases. The system will consist of a generic Web Service, customized client APIs, and tools that facilitate generation of custom client APIs. The access to the data will be mediated by mechanisms that provide data caching and adaptive scheduling of the query plans.

PROJECT DESCRIPTION:

An increasing fraction of the data generated in Nuclear and High-Energy Physics is managed in distributed and relational databases. As the size of this data grows and the collaborative nature of these experiments increases, the ability to access differently organized relational databases remotely, efficiently and yet in a user-friendly and interoperable manner is becoming very important. This community lack tools addressing this need and accommodating related challenges.

Tech-X therefore proposes a system to overcome the outlined challenges by bridging relational databases with high-level APIs through Web services. In particular, the distributed and heterogeneous nature of the databases will be addressed by creating a Web service on top of OGSA-DAI, which provides mechanisms coordinating access to diversified data resources. Such Web service would not be easily integrated into legacy code and applications or present a user-friendly environment for configuration and management. Therefore, the challenge of allowing for high-level queries will be overcome by providing a means to generate customized interfaces on top of the Web service client and provide tools and infrastructure for load balancing, efficient caching, on-demand services and tiered deployment and management. Additionally, we intend to

address the challenge of efficiency of data access in the situations when there are many queries of different types over distributed data sources, so that the

number of data transfers is minimized. Other factors may also be taken into account when making query plan include workload of database servers, network bandwidth, pattern of requests, and priority of different operations.

ROLES AND RESPONSIBILITIES:

The STAR Software and Computing (S&C) team with share the responsibilities with Tech-X for the determination of Phase II specific CWS4DB system and load balancing additional requirements and properties definition, the design and implementation of an auto-caching infrastructure, the development of a prototype on-demand data resource node, and a prototype pre-cache capability for production job workflows. This collaborative work is essential for providing a useful, robust, and tested software infrastructure.

The STAR S&C project has the capability of supplying a realistic testbed for the CWS4DB framework, along with the operational experience to critically assess the system performance and efficient data access that is essential for the project success. Furthermore, STAR S&C will provide the deployment and integration use case for delivering the CWS4DB framework in a smooth and non-disruptive manner by deploying the prototype framework within their development and eventually (after a hardened product is developed) production environments.

ACCOMPLISHMENTS: See attachment.

SBIR Phase II Final Report

CWS4DB: A Customizable Web Service for Efficient Access to Distributed Nuclear Physics Relational Databases

November 15, 2011

Award Number: DE-FG02-07ER84757
Grant Supported by the DOE office of Nuclear Physics

Award Recipient: Tech-X Corporation

Reporting Period: August 15, 2008 - August 15, 2011

Submitted by: Dr. Mark L. Green, Principal Investigator
Email: mlgreen@txcorp.com
Telephone: 716-204-8690

Submitted to: Dr. Manouchehr Farkhondeh
Email: manouchehr.farkhondeh@science.doe.gov

Contents

1 Executive Summary

The CWS4DB system architecture was developed from the successful completion of our Phase I investigation and input from our Nuclear Physics collaborators. The Execution Node as denoted in Figure 1 is where the user jobs are executed. We are presenting two CWS4DB client possibilities in support of the Execution Node applications. The top CWS4DB client is MySQL specific and represents an extension of the Phase I project, the bottom CWS4DB client illustrates the further development of the Phase II project with a Phase III effort. The significant difference between the two is the further abstraction of the CWS4DB client library from a MySQL specific interface representation to a generic interface specification. This would be developed in a Phase III effort as it extends our CWS4DB MySQL client library to a vendor-neutral connector specification. The CWS4DB MySQL specific or generic SQL provide user code bindings and have the capability of generating Web Service query requests to the Data Resource Node CWS4DB service. The proposed query auto-caching capability described in the following tasks is denoted here as the "queryCache", this is where the cached query results are stored on the Execution Node.

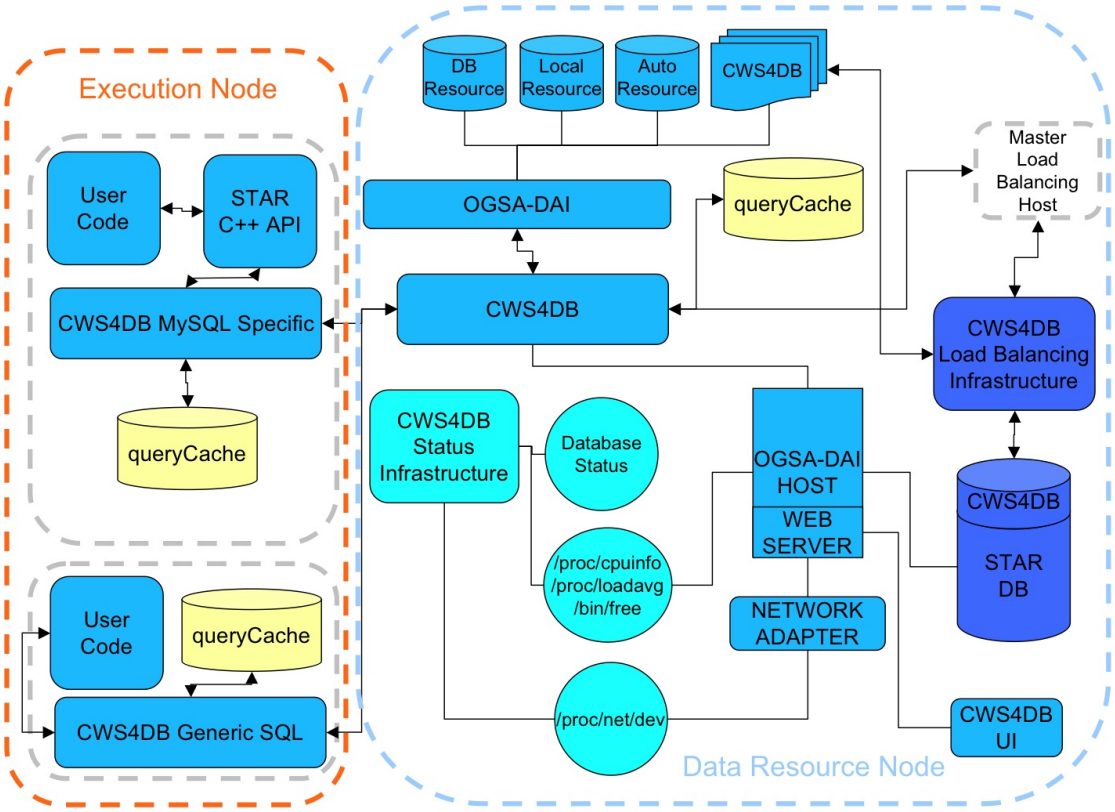


Figure 1: CWS4DB Architecture Diagram

The Data Resource Node as denoted in Figure 1 is where the main CWS4DB Web Services reside. The OGSA-DAI HOST provides a Tomcat server for the CWS4DB Web Service URIs. The back-end of the CWS4DB Web Services is supported by the OGSA-DAI system. The

OGSA-DAI data resources shown at the top of this figure are configured via the CWS4DB User Interface (UI). There are several different modes of operation under investigation for configuring OGSA-DAI data resources describe in the following tasks. The proposed query auto-caching capability described in the following tasks is denoted here as the "queryCache", this is where the cached query results are stored on the Execution Node. The CWS4DB Status Infrastructure depicted in the Figure 1 was developed in the Phase I project and will be including in the Phase II CWS4DB system. The CWS4DB Load Balancing Infrastructure stores the data resource node statistics in a local database or file and also transmits them to the centralized Master Load Balancing Host.

The Phase II objectives included taking into account what was learned from the research in Phase I and extending the CWS4DB prototype into a production-quality, load-balanced, auto-caching, grid-enabled, fault-tolerant, on-demand system that is described in the introduction. Each step of the work plan involved a separate piece of technical functionality that was implemented in way that can be exercised in the STAR computing environment, yet developed in a general way for application to other NP projects. The goal was to produce a set of software tools and services that can be easily adapted by the NP application developer.

2 Technical Achievements and Project Activities

During the project startup it turned out that it required far longer than expected to successfully obtain a Cooperative Research and Development Agreement (CRADA) with our Brookhaven National Laboratory (BNL) collaborators. Shortly after beginning the CWS4DB Phase II project in August 2008 the PI, Dr. Green, began the CRADA paperwork process including the refinement of the proposed project Statement of Work with our BNL STAR collaborator, Dr. Lauret. We continued to work diligently with the BNL research partnerships team and BNL attorneys until the CRADA was approved. We subsequently transferred the required funds to BNL so that our BNL collaboration could move forward unencumbered. Unfortunately, the delay in the CRADA put our collaboration with BNL behind schedule but through an outstanding commitment by Dr. Lauret we were able to identify an appropriate individual on the BNL staff to immediately start work on the CWS4DB project. In addition our Tech-X and BNL teams were able to move forward on accomplishing a significant amount of work on all TASKS listed in the Project Schedule.

The overall Phase II project tasks tracked the project schedule put forth in the Phase II proposal. The Phase II project schedule, Figure 2, is included below for reference. The Phase II task summary, goals, and our accomplishments, results, and analysis are listed in the following subsections.

TASKS	Quarters After Contract Initiation							
	1	2	3	4	5	6	7	8
1. Determine CWS4DB System and Load Balancing Additional Requirements and Properties	X	x						
2. Design and Implement Tiered Deployment Capabilities		x	x	x				
3. Design and Implement Auto-Caching Infrastructure		x	x	X	x			
4. Enable Multi-VO Role-Based Capabilities				x	x			
5. Develop Dynamic On-Demand Data Resource Access				x	x	x		
6. Develop Fault Resilient Data Resource Pathways					x	x		
7. Develop a Prototype On-Demand Data Resource Node					x	x	x	
8. Prototype Pre-Cache Capabilities for Production Job Workflow						x	x	x
9. Develop a Customizable Site Specific Test Suite							x	x
10. Write Progress and Final Reports		x		x		x		x

Figure 2: Phase II Project Schedule

We managed this project using *dotProject*, an open-source, multi-user project management tool that helps us organize the projects, tasks, companies, departments, events, contacts, to-do lists, resources and generate Gantt charts and reports. We input this Phase II project in dotProject, breaking each task into subtasks by their allotted quarter.

2.1 Task 1. Determine CWS4DB System and Load Balancing Additional Requirements and Properties

Objective: Extend the Phase I developed requirements and properties and continue prototype work with our partners. An expanded listing of requirements and properties obtained during the Phase I project that will further satisfy the STAR collaboration and in addition we strive to generalize these requirements to meet a broader scientific and enterprise community need.

Summary:

a) Expand the list of requirements and properties obtained during the Phase I project that will further satisfy the STAR collaboration and strive to generalize these requirements to meet a broader scientific and enterprise community needs
b) Investigate the potentially high-impact parameters identified in Phase I project
c) Investigate placement strategies for individual STAR databases on appropriately optimized servers
d) Perform additional research to determine the optimal grouping of STAR databases
e) Investigate the impact of the factors like wait, interactive, and connection timeout, join buffer size, query cache size and limits, max allowed packet size etc on the STAR database servers tuned for specific database and table support
f) Investigate incorporating the additional database statistics determined in the Phase I project into an objective function that can be fit based on actual root4star database query loading
g) Use the capability of generating root4star database queries based on typical STAR jobs to test and develop this system on Tech-X computer resources for developing the load balancing algorithm and system
h) Investigate genetic algorithm based optimization of this load balancing weighting function as indicated by preliminary results from the Phase I project
i) Perform further investigation which has the potential to yield additional requirements and properties

We have met our goals for this task over the course of this effort, building on the requirements from the Phase I effort and developing an architecture to meet the needs of our STAR collaborators. We discuss in particular our technical achievements and results from this effort in the following subsections.

Expand on Requirements

In working with our BNL collaborators, the CWS4DB system and load balancing requirements have been refined. In Figure 3, the *"data storage"* represents the backend data storage i.e. MySQL, Memcache, Distributed File System, etc. and *"pool"* is a set of *data storage* server nodes i.e. a group of identical database servers with a common purpose. Here a group of MySQL servers dedicated to data reconstruction tasks; for example may have *pool* categories: OFFLINE_PRODUCTION, OFFLINE_CALIBRATIONS,

OFFLINE_ANALYSIS, ONLINE_CALIBRATION, ONLINE_MONTORING, FILECATALOG, LOGGER, or TEST.

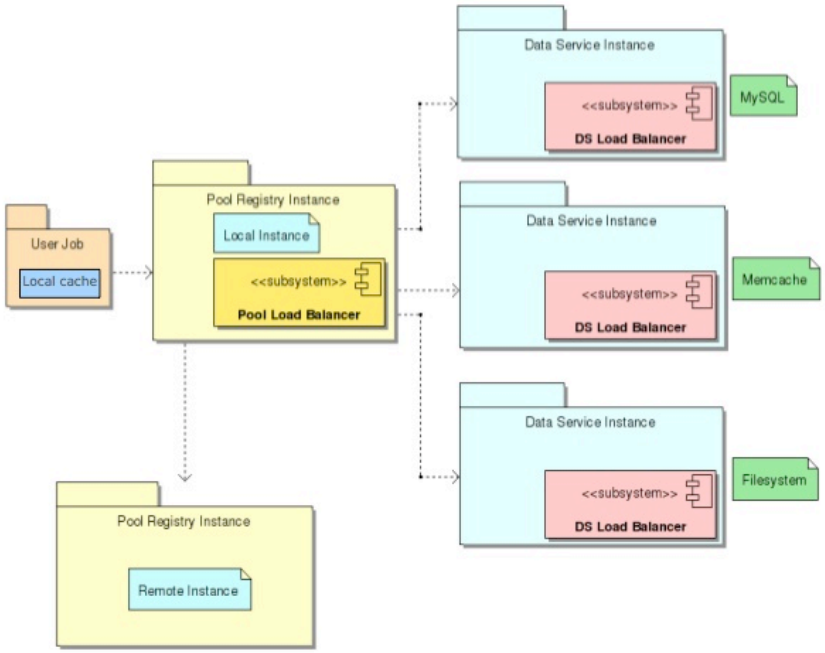


Figure 3: Generic component scheme consistent with the STAR collaboration point of view

Useful diagrams for the Data Service and Pool Registry Web services that met the needs of the STAR collaboration are presented in Figures 4 and 5.

The **Data Service**, Figure 4, is a Web service package that knows how to execute requests on a single *data storage* type, and how to collect performance metrics for this type. The desired **Data Service** typical capabilities are: a) announce one or more *pools* and related performance metrics to a **Pool Registry**, b) perform a load balancing between available nodes from a selected pool, c) execute requests passed by **Pool Registry** to **Data Service**, d) periodically poll *pool* nodes and collect performance metrics to be used in internal load balancing.

The **Pool Registry**, Figure 5, is a Web service package that keeps records of available **Data Services** and announced *pools* (local or remote). **Pool Registry** also provides an interface to submit request for user jobs. The desired **Pool Registry** typical capabilities are: a) store local *pool* announcements from various local **Data Services**, b) store remote *pool* announcements from remote **Pool Registry** instances, perform load balancing between available local and remote *pools*, d) pass user requests to least loaded *pools* found by Pool Registry Load Balancer.

There are a couple of approaches for efficient load balancing defined by:

1. Pool Registry Load Balancer, which takes care of remote vs. local pools, and

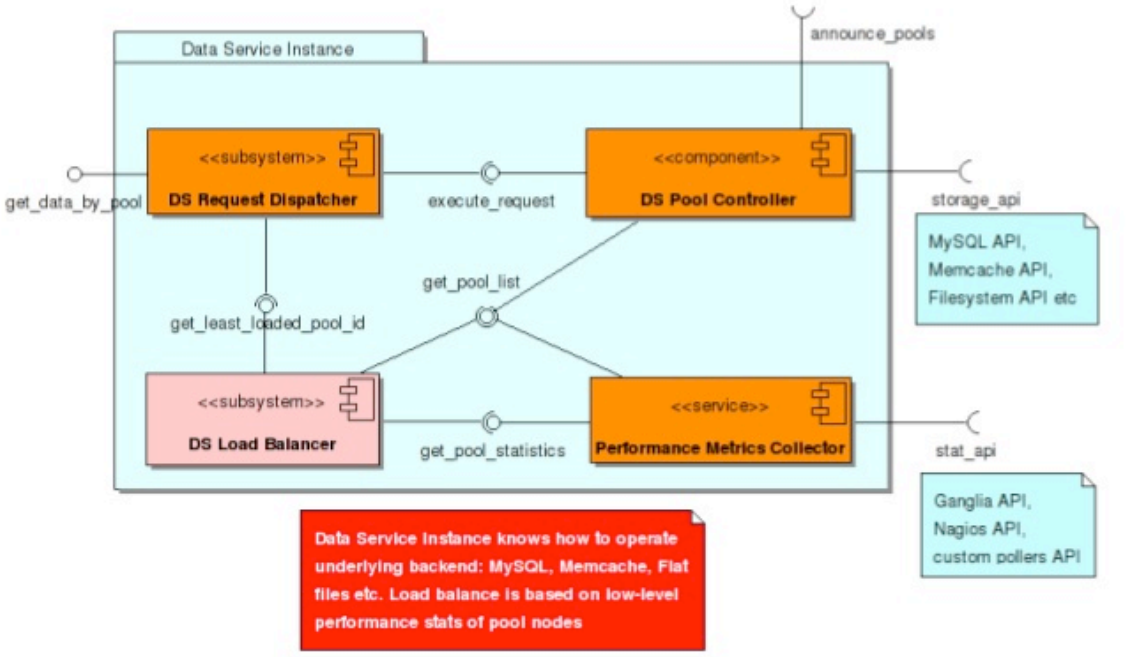


Figure 4: Data Service Instance

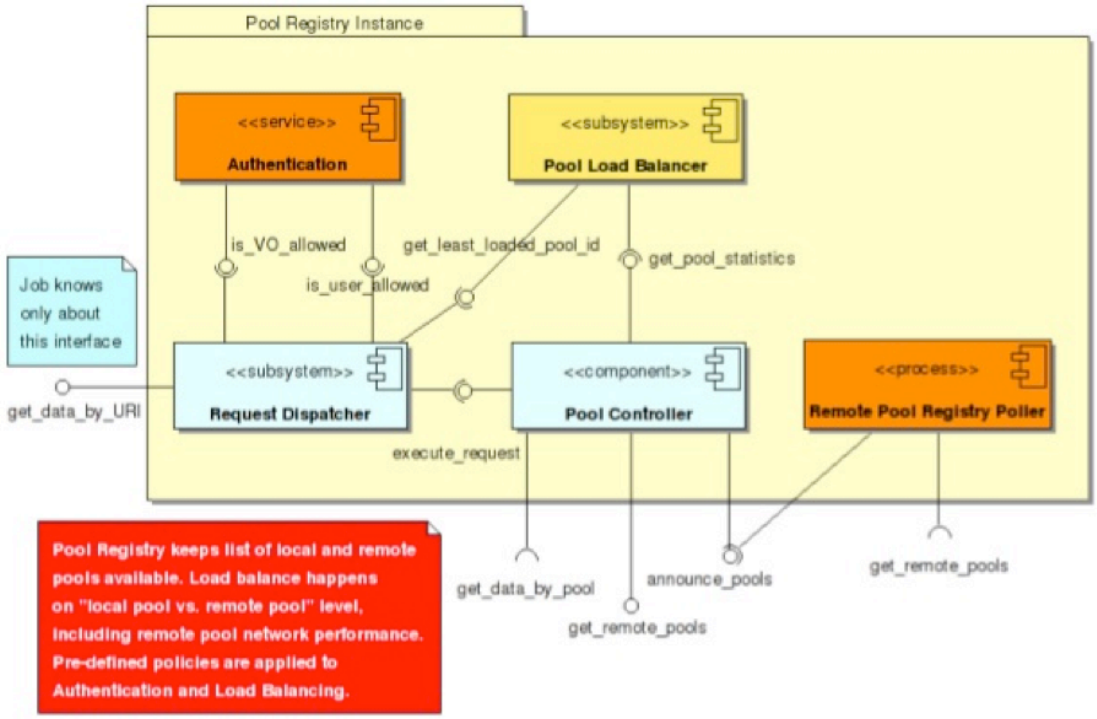


Figure 5: Pool Registry Instance

2. Data Service Load Balancer, which takes care of local node load balancing.

A thorough investigation of caching options was required and represented an integral part of the overall load-balancing scheme. An intelligent or smart cache strategy features of interest are:

1. "cache" type Data Services where "cache" Data Service pools take priority over "storage" Data Service pools in terms of Pool Registry load balancing, thus providing cache options early in the request life cycle,
2. Data Service internal caches that are specific to storage types. For example, MySQL Query Cache and Oracle In-Memory Cache. The local Data Service policy file should regulate internal cache parameters and usage.

We continued to add additional requirements as the underlying project schedule tasks progressed. In addition to the requirements identified in the Phase II project work plan and the first two quarters of project work we determined the need for:

1. Advanced indexing server for enhancing MySQL query auto-caching infrastructure.
2. High availability and fault tolerance of STAR MySQL data resources.
3. Investigation of cache control policies including explicit invalidation, time to live, and invalidation on read.
   (a) Local caches with hash table or variable:
      i. Set static variable i.e. - in php RESTful service "static $name"
      ii. If not set i.e. - in php RESTful service "if (! $name) {//fetch from database}"

The OGSA-DAI infrastructure has some significant limitations in utilizing SOAP messaging exclusively. We identified that a RESTful interface for the CWS4DB infrastructure and Data Services provided a factor of 2 faster accesses. Furthermore, optimizing the interface object definitions with JSON instead of XML provided a significant boost in performance with a significant reduction in the required network bandwidth. The STAR collaboration desired use case for "simplified read" is illustrated below:

1. User job action:
   (a) Call Pool Registry Web service get_data_by_URI method, with request URI like urn:OFFLINE_PRODUCTION: Calibrations_emc/bemcMapping
2. Pool Registry Web service actions:
   (a) Pool Registry Dispatcher:
      i. Checks user authentication via internal or external authentication infrastructure component,

ii. Parses URI and determines pool information and request parameters,
iii. Ask Pool Registry Load Balancer to determine best available pool ID for current request,
iv. Pass the request to Pool Controller, asking to execute it on a pool ID from the Load Balancer, and
v. All actions by the Pool Registry Request Dispatcher are regulated by local policies (user authentication, local vs. remote preferable dispatch, etc.).

(b) Pool Controller:
i. Contacts owner of the selected pool (Data Service Web service) and passes job request.

3. Data Service Web service actions:
(a) Data Service Request Dispatcher as Data Service Load Balancer to find least loaded node from provided pool,
(b) Data Service Load Balancer determines least loaded pool node using the information provided by the Performance Metrics Collector,
(c) Data Service Request Dispatcher passes the Job Request and Node ID to Data Service Pool Controller, and
(d) Data Service Pool Controller executes request on Data Service server node and returns the result to Data Service Request Dispatcher.

A summary of the STAR requirements for the CWS4DB Load Balancer include

Local Job Cache requirements:

1. Should be simple,
2. Fast in-memory cache,
3. Accommodate pre-identified cache-able request according to Job Execution policy.

Pool Registry Load Balancer requirements:

1. Distributed round-robin algorithm based on local and remote pool states and external policies,
2. Transparent cache capabilities, required to reduce the amount of "remote" requests,
3. Ability to intelligently re-direct failed "remote" call to next available remote pool, thus providing the required fault tolerance,
4. Ability to perform multiple parallel requests to Data Service instances, thus reducing overall wait time for multi-request calls.

Data Service Load Balancer requirements:

1. Simple round-robin algorithm for pools composed of nodes with equal processing capability (default method),

2. Weighted round-robin with response time as weight for a mixture of low and high performing server pools,
3. Weighted round-robin with limits on number of active connections for a mixture of low and high performing server pools,
4. Request dispatch, based on a data storage defined performance metrics, using real-time or near real-time node information collected by Performance Metrics Collector,
5. Utilize transparent local cache capabilities, specific to chosen Data Service data storage, to reduce "local" requests,
6. Ability to distinguish between "overhead" requests to local data storage, and data retrieval requests. For example, this feature should allow to dispatch most of the "overhead" requests to local cache pool, thus reducing data storage queries substantially,
7. Planned maintenance or outages of pools should be identified automatically, redirecting requests to backup pools according to local policy.

Architectural Design and Development

We worked with our STAR collaborators to determine architectural design that would provide the level of service required by the STAR infrastructure at our February 2010 face-to-face meeting. The original design of the new infrastructure was limited by the requirement of not changing any of the STAR production code base, however during this meeting our collaborators pointed out that they were redesigning the production code base and could allow changes to occur. Drs. Lauret and Green seized this opportunity to make the collaborative project infrastructure much more valuable to the STAR collaboration by including RESTful interfaces into the STAR code redesign. We believe that the multi level caching infrastructure provided by Tech-X and STAR is quite impressive and will provide a great deal of value to the STAR collaboration. These well-defined interfaces have proven to be pivotal in providing a more robust environment for the STAR production jobs database queries. We have followed the architecture shown in Figures 6 - 10 to develop our service components:

Project Task Tracking and Management

We used several tools for tracking this long running task that are highlighted below:

dotProject: dotProject is an open-source, multi-user project management tool that helps us organize the projects, tasks, companies, departments, events, contacts, to-do lists, resources, and generate gantt charts showing our progress. We used this tool to manage assigned personnel, resources, task logging, and our *Work Breakdown Structure (WBS)* for this effort. This WBS and the gantt chart summarizing our performance in the beginning of this document were produced using our dotProject instance.

Redmine Project Management: Redmine is a flexible project management web application design of which is influenced by *'Trac'* and developed using *'Ruby on Rails'* framework and it is cross-platform and cross-database. The notable features of Redmine are multiple projects support, flexible role based access control, issue tracking system, gantt chart and calendar, documents and files management, feeds and email notifications, per project wiki,

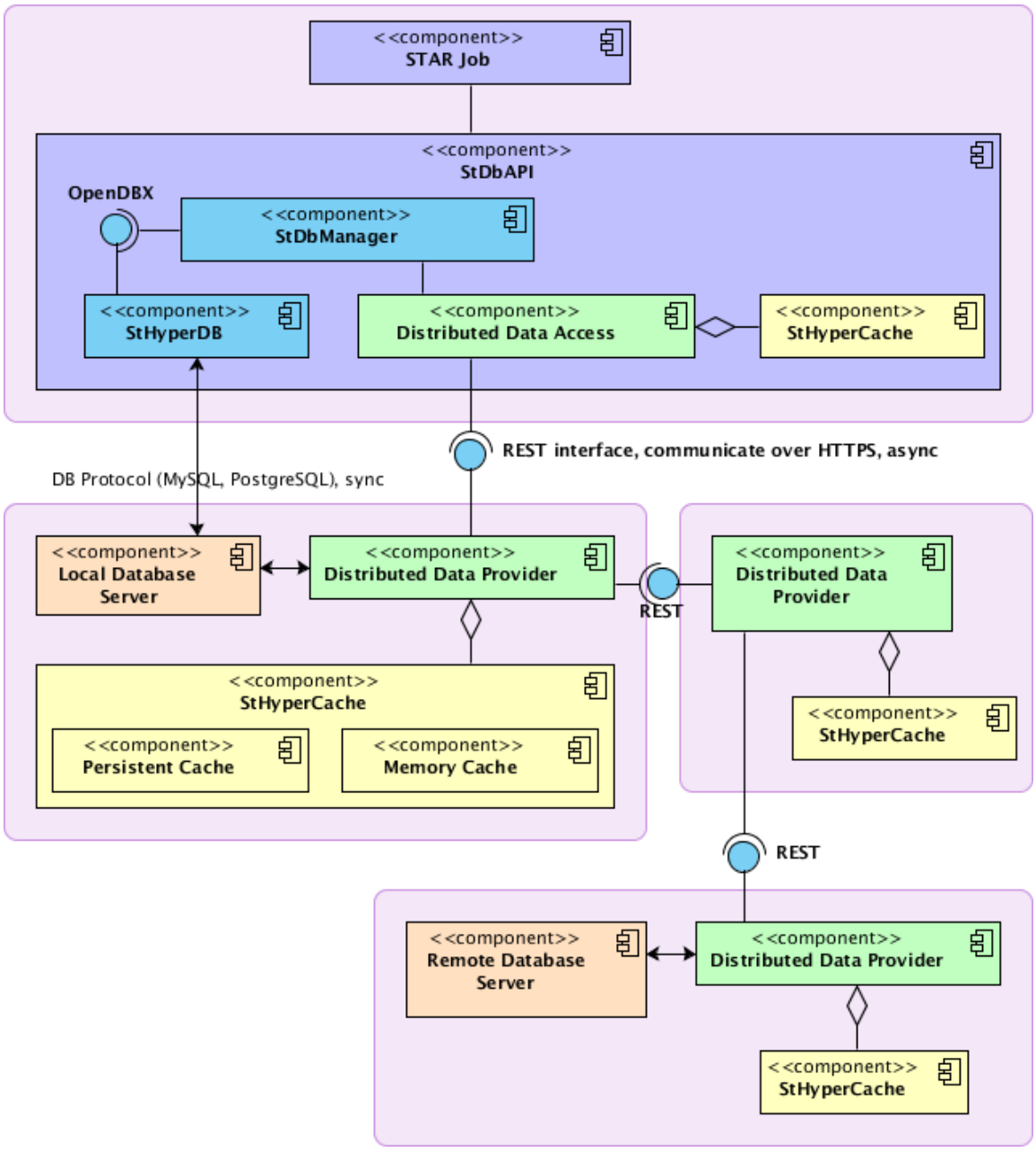


Figure 6: CWS4DB Architecture Diagram

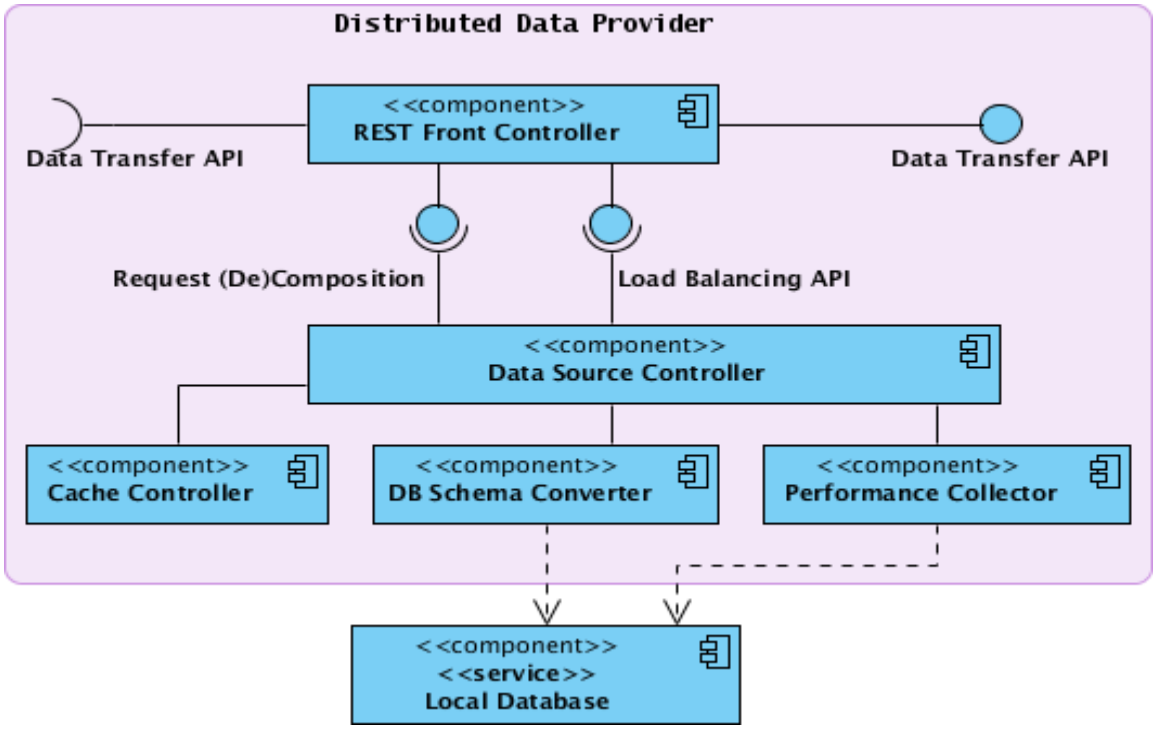


Figure 7: CWS4DB Architecture Distributed Data Provider Diagram

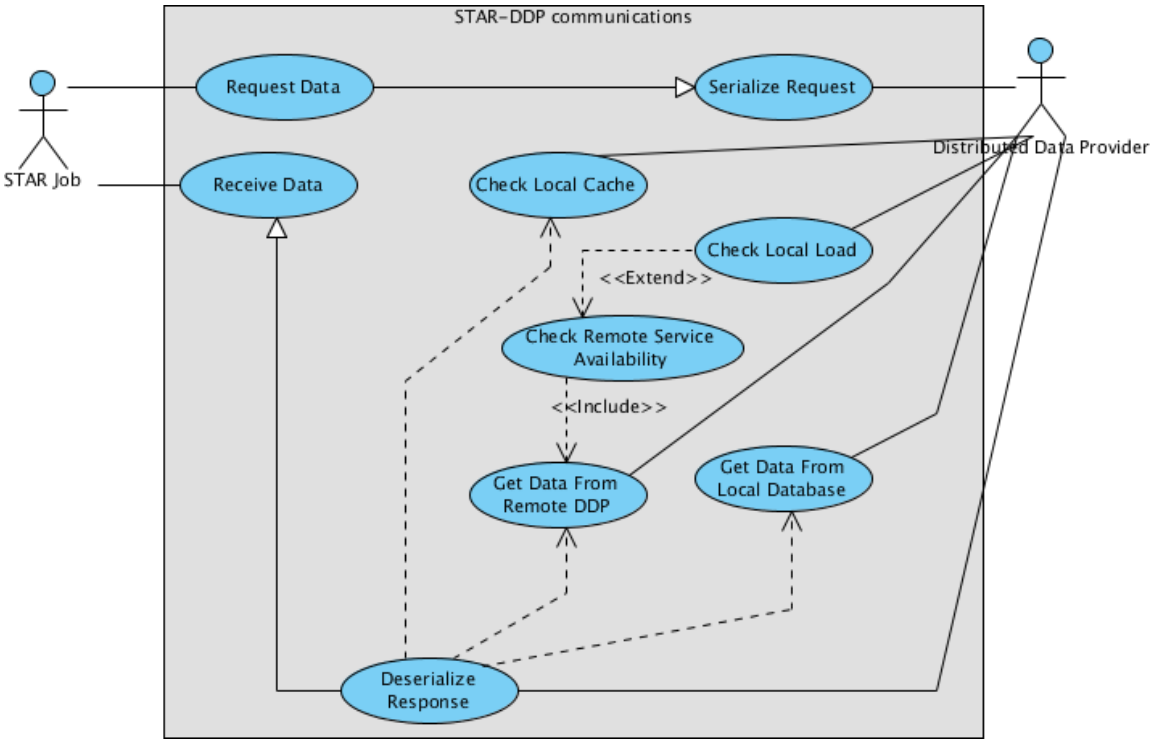


Figure 8: CWS4DB Architecture Star DDP Communications Diagram

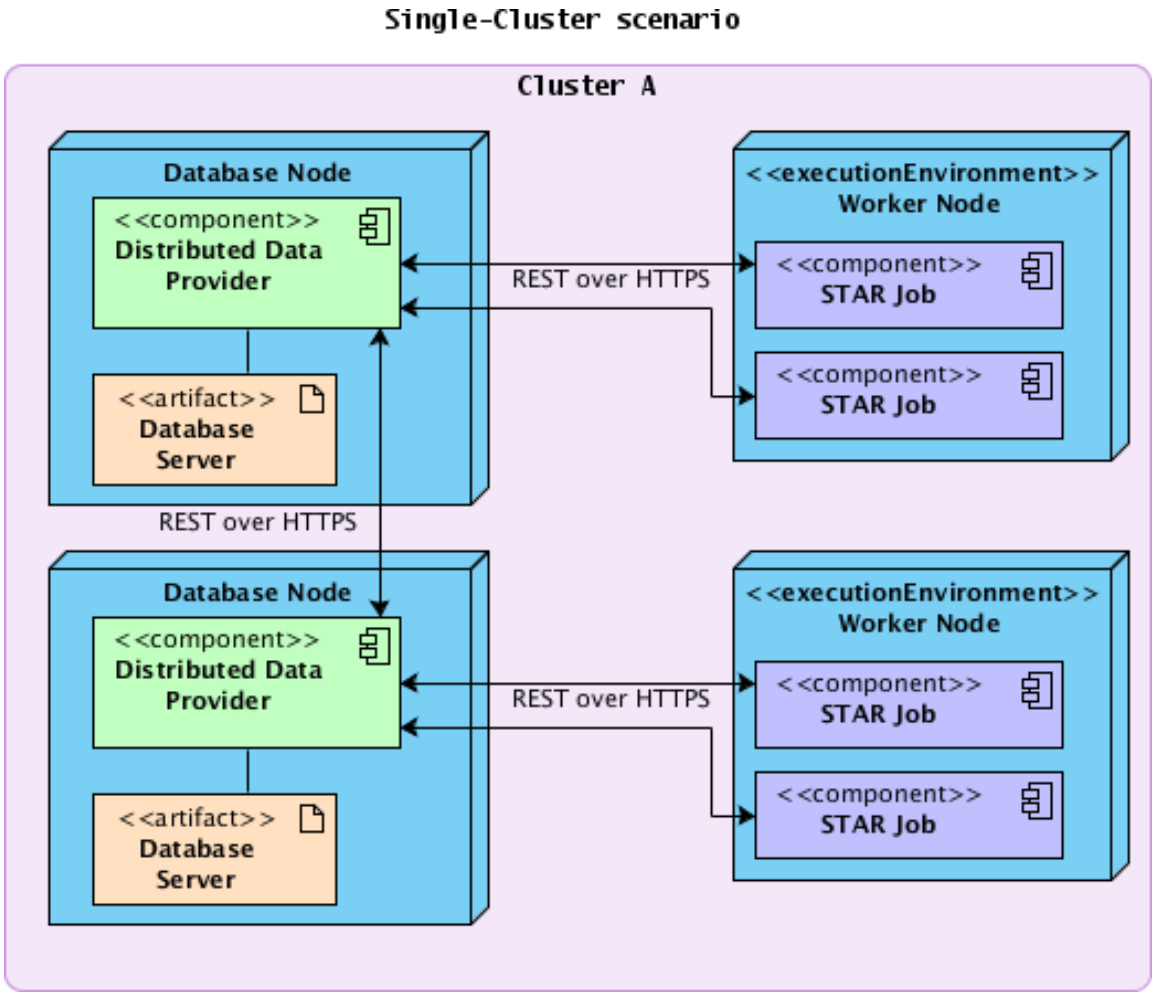


Figure 9: CWS4DB Architecture Single Cluster Scenario Diagram

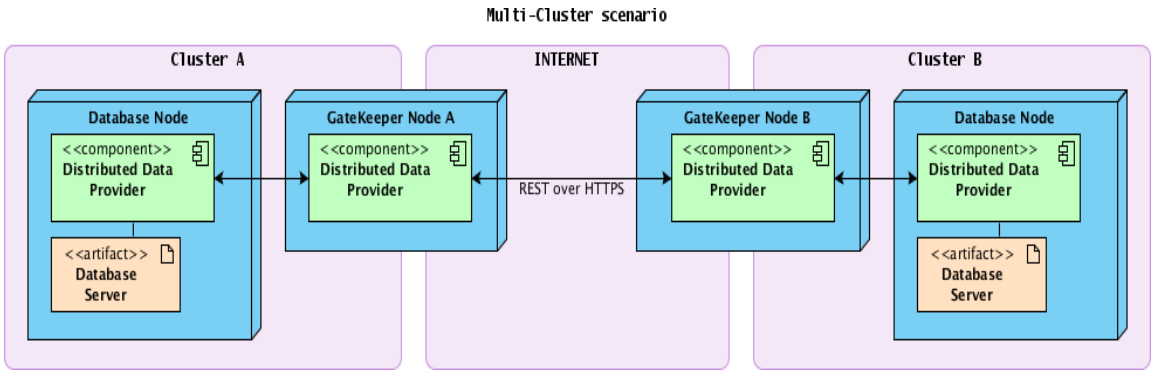


Figure 10: CWS4DB Architecture Multi Cluster Scenario Diagram

per project forums, time tracking by providing an option to update time as one works on the task, custom fields for issues, time-entries, projects and users, integration with SVN, issue creation via email, multiple LDAP authentication support, and multiple databases support.

Over the course of this effort we phased in Redmine as another tool for managing this project. This tool, complete with its issue tracking and Wiki documentation engine have allowed us to create and maintain a rich interface for managing the resources and schedules associated with CWS4DB. An example of the Wiki documentation we have built up over the course of this effort is included in Figure 11.

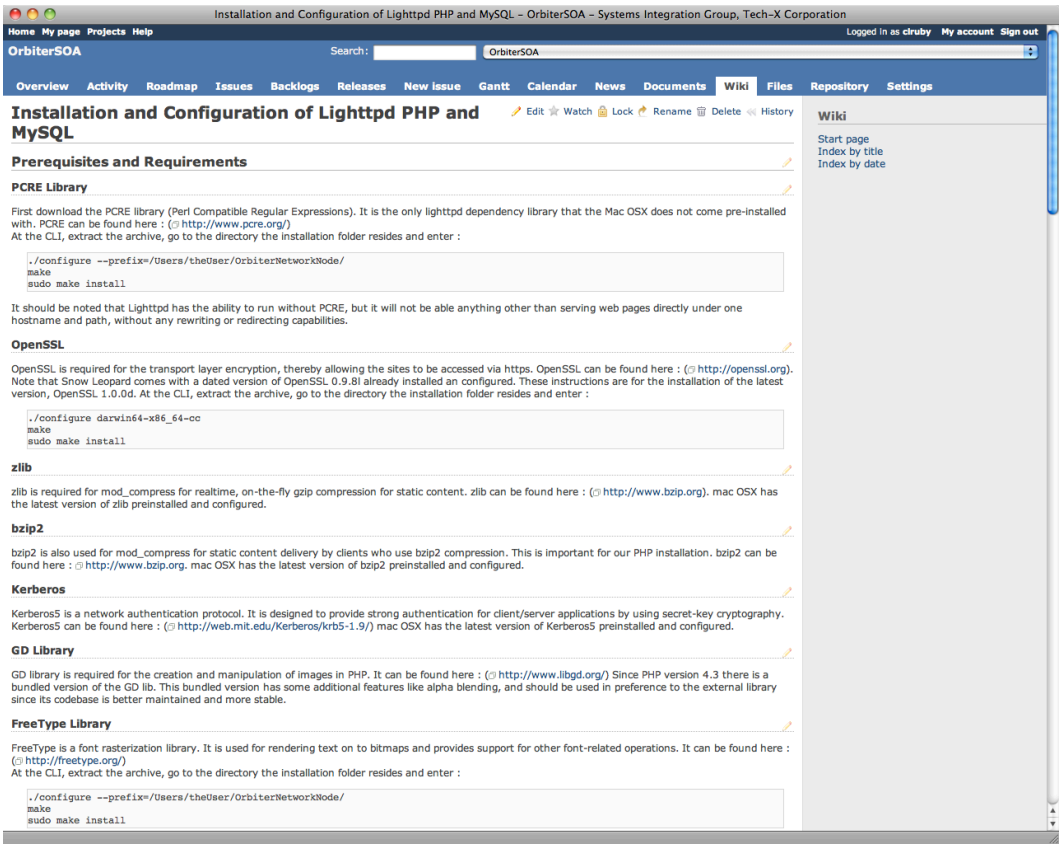


Figure 11: Redmine Project Management Wiki Documentation Interface

Tech-X Knowledge Base: KnowledgeBase Manager Pro is used to for sharing information about organization or business unit, troubleshooting information, articles, white papers, user manuals, or answers to frequently asked questions as shown in figure 12

Tech-X Mac A&D: MacA&D is a comprehensive tool for system modeling and simulation, requirements management, structured analysis and design, object-oriented modeling with UML and data modeling of information systems. It has diagram editors for process models, data models, class models, state models, object models, structure models and task models. Each model shows a different view of the software system integrated through a global data dictionary as shown in figure 13, figure 14.

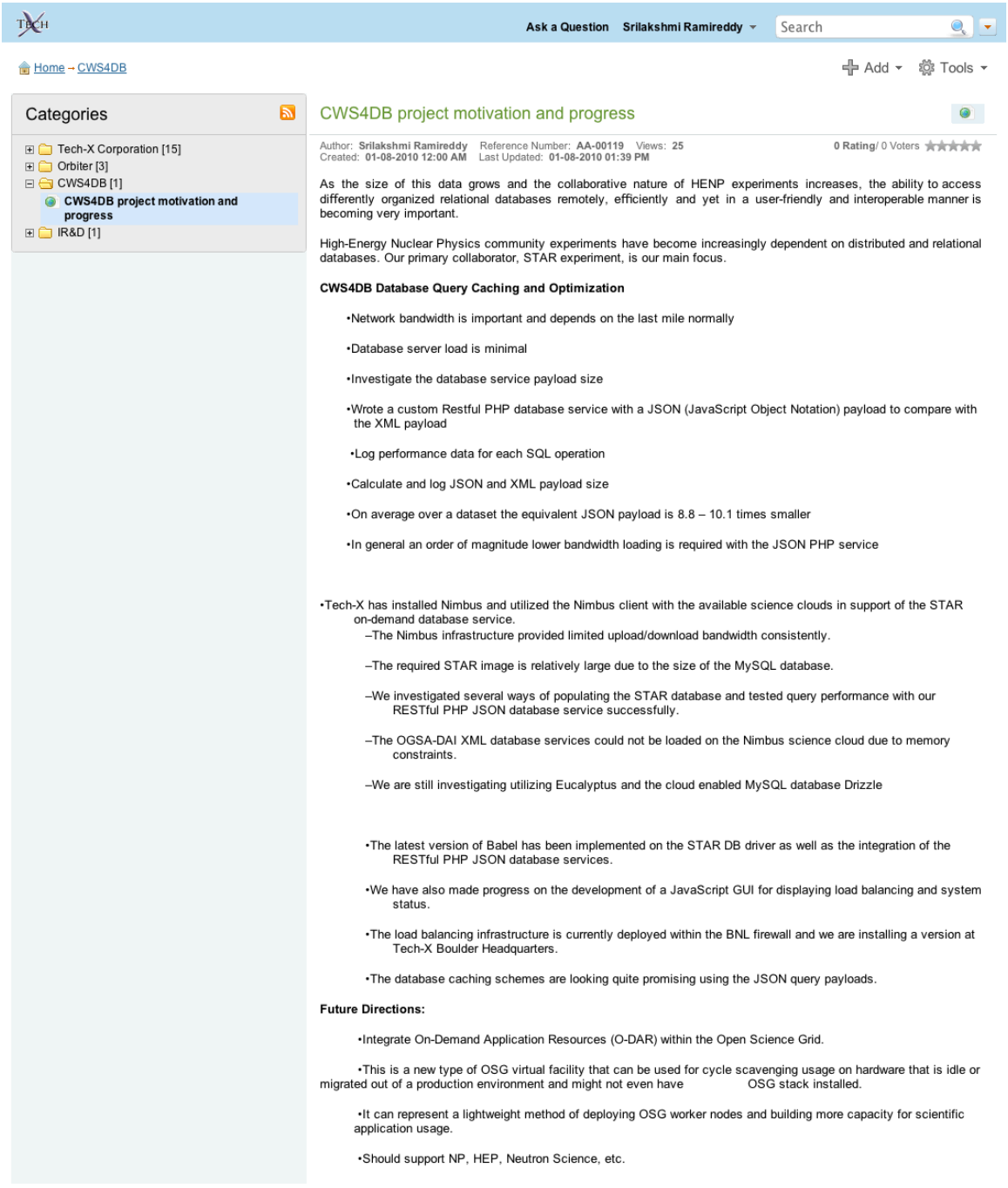


Figure 12: Knowledgebase CWS4DB

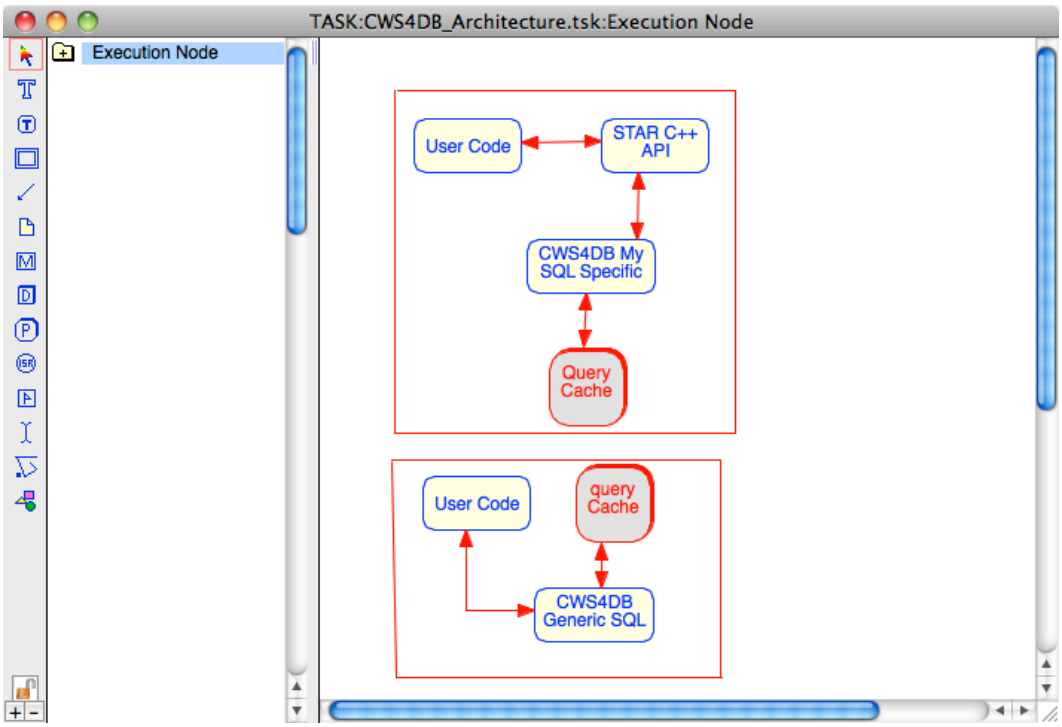


Figure 13: CWS4DB Execution Node

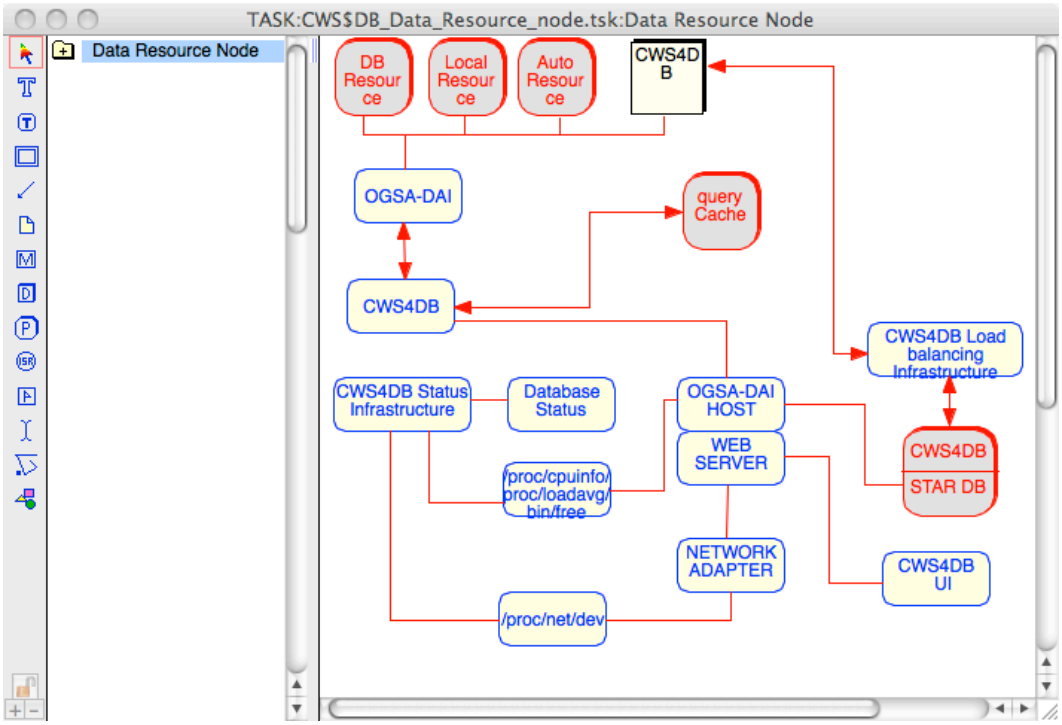


Figure 14: CWS4DB Data Resource Node

2.2 Task 2. Design and Implement Tiered Deployment Capabilities

Objective: Develop a tiered deployment based protocol for the CWS4DB system. The potential STAR collaboration resource providers will have various and potentially limited infrastructure and/or network connectivity. Thus, a tiered deployment protocol is necessary for a successful CWS4DB system deployment on these resources.

Summary:

a) Develop the tiered deployment based on the possibility of remote servers that may have limited internet bandwidth and/or high latency issues
b) Specify and implement this tiered deployment protocol for user-friendly configuration of the CWS4DB system
c) Provide the necessary installation configuration tools required for deployment and subsequent system validation and verification provided by the customizable site specific test suite

We have met our goals for this task in developing and implementing RESTful web services and user interfaces in support of the tiered deployment of the CWS4DB system. We discuss our individual accomplishments in the following subsections.

Develop a Tiered Deployment of the CWS4DB System

The Phase I work was performed on the Tech-X server cyber.txcorp.com. We have since repurposed this server and configured a new main server (orbiter.txcorp.com) for the CWS4DB development and testing. This required the migration of the original Phase I CWS4DB infrastructure along with new versions of Globus, Tomcat, OGSA-DAI, PHP, and Apache Web Server.

In order to provide a secure and robust build and test environment we installed a commercial Thawte SSL certificate for the Apache Web Server. This provides the transportlevel encryption that our CWS4DB Web service infrastructure uses for security. These two servers are configured such that the testing of our tiered deployment infrastructure and protocol can be tested.

All of our web services will support these four levels of deployment and cyber.txcorp.com is configured to emulate our four CWS4DB tiers:

Level 1: CWS4DB Base Execution Node provides the STAR C++ API CWS4DB library bindings with a CWS4DB public data resource node configuration.

Level 2: CWS4DB Private Static Data Resource Node provides a static local configured private data resource node.

Level 3: CWS4DB Private Dynamic Data Resource Node provides a dynamically configured private data resource utilizing local and remote data resources.

Level 4: CWS4DB Public Dynamic Data Resource Node provides the complete public dynamic system utilizing local, remote, and auto configured data resources and the CWS4DB status infrastructure.

The testing of all levels have been performed on the cyber.txcorp.com utilizing a locally configured 8 node simulated cluster resource provided by this multi-core server. The Level 1 deployment protocol utilizes a local MySQL database presented as a RESTful data resource. The Level 2 deployment protocol utilizes the Level 1 deployment but utilizes a remote MySQL database served by the orbiter.txcorp.com server and presented as a RESTful Data Resource. Levels 3 and 4 use a combination of 'cyber and orbiter' server cores for testing the STAR data resources.

Eucalyptus and OpenNebula

In conjunction with this task we have investigated a variety of cloud computing tools. That is, though Eucalyptus has many features for developing and maintaining cloud infrastructure, we found that it is not feasible to implement and use this infrastructure for CWS4DB development because several of its critical features are not open-source. After investigating other cloud infrastructure technologies we have settled on OpenNebula, an open-source project that has been adopted by several big players in the cloud and grid computing community.

We have setup our physical infrastructure to have a cluster-like architecture with a front-end, and a set of cluster nodes where a physical network joining all the cluster nodes with the front-end. We have installed OpenNebula on the front end and ran node scripts on all the nodes where Virtual Machines will be executed. We have developed the service architecture for different nodes and tested with all 4 levels of deployment protocols using the cloud's front-end node infrastructure setup. We have hosted our services and data resources at different nodes on the cloud environment for resource distribution. More explanation of implementation and testing can be found in Task 5 accomplishments.

Orbiter Service Oriented Architecture

Over the course of this effort we have enabled several CWS4DB capabilities as web services through the Orbiter Service Oriented Architecture (SOA). This system expresses a software architecture that capitalizes on use of loosely coupled software services to support the requirements of the business processes and software users. In a SOA environment resources on a network are made available as independent services that can be accessed without knowledge of their underlying platform implementation. The software components are reusable because the interfaces are standards-based and they are independent of the underlying implementation.

Web Services (WS) are open standards-based, modular, distributed, dynamic web applications that are self-described, published, located, or invoked over internet protocols. Portals

and Gateways use collections of WS that perform simple or complex tasks and they might be used to create more complicated applications. WS normally use the standard client-server model where the client sends a XML query to the server and the server responds with a XML result. Web Services Definition Language (WSDL) is a special XML grammar created for self-describing web services. A WS has a WSDL file that defines the communication protocol, the available methods, data transfer syntax, and the location of the service endpoints.

It was determined as a result of our work in task 1 that a RESTful web service interface would provide the most flexible, robust and efficient access to CWS4DB resources. To this end we have developed a number of RESTful Orbiter web services, featured in Table 1 facilitating the work on the rest of these tasks, in addition to supporting the Commander thick-client interface and site-specific testing.

Table 1: CWS4DB RESTful Web Services

Services	Description
OrbiterConnectivityService	Provides an endpoint for testing and verifying connectivity.
OrbiterErrorHandlerMessageService	A service for creating and managing other web service interface errors
OrbiterFederationExplorerService	Provides a listing of services, by node and package
OrbiterMasterSlaveDatabaseValidationService	Compares the Orbiter Master database tables with the Slave database tables, can replace/repair as well
OrbiterNoopService	An "emtpy" service for Quality of Service testing, which does full authentication verification and database connection without querying or returning any information
OrbiterQueryDbConnectionStringService	A service interface for managing database connection string information
OrbiterQueryDbLoadBalancerService	Provides load balancing support including updating database rank and status
OrbiterQueryService	A service interface for running queries against a particular database. Supports query caching.
OrbiterResourcePreCacheService	Provides the ability to pre-cache Query Db sql queries when a new database resource was added to resource table
OrbiterSimulatorService	Simulates running several queries against a particular database. Collects timing and performance information.
OrbiterVersionInformationService	Provides information on web service family versions

Orbiter services were designed in an object-oriented manner, where common functionality was built in reusable components that allowed us to rapidly prepare web service-enabled functionality for deployment. A common Orbiter Service component was developed that encapsulates much of the lifecycle and infrastructure associated with an Orbiter service implementation, enabling the more streamlined development of new Orbiter services. This component allows services to define their properties and business logic in a more straightforward manner, allowing the Orbiter services to be more maintainable and stable as the infrastructure continues to grow.

Orbiter Services return a WSDL (Web Service Description Language) description of their interfaces as well as an XSD (XML Schema Definition) for its defined service attributes and their values. The service WSDL and XSD provide a regular description of the operations and parameters accepted by an Orbiter service and also permit client code to be automatically generated that will comply with the service's expected requests. The Orbiter Federation Explorer, a Java Thick Client Commander module that is discussed in further detail later in this section, both displays the WSDL and XSD and also uses them to construct customized test cases for every service. The Orbiter Federation Explorer dynamically constructs testing Graphical User Interfaces from these API descriptions, allowing users to systematically run Orbiter Services against their input. A screenshot of this capability is shown in Figure 15.

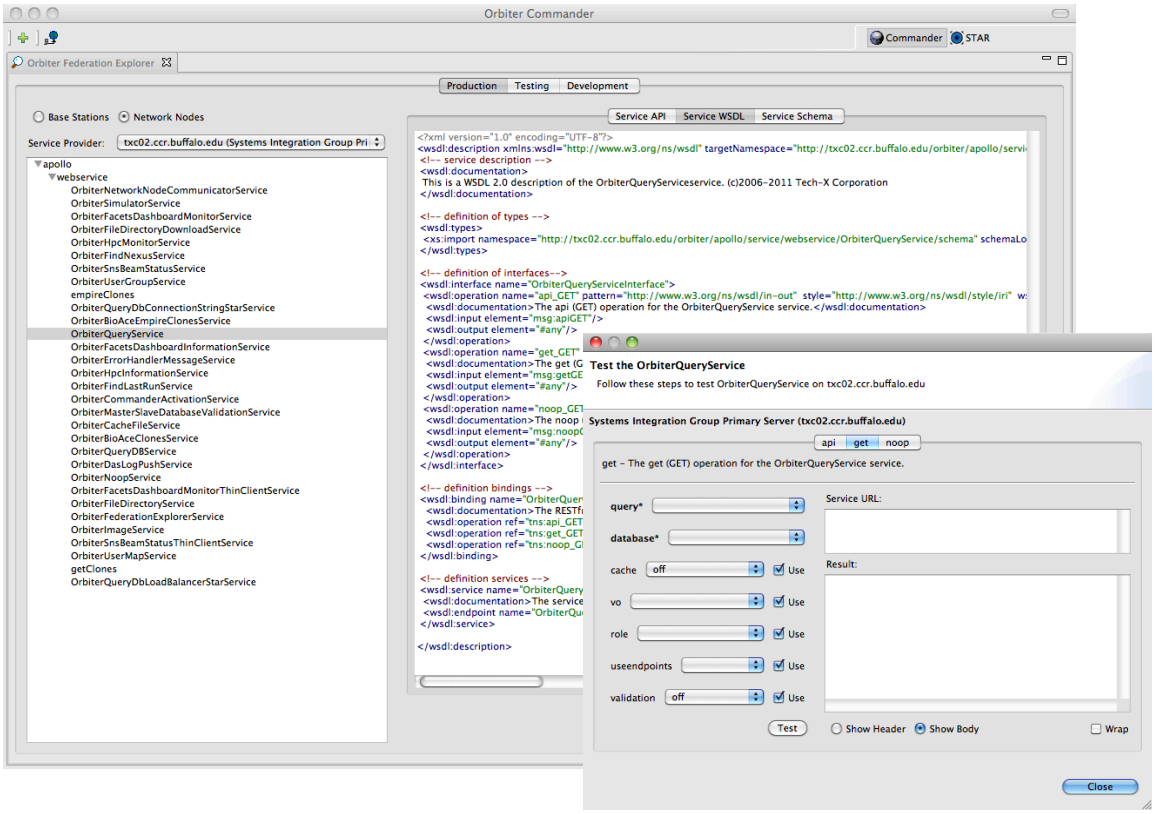


Figure 15: Commander Federation Explorer for the Orbiter CWS4DB Query Service

Orbiter SOA development was performed within the context of a major and minor versioning system, where services released for a specific major version are maintained in minor version revisions that are abstracted for service users. *Apollo*, Orbiter's first and current major version, can be invoked by using *apollo* in the service URI, where this will always resolve to the latest minor version deployed on the host. By using this system the Orbiter service development team can continuously release improvements to deployed services without affecting Orbiter service users. Orbiter *Taurus* and *Orion* were also released over the course of this effort.

Commander Application

In conjunction with this task we have investigated, designed and developed Orbiter Commander, a thick-client desktop application designed to facilitate graphical user interaction with the Orbiter SOA infrastructure developed over the course of this effort. Commander is being developed using the Eclipse Integrated Development Environment (IDE), which provides streamlined compile/launch features as well as integration with Subversion, a revision control system that preserves the integrity and history of developed code. Eclipse's API for its Rich Client Platform (RCP) allows Graphical User Interfaces (GUIs) to be rapidly developed using several constructs, including well-defined *extension points* for defining custom extensions to its capabilities. Through Eclipse RCP Orbiter Commander is able to provide a sophisticated interface complete with dock-able windows, pre-defined preference pages, an integrated help system, and complete branding and licensing information.

We prepared both a paper and a talk on Orbiter Commander that highlights its flexible and extensible design and its seamless integration with Orbiter web services. This peer-reviewed paper was accepted by the Gateway Computing Environments 2010 (GCE10) workshop and a talk was given at the workshop at the Supercomputing Conference (SC10) on November 14, 2010 in New Orleans, Louisiana. The paper and talk are attached later in this document.

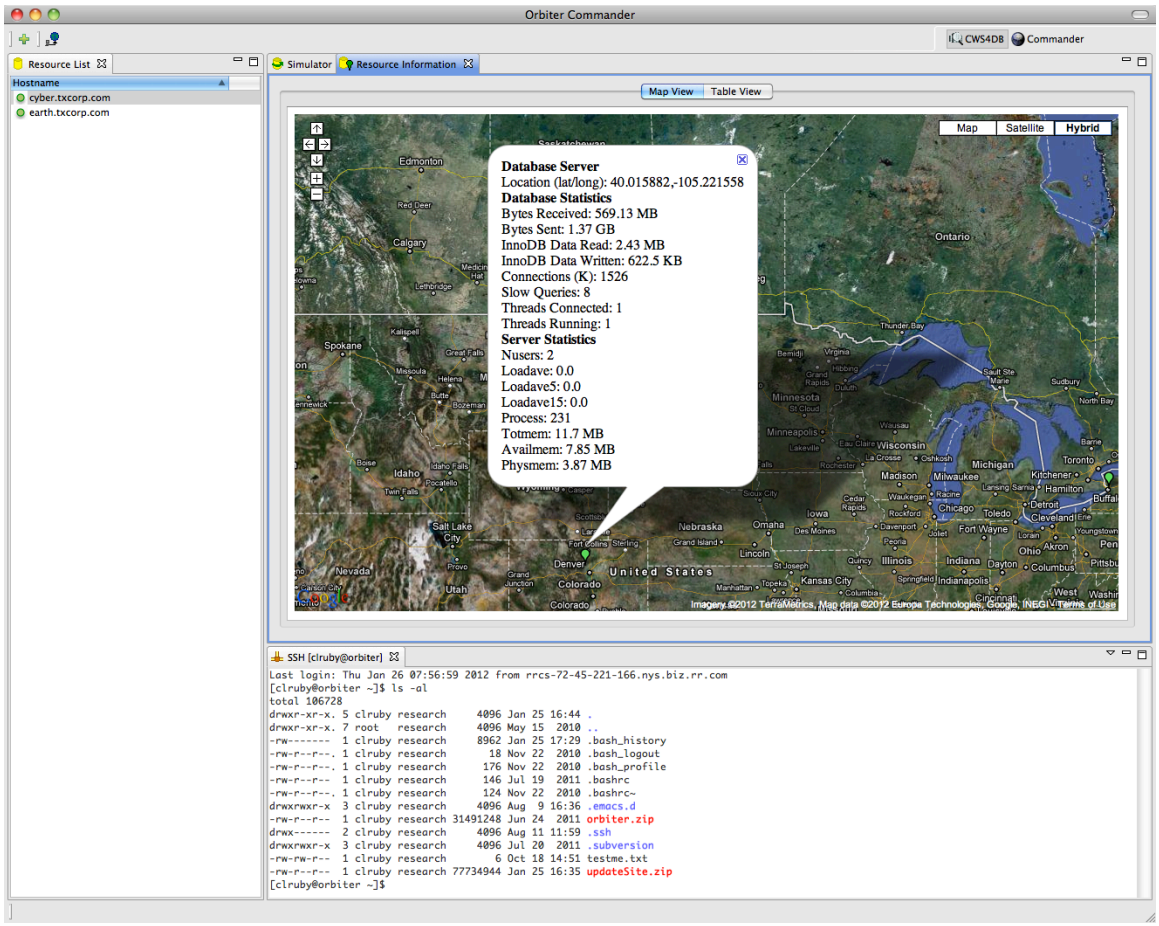
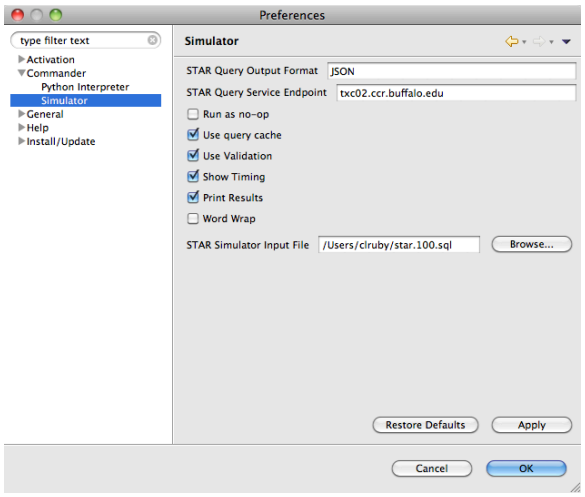
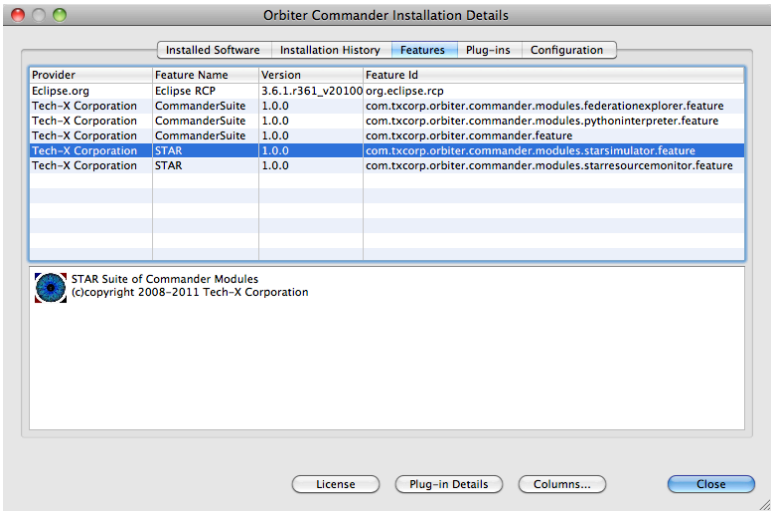


Figure 16: Orbiter Commander CWS4DB Suite of Modules

Commander's functional capabilities are implemented as independent and self-contained components called *modules*, which are organized in *suites* of broader categories. Users can use multiple modules/suites at one time, giving them a rich experience in a fully customizable desktop interface. A screenshot of the CWS4DB Orbiter Commander suite is shown in Figure 16. Eclipse RCP also automatically enables user-configurable preference pages and about/licensing information that improves the overall experience for Commander users, shown in Figure 17.



(a) User-Configurable Preference Pages



(b) About and Licensing Information

Figure 17: Preferences and Licensing information

Commander is capable of utilizing Orbiter RESTful services from an arbitrary remote location, utilizing full authentication and validation for any of the STAR services provided in this project. Interaction with Orbiter RESTful services is enabled through an *OrbiterRESTClient*, which was developed as a part of a *Connectivity* component for Orbiter Commander. This client code handles web service parameters and authentication in a simple and easy-

to-use set of classes. The architecture for Commander is illustrated in Figure 18, where the Commander framework and its individual modules communicate with the Orbiter web services to gain access to their capabilities and to fuse their information into rich user interfaces.

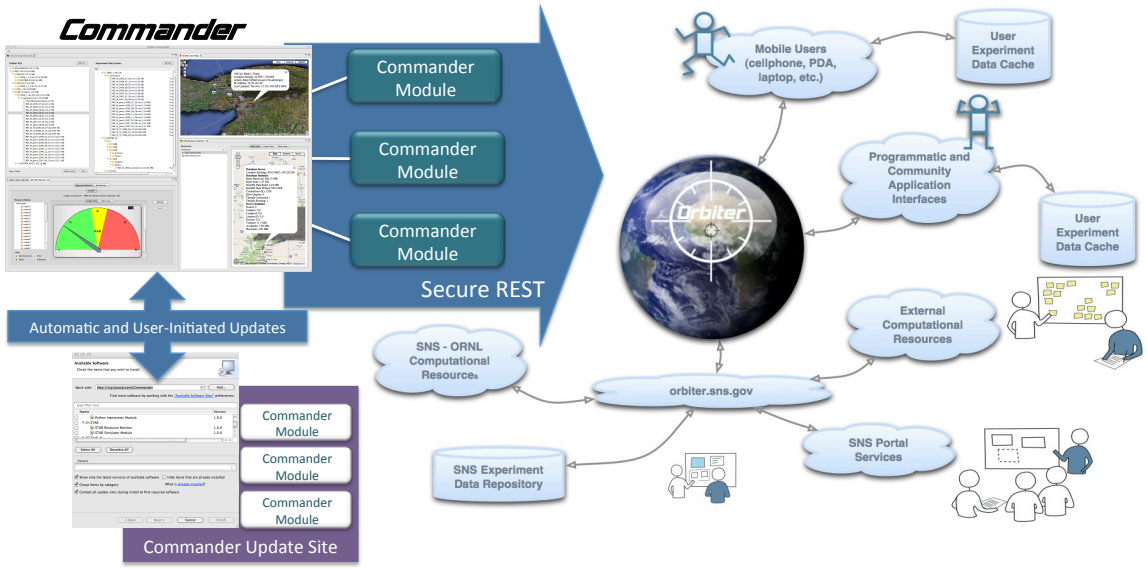
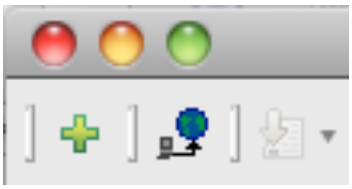


Figure 18: Commander Architecture with Orbiter Web Services



(a) Successful Connection

(b) Unsuccessful Connection

Figure 19: Commander Connectivity Indicator/Tester

Commander includes an indicator in its main toolbar that allows users to see and test Commander's connectivity with the deployed Orbiter infrastructure. This button, shown in Figures 19a and 19b, will indicate whether Commander is able to connect to the Orbiter connectivity service deployed at the Spallation Neutron Source. A successful connection indicates that Commander is able to interact with the other deployed services. An unsuccessful connection, shown with a red 'X', indicates that Commander cannot connect to Orbiter, which could be due to network issues on either end. Commander will periodically retry this connection, and clicking this button will re-try this test and show the current status.

Commander also enables fully configurable HTTP and HTTPS proxy settings, allowing Commander to interact with the Orbiter SOA through a proxy instead of directly through an internet connection. Proxy settings can be configured via Commander's preference pages, and will persist in future sessions.

Commander provides interfaces for configuring its individual service endpoints. This advanced capability allows experienced users to alter the hostname, service package, and service version for a particular service call made within Commander, facilitating troubleshooting or manual fail-over in the event that the given host cannot be reached. This interface, shown in Figure 20, also permits configured service endpoints to be tested individually, which can

be used in troubleshooting a capability in an individual module.

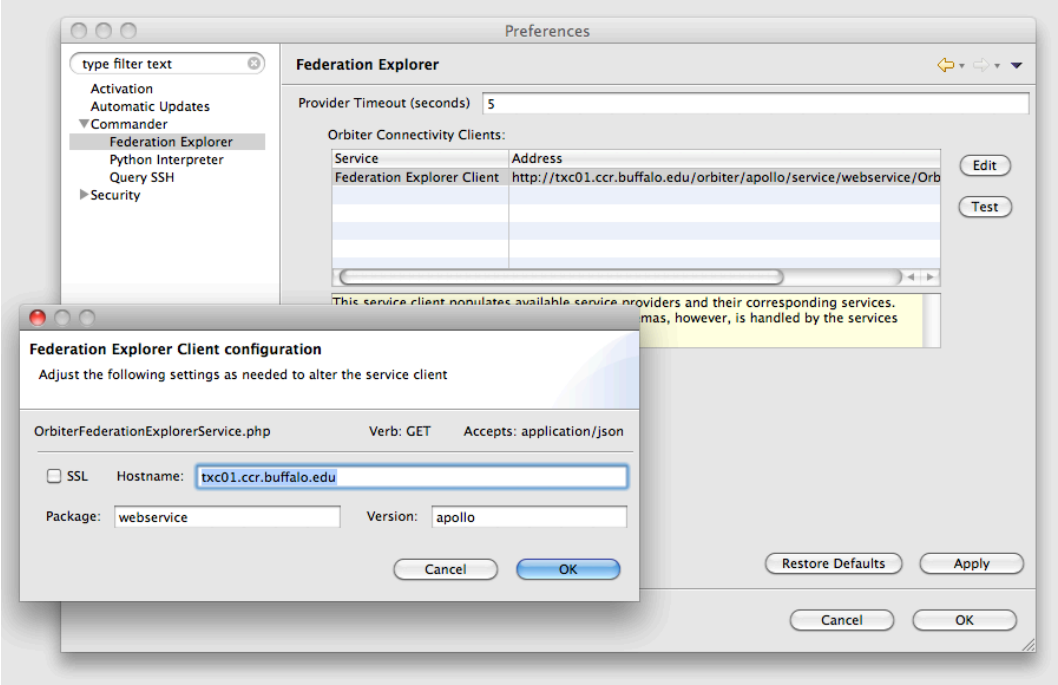


Figure 20: Commander Orbiter Service Client Configuration

Commander takes advantage of the *perspectives* extension point within the Eclipse Rich Client Platform, providing configurable view orientations that allow users to better organize their modules for their purposes. Module views automatically appear in an Eclipse *perspective* corresponding to the application *suite* that they belong to. Users can click to switch between these suites, preventing view "crowding" and automatically organizing modules according to their functional capabilities. Module views are programmed with a default orientation that we believe most users would favor, however each view is fully dock-able to allow users to completely tailor Commander to their needs. Commander uses multiple threads to start modules such that the application can still be used while another module is loading.

We have also implemented a mechanism for communication between running modules within the application. Though we have used an approach that allows modules to be developed completely independently, we have enabled interaction through a loosely-coupled communication framework. This allows modules to not only share functionality without code duplication, it allows certain Commander features to be programmatically accessed, opening the door for future interaction through third-party tools, interfaces, and scripts.

Commander is designed using industry-accepted principles of object-orientation and encapsulation, ensuring that the application will remain scalable and maintainable over a long period of time. We have also defined custom Eclipse *extension points* for Commander which greatly simplifies the process of creating and maintaining new Commander modules. That

is, the Eclipse IDE will automatically generate Commander-compliant code based on the extension points we've defined for the application. An example of the Commander module extension, from which code can be generated, is shown in Figure 21.

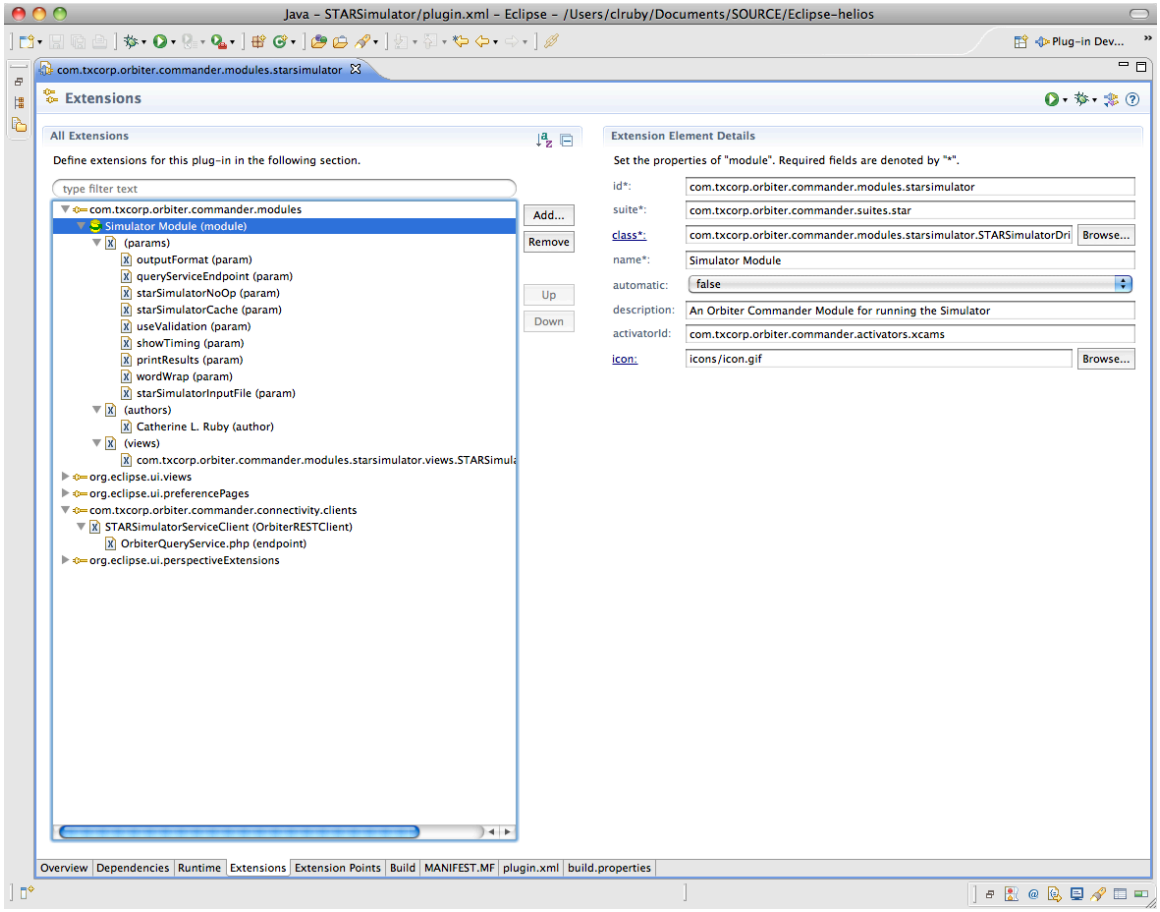


Figure 21: Commander Extension Point for the Query Simulator Module

Over the course of this effort we have developed the following Eclipse RCP extension points, which provided a rich and comprehensive development environment for Commander capabilities.

Commander Modules: an extension point for generating a Commander-compliant module

Commander Suites: an extension point for generating a Commander-compliant module suite

Connectivity Activators: an extension point for generating a Commander-compliant activation workflow

Connectivity Clients: an extension point for generating a Commander-compliant Orbiter web service client

Orbiter Commander consists of 503 Java source files, totaling nearly 80,000 lines of code/comments.

It has currently been tested on and can be deployed to the following systems and architectures:

- Mac OS X (cocoa/x86) 32-bit - tested with Snow Leopard (10.6)
- Mac OS X (cocoa/x86) 64-bit - tested with Snow Leopard (10.6)
- Windows (win32/x86) 32-bit - tested with Windows 7
- Windows (win32/x86) 64-bit - tested with Windows 7
- Linux (gtk/x86) 32-bit - tested with Fedora 13
- Linux (gtk/x86) 64-bit - tested with Scientific Linux 5

The Eclipse IDE and its exported RCP applications are supported by the Eclipse Equinox p2 provisioning system, an OSGi implementation. An RCP application, broken into several supporting JAR files, can be individually and incrementally updated from a central application repository, allowing the user to either automatically or manually retrieve the latest releases for their application instance. Orbiter Commander was designed to allow its individual modules to be updated by the user using this system, and we have enabled this Eclipse feature and built a repository site for Commander modules. This mechanism allows users to easily and seamlessly retrieve the latest releases of Commander capabilities or to integrate new capabilities into their installation using a simple point-and-click interface (shown in Figure 22).

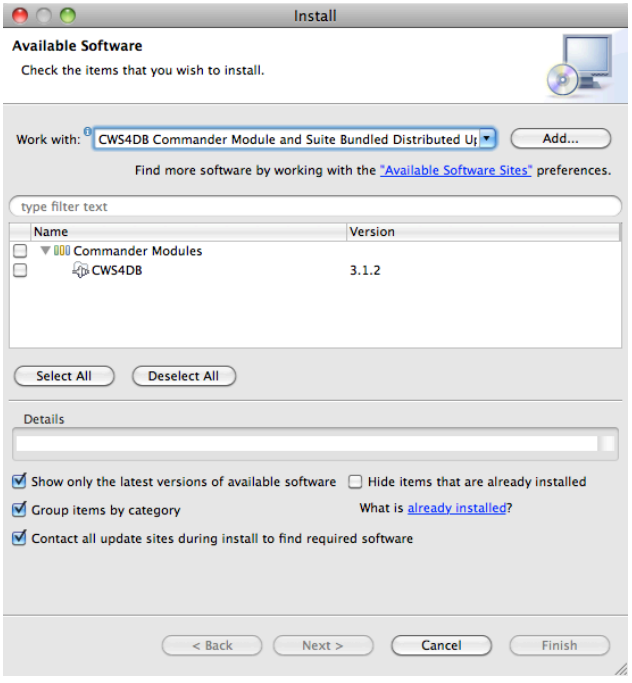


Figure 22: Orbiter Commander Update Site

For this effort we have developed several modules that enable the development and testing of capabilities for this project. Users can invoke these capabilities using a simple point-and-click interface. These modules include:

Query Resource Monitor: An Orbiter Commander Module for viewing the status of query resources

Query Simulator: An Orbiter Commander Module for running the CWS4DB Query Simulator

Query SSH: An Orbiter Commander Module enabling an interactive SSH terminal to remote resources

Python Interpreter Module: An Orbiter Commander Module for executing Python commands

Orbiter Federation Explorer: An Orbiter Commander Module for browsing and testing Orbiter web services

Further detail on these modules is included in the following subsections.

STAR Resource Monitor

The Orbiter Commander Resource Monitor allows users to browse CWS4DB query resources and determine their running status. Query resources are placed on an interactive Google map, where statistics including bytes sent and received, thread counts, load averages and memory utilization are displayed according to the resources' geographic locations. Alternative views show this information in a table or graph form. The host list on the left side of the module displays the STAR resources, where clicking on one centers the statistics tab on the particular resource of interest.

This module facilitates the streamlined use of the other modules in this CWS4DB suite by enabling users to perform actions on a selected query resource. Users may opt to run a query against a resource, run a full query application simulation against a resource, or open an SSH terminal window and connection to a resource, simply by right-clicking on a resource in the list provided. A screenshot of this module is shown in Figure 23.

Query Simulator

The Orbiter Commander Query Simulator Module allows users to interact with the Orbiter query service (discussed in more detail under Task 3). Using this module users may run a STAR input file against any host, specifying the output format, whether or not to perform the query (or a no-op operation), use the query cache, or to use validation, allowing this service to be tested wherever it is deployed. Service round-trip timings can be displayed to benchmark the services between resources as well, providing a powerful tool for interacting with the query service. This module also enables interaction with the simulator service, where multiple queries can be run against the tiered deployment infrastructure for testing and benchmarking. A screenshot of this module is shown in Figure 24.

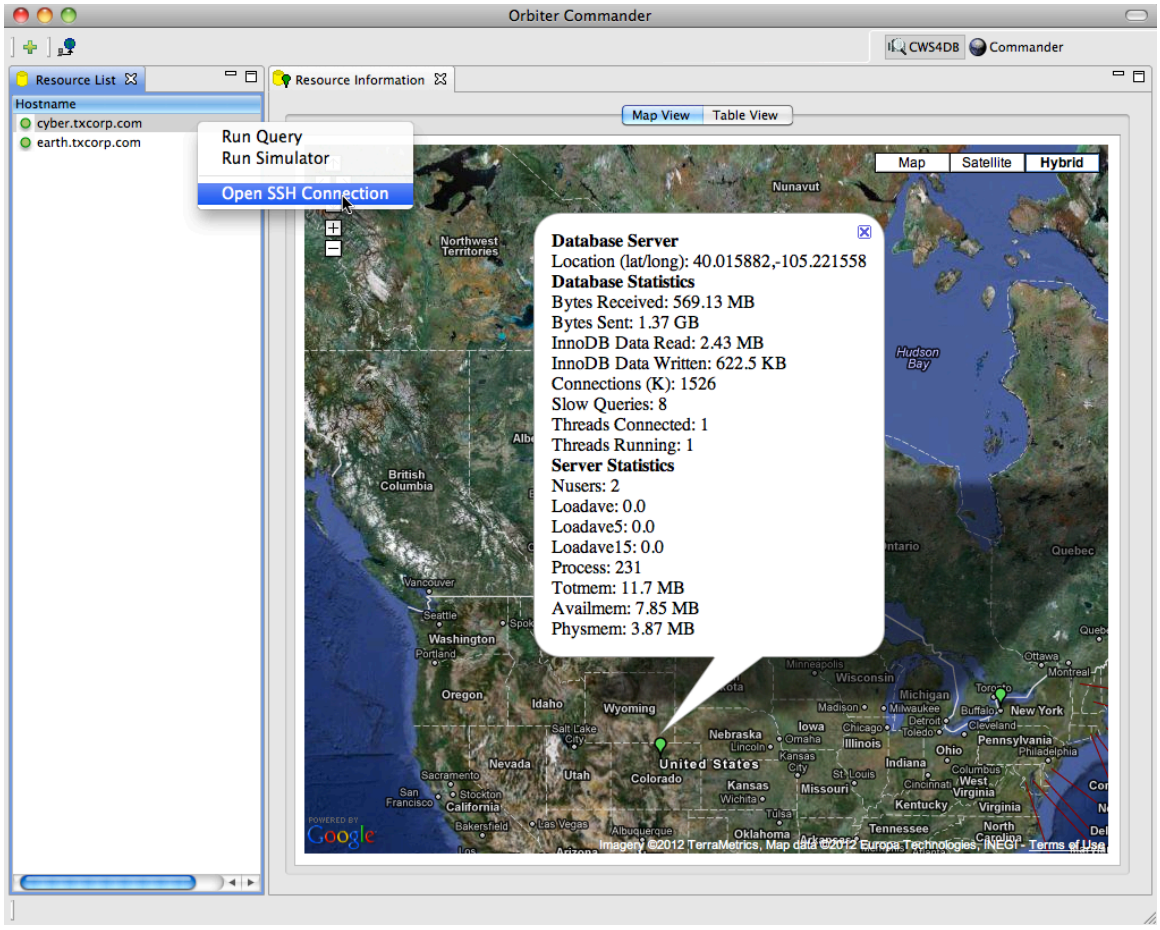


Figure 23: Orbiter Commander STAR Resource Monitor

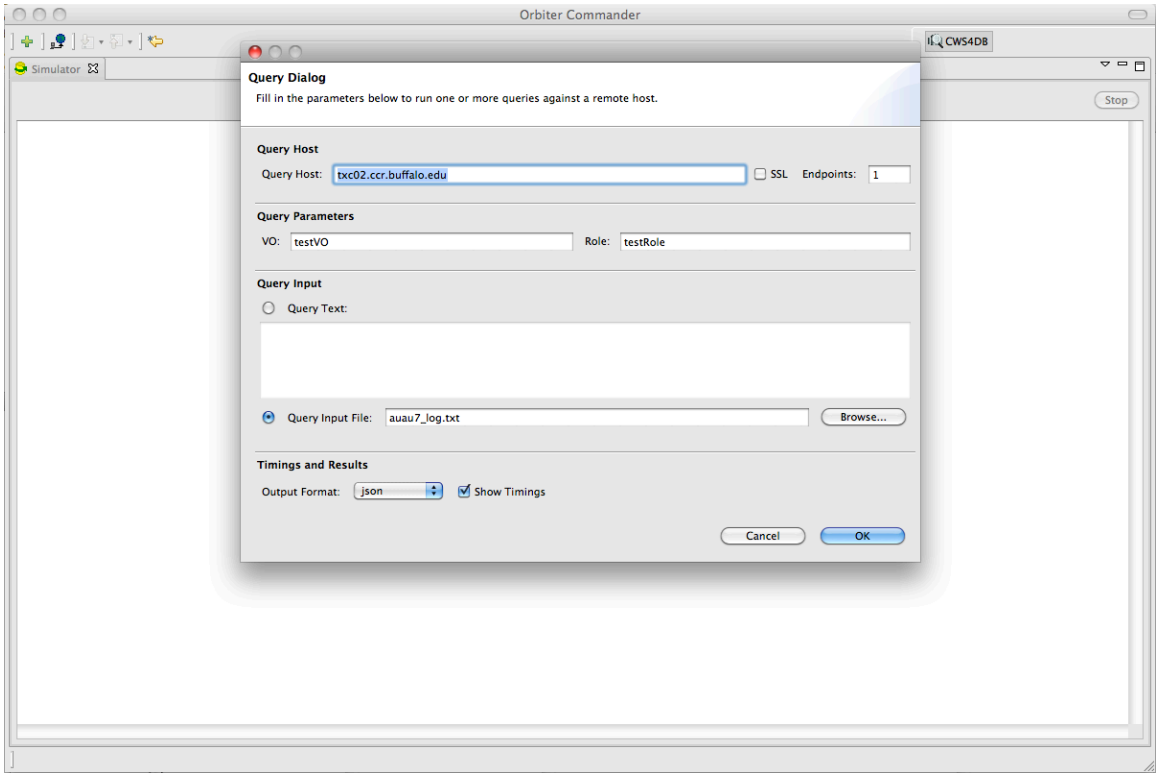


Figure 24: Orbiter Commander STAR Simulator Module

Query SSH Module

The Orbiter Commander Query SSH module provides a terminal window permitting secure connections to remote resources. This module will accept connection information and make an SSH connection to a remote resource, providing an interactive terminal for working on the machine. This module enables secure authentication using a password, passcode or SSH key pair. Complete with VT100 character support, this module provides a rich interface for interacting with remote resources used in the CWS4DB effort. A screenshot of this module is shown in Figure 25.

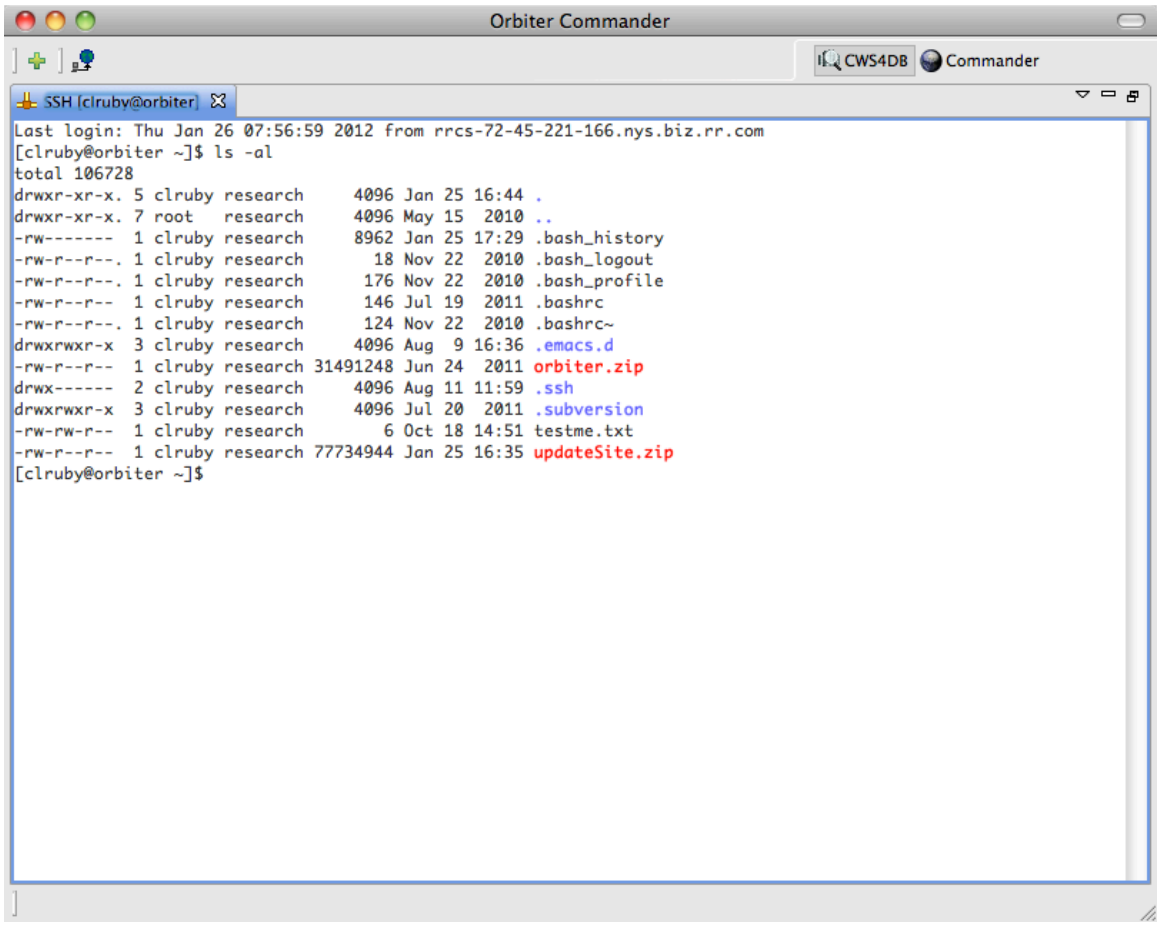


Figure 25: Orbiter Commander Query SSH Module

This module delivers its own complete version of the SSH protocol, such that systems without SSH capabilities can immediately interact with remote resources out of the box, without the need for other supporting libraries or applications on the system. However, if the user's system has already been configured for SSH via the command line, Commander will utilize the existing SSH configuration file to provide a dynamic point-and-click interface to their stored configurations. Shown in Figure 26, the saved configurations are displayed on the left, and selecting one of these configurations will automatically fill in the configuration options on the right. This permits a simple one-click connection to preconfigured hosts.

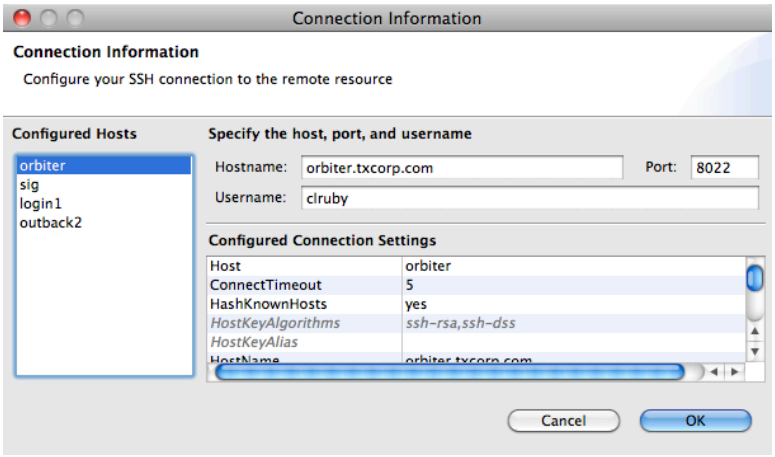


Figure 26: Orbiter Commander Query SSH Configuration

Python Interpreter Module

The Orbiter Commander Python Interpreter module allows users to interact with their command-line python installation on their local machines. By configuring Commander to use the path to an existing python installation, this module is capable of executing single or multi-line Python scripts loaded from an external file. In addition, this module can be used interactively by entering python commands through the prompt provided in the module's main view. In the screenshot featured in Figure 27, the Commander Python Interpreter module is being used to execute a Python script that tests an Orbiter web service. The service URI was specified as a command-line argument.

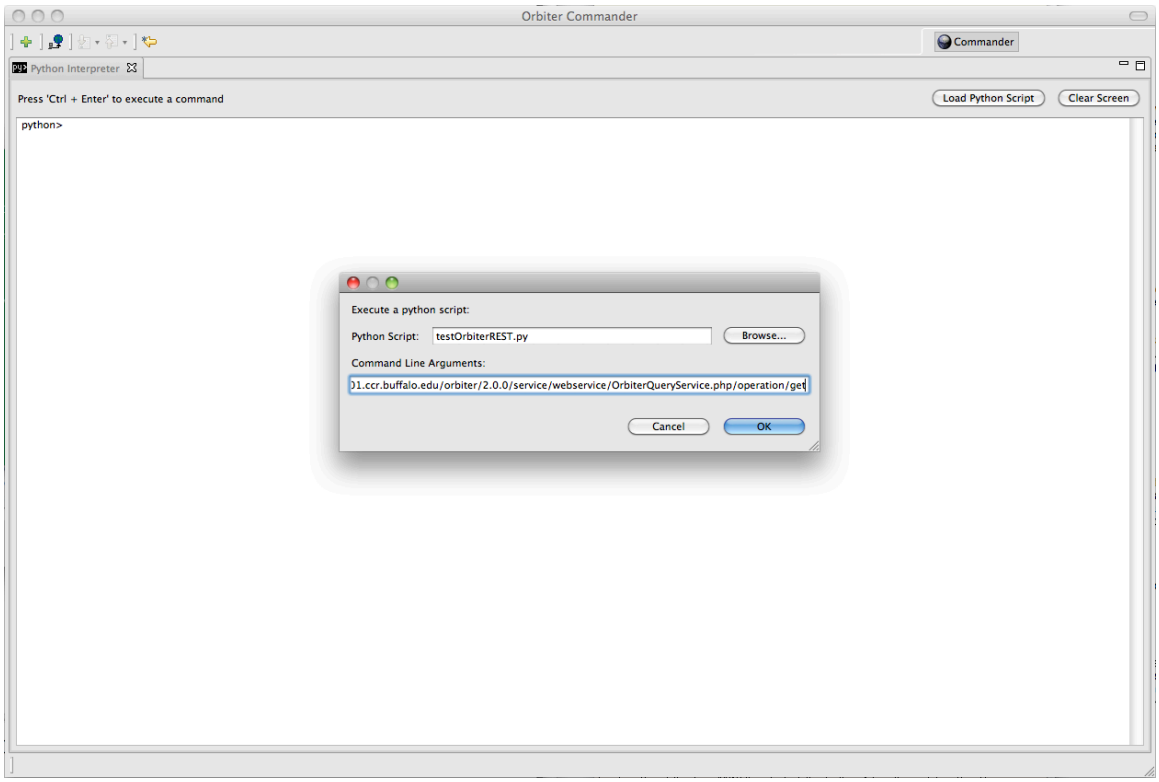


Figure 27: Orbiter Commander Python Interpreter Module

Orbiter Federation Explorer

The Orbiter Commander Federation Explorer module provides an interactive interface for browsing and testing Orbiter web services developed over the course of this effort. An Orbiter web service interface was created that returns the base stations, network nodes, and available services for an Orbiter host endpoint. This tool permits Orbiter administrators to interact directly with the Orbiter web services for development and testing purposes.

After an Orbiter base station or network node is selected, a list of all Orbiter SOA web services is provided, in groups by package names. Selecting a service will display its service Application Programming Interface (API), Web Service Description Language (WSDL) definition, and a schema, generated by the service itself to describe its capabilities and parameters. A screenshot of this interface is included in Figure 28.

This module also permits users to interact with these services through a simple and dynamically-generated Graphical User Interface (GUI). This dynamic interface, shown in Figure 29, will display all possible operations for the selected service, and for each operation will list the possible parameters, and if applicable, the possible values. The graphical representation of the available service operations and attributes is driven by a call to retrieve the WSDL and schema for the service. This tool examines these descriptors and dynamically generates the testing interface, allowing this interface to automatically adjust as services and their APIs evolve over the course of development.

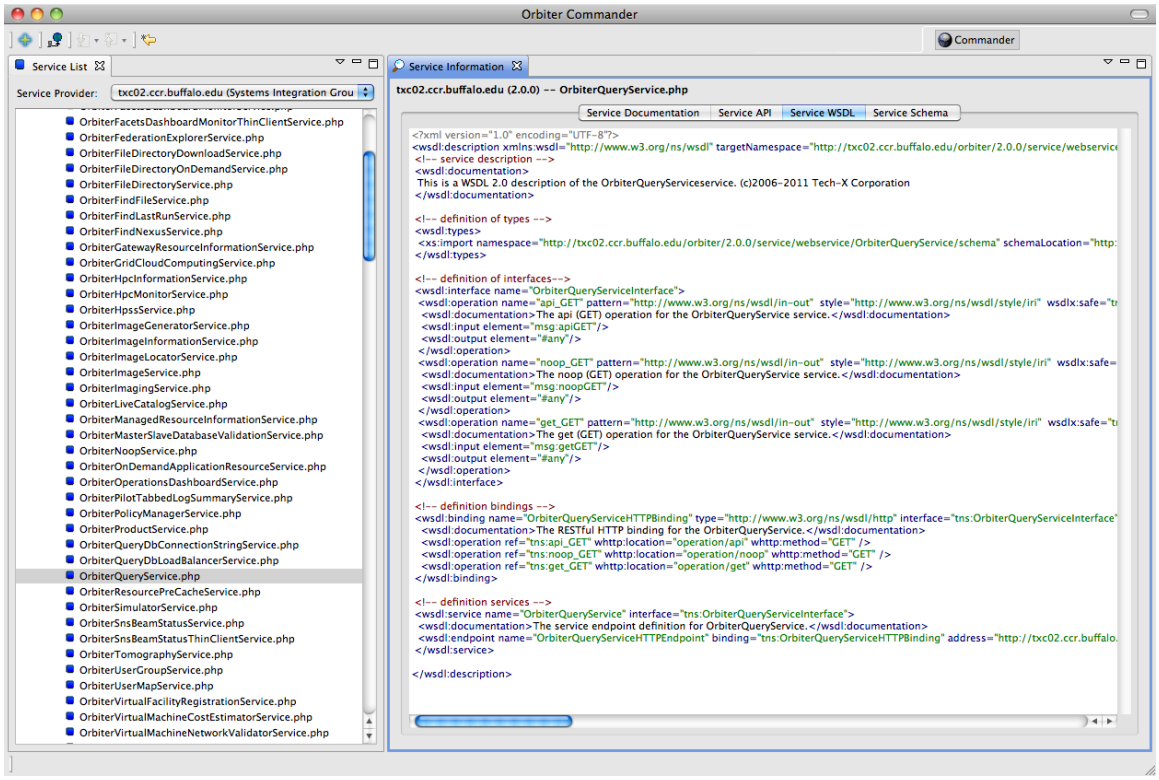


Figure 28: Orbiter Commander Federation Explorer Module

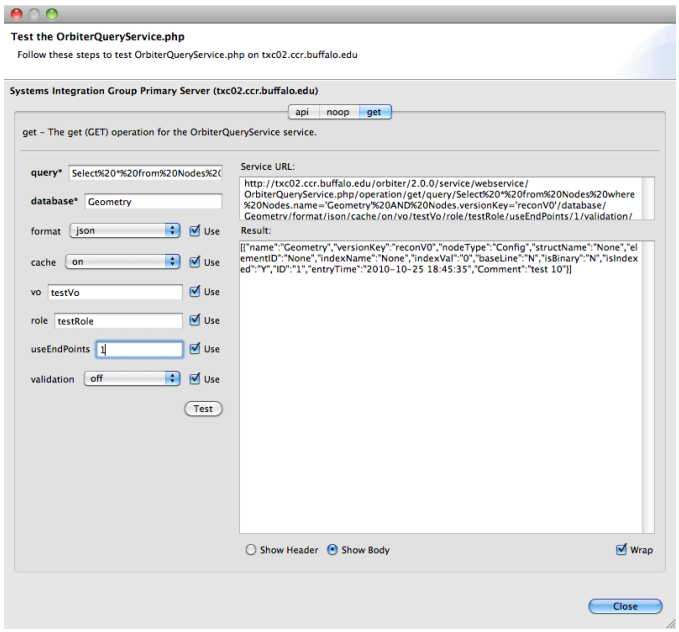


Figure 29: Orbiter Commander Federation Explorer Dynamic Testing Interface

Users can set service call values, omit or include optional parameters, and opt to use SSL encryption for the calls. Clicking "test' will execute the service, using imported Orbiter credentials for secure service calls, and will print the authenticated call, response header, and response message body for the user's review. This has proven to be an indispensable tool in developing and testing the services supporting this effort.

Commander Help and Manual

In conjunction with Orbiter Commander we have been developing a comprehensive user manual that is fully integrated into the Commander application via Eclipse RCP extension points. This manual, with content for each Commander module as well as the application framework as a whole, provides searchable as well as browsable content to assist users in understanding and interacting with the application. This help content can also be reviewed through standard web browser; a screenshot of this portion of this user manual content is included in Figure 30. These help pages are attached later in this document.

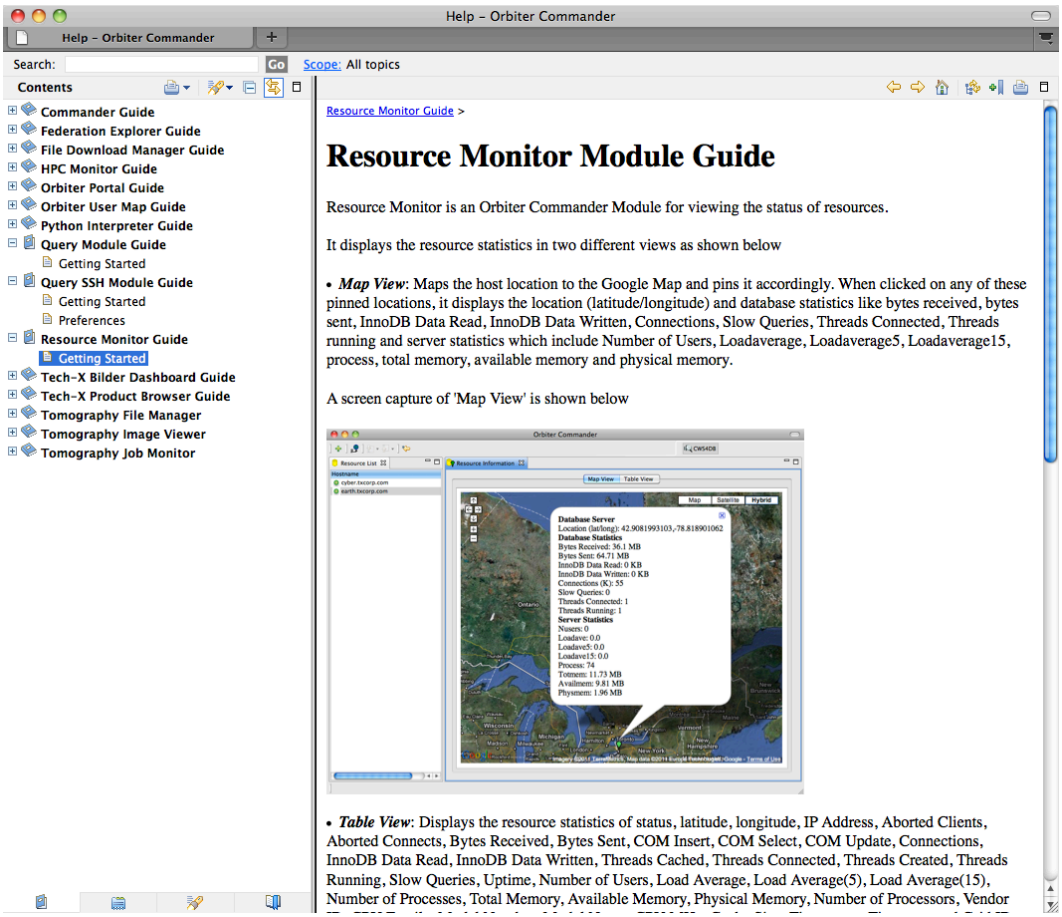


Figure 30: Orbiter Commander Manual and Help Content

2.3 Task 3. Design and Implement Auto-Caching Infrastructure

Objective: Provide a sophisticated auto-caching mechanism in order to increase the effective system performance based on work with our partners. We have determined that a two-level auto- caching algorithm is necessary and desirable to increase the effective system performance for our STAR collaborators. The development and testing of this system is our main objective.

Summary:

a) Develop a sophisticated auto-caching mechanism to increase system performance
b) Develop a testing suite for determining performance of the auto-caching candidate infrastructure configurations. This will aid in the determination of the auto-caching performance evaluation.
c) Investigate the performance of several execution node auto-cache locations
d) Investigate several auto-caching storage strategies in order to efficiently and reliably determine if a query result file exists
e) Investigate hierarchical storage strategies for potentially millions of query result files
f) Investigate file system inode limitations and performance during our investigation of the auto-caching file and directory structure
g) Implement caching query results on a proxy server and retrieve/copy them from another server when requested without making a database call.

We have accomplished our goals in this task over the course of this effort, investigating several caching mechanism and developing a RESTful web service interface as well as a graphical user interface for interacting with this CWS4DB component. This work has a significant potential for increasing STAR application performance where network congestion or network latency or network bandwidth are key factors. Specific work on this task is included in the following subsections.

Caching Mechanism Investigation and Service Interface

In this task we implemented query result caching on proxy servers at different levels of network and made them available for other servers in the network. When a request comes in and if the query result was not found in cached files list on this requested server, and is available in one of the proxy servers in the network, the cache file will be retrieved rather than making a call to the database server. The function of a proxy server that caches query result on the server's hard disk so that the result can be quickly retrieved by the same or a different user the next time that query is requested. The proxy cache eases bandwidth requirements and reduces delays that are inherent in a heavily trafficked, Internet-connected network. Because the result is stored locally on the proxy server, the file is delivered to the next request at local network speeds. The proxy cache has query results already stored and can retrieve them for the user quicker than having to retrieve them from the Database Server.

We have wrapped this caching mechanism in an Orbiter RESTful web service interface to allow remote use to authenticated and authorized users. These users can manage the caching by setting a request parameter for caching to 'on' or 'off'. When cache is set to on, first the service will look for any cached files in the cache folder and the cache file will be retrieved rather than making a call to the database server. If the cache file doesn't exist in the cache folder then the service will make a database call and the query result will be cached so that the next time the same request comes in the cache file will be retrieved. An authenticated and authorized user has the ability to get, list, delete, and recreate the cache by using OrbiterCacheFileService.

We developed and tested the auto-caching mechanism by using a representative STAR application code database run log. This log is representative of a STAR application reconstructing 50 events of 500GeV data. There are approximately 6700 database queries required for this reconstruction. The simulation was run between our Boulder, Colorado Tech-X orbiter.txcorp.com testbed server and the dbx.bnl.gov server located at BNL. The processing time for the simulation was approximately 806 seconds. We also have run this simulation from the Tech-X Buffalo, New York office with significantly better performance of approximately 334 seconds. This is representative of the current network conditions, server load, script and database overhead. We also ran this simulation on our local orbiter.txcorp.com MySQL database server in approximately 38 seconds. This analysis gave us insight in creating an efficient auto-caching mechanism capable of storage and indexing of many queries representing a relatively small amount of data.

We have investigated the performance of several execution node auto-cache locations for example, private local scratch space, network mounted files system placement, and group accessed network mounted file system placement. We have tested this scenario for determining the performance of the auto-caching and the testing results are shown in Figure 31.

CWS4DB Infrastructure Load Balancing Design

In conjunction with this effort we conducted an in-depth investigation into the STAR task 1 requirements, and incorporated several testing scenarios in order to understand the load balancing impact on our auto-caching mechanism. There were many different algorithms to consider in load balancing which server should receive the next connection, and how to cluster queries to servers that potentially have memory caches of the query result of interest. We investigated random, round-robin, least connections, fastest response, hashed, and weighted algorithms.

In general the best algorithm depended heavily on the server workload. Adding a new server to the pool was generally not as simple as plugging it in and notifying the load balancer. Normally, there must be a ramping up of workload to a given server that complicated our auto-caching mechanism even further. We made progress on functional partitioning and filtering and data partitioning methods. We implemented the Sphinx indexer for MySQL and tested the increased performance in indexing query strings.

Network bandwidth was important and depended on the last mile normally. To investigate

Figure 31: Orbiter Simulator Service Performance

```
python ./testOrbiterREST.py https://cyber.txcorp.com/orbiter/service/star/OrbiterStarSimulatorService.php
/operation/runfile/debug/on/format/json/host/local/file//tmp/testfiles/star.pp500.full.sql/
address/http://64.240.154.24/orbiter/service/star

Service attributes : host/local, address/http:// and address/http://64.240.154.24/orbiter/service/
star IP Address for DNS name resolution.

Result:

Number of trials averaged: 1
Total number of queries: 6549
Total size of queries: 38963293
Total query time: 139.37202596664

Before the pre-cache service populates any cache files, the service request has to make database calls
to retrieve the query results. We have observed delays in processing the request.

python ./testOrbiterREST.py https://cyber.txcorp.com/orbiter/service/star/OrbiterStarSimulatorService.php
/operation/runfile/debug/on/format/XML/host/remote/file//tmp/testfiles/star.pp500.full.sql/
address/http://64.240.154.24/orbiter/service/star

Service attributes : host/remote, address/https:// and address/https://cyber.txcorp.com for
DNS name resolution.

Result :

Number of trials averaged: 1
Total number of queries: 6549
Total size of queries: 38963175
Total query time: 644.20057988167

Now we ran the pre-cache service to populate the pre-cache files at this resource node.

python ./testOrbiterREST.py https://cyber.txcorp.com/orbiter/service/star/
OrbiterResourcePreCacheService.php/operation/runprecache/
resource/64.240.154.24\|star\| ...

Once the pre-cache service has populated all the cache files, we made a service request and the processing
response was significantly faster than when no cache files were existed at this node.

python ./testOrbiterREST.py https://cyber.txcorp.com/orbiter/service/star/OrbiterStarSimulatorService.php
/operation/runfile/debug/on/format/json/host/remote/file//tmp/testfiles/star.pp500.full.sql/
address/http://64.240.154.24/orbiter/service/star

Service attributes : host/remote, address/http:// and address/http://64.240.154.24/orbiter/service/
star IP Address for DNS name resolution.

Result:

Number of trials averaged: 1
Total number of queries: 6549
Total size of queries: 38963293
Total query time: 138.62204384804
```

the database service payload size we wrote a custom RESTful PHP database service with a JSON (JavaScript Object Notation) payload to compare with the XML payload. We logged the performance data for each SQL operation to calculate and log JSON and XML payload size. On average over a dataset the equivalent JSON payload was 8.8-10.1 times smaller. In general an order of magnitude lower bandwidth loading is required with the JSON PHP service.

Tech-X has installed Nimbus and utilized the Nimbus client with the available science clouds in support of the STAR on-demand database service.

- The Nimbus infrastructure provided limited upload/download bandwidth consistently.
- The required STAR image is relatively large due to the size of the MySQL database.
- We investigated several ways of populating the STAR database and tested query performance with our RESTful PHP JSON database service successfully.
- The OGSA-DAI XML database services could not be loaded on the Nimbus science cloud due to memory constraints.
- We investigated the cloud enabled MySQL database Drizzle.

The latest version of Babel has been implemented on the STAR DB driver as well as the integration of the RESTful PHP JSON database services. We have also made progress on the development of a JavaScript GUI for displaying load balancing and system status. The load-balancing infrastructure was currently deployed within the BNL firewall and we installed a version at Tech-X Boulder Headquarters. The design and implementation of this mechanism is featured in Figures 32, 33, and 34.

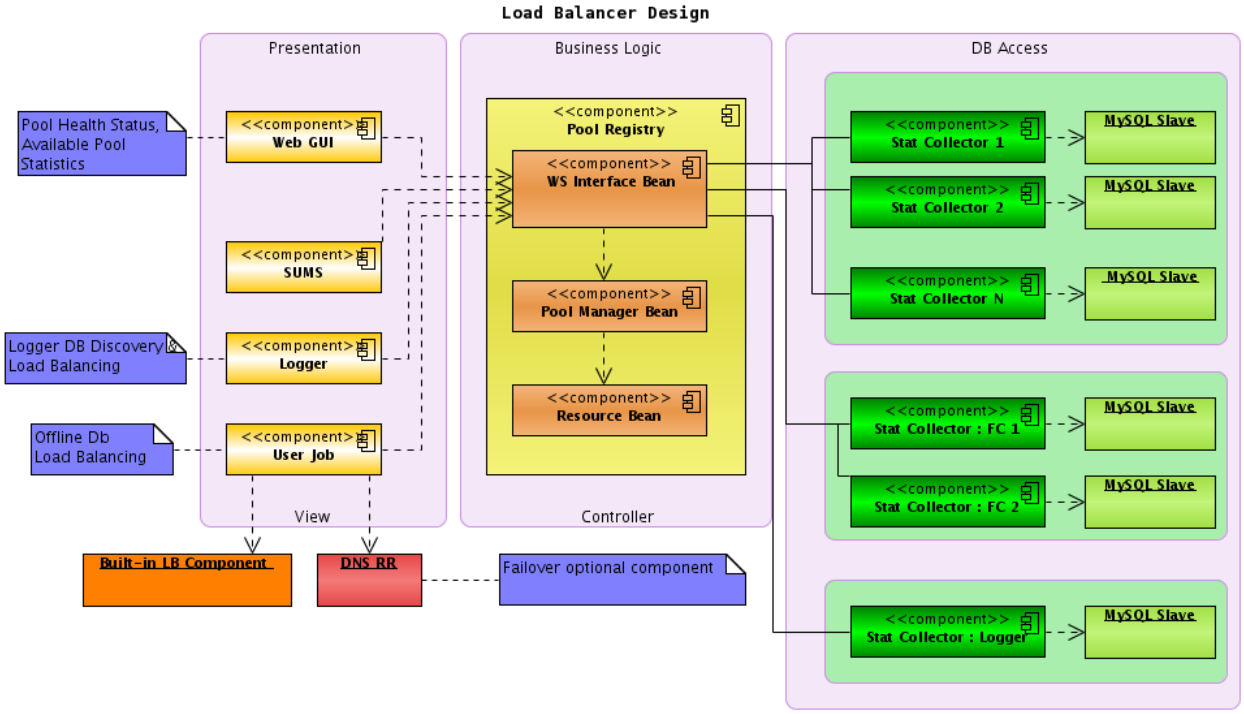


Figure 32: CWS4DB Load Balancer Design

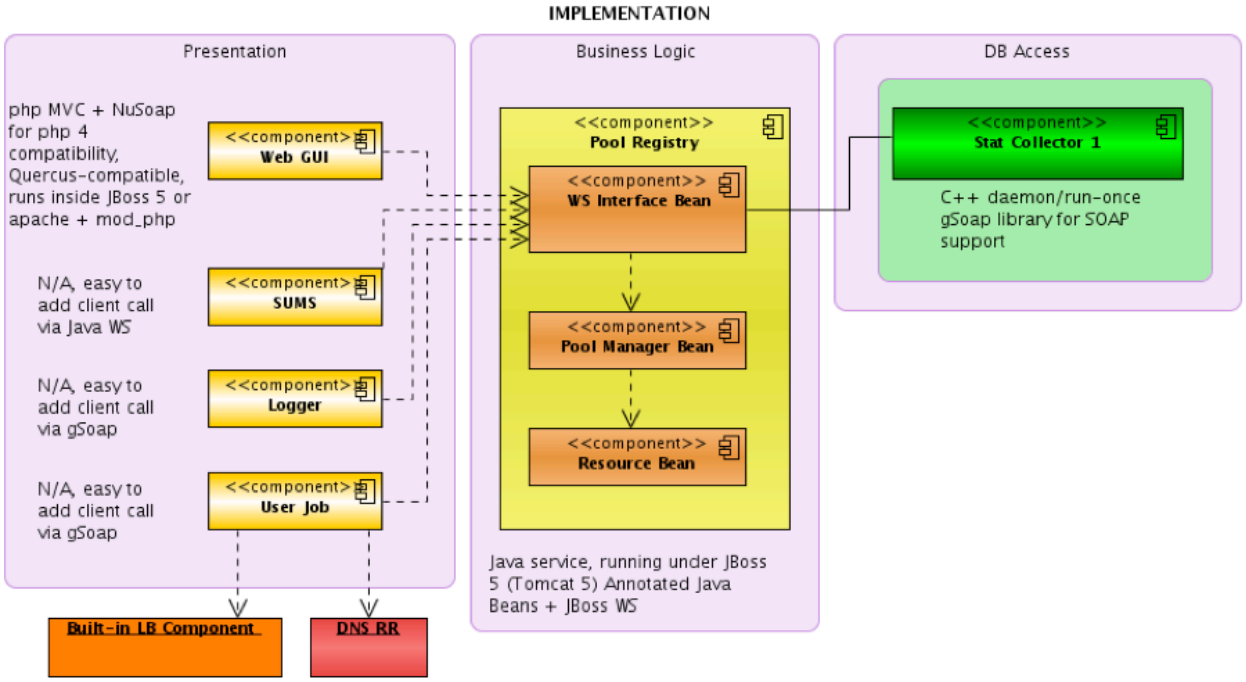


Figure 33: CWS4DB Load Balancing Implementation

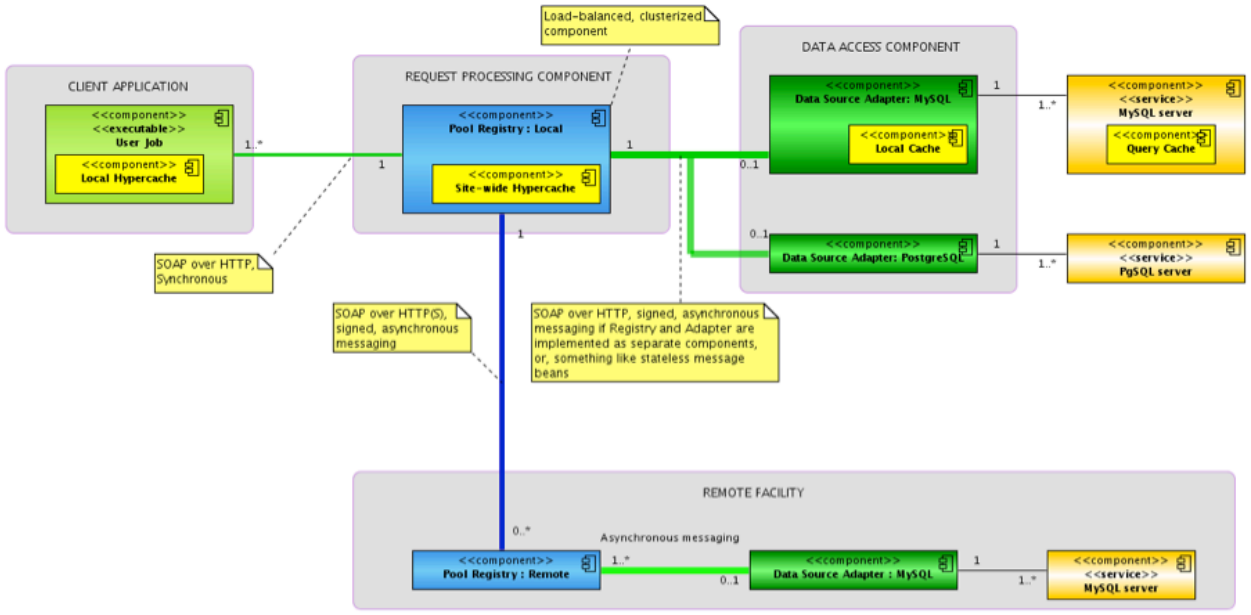


Figure 34: CWS4DB Load Balancing Composite

2.4 Task 4. Enable Multi-VO Role-Based Capabilities

Objective: Develop the CWS4DB infrastructure required for user-friendly management and caching capabilities. The authentication and authorization for multiple Virtual Organizations (VOs) is essential for delivering a high quality of service with user-friendly management capabilities. We will use the standard GSI security, authentication, and authorization methods available within the Globus infrastructure for this task.

Summary:

a) Enable Multi-VO Role-Based Capabilities using standard OSGA-DAI security, authentication, and authorization methods available within the 3.0 version
b) Implement the GSI security features of Globus
c) Investigate the effective VO role-based utilization of the auto-caching capabilities as described in Task 3
d) Investigate providing the capability for private usage of cached queries

We have met our goals for this task over the course of this project, developing services supporting VO management and query capabilities. Details are discussed in the following subsections.

Virtual Organization Management Service

We have completed the development and testing of the hierarchical Virtual Organization (VO) role-based task work. We installed and configured a test Virtual Organization Management Service (VOMS) on a testing server called cyberrepo.txcorp.com at the Tech-X Corporation. This server was be used for generating a VOMS Proxy with role-based attributes

for testing our CWS4DB role-based capabilities and in addition the project Principal Investigator, Dr. Green is a member of the STAR Collaboration, which has a VOMS server for testing also. As envisioned in our Phase II task work plan the use of a grid unique id (guid) functioned as expected as the project PI, Dr. Green, has used this approach in some of his previous work.

In addition, we completed a proof of concept trial run using our CWS4DB RESTful Data Service interface for passing a X509 proxy or a VOMS X509 proxy to our service and successfully extracting the subject Distinguished Name and role where applicable. This information was then used for authentication/authorization and enabling access to the auto-caching hierarchical storage scheme.

RESTful VO-Enabled Query Service

We were able to provide the capability for private usage of cached queries by using the following directory structure: *path-to-cache-storage/cache/guid/VO/role/md5sum*. Only an authenticated and authorized user can set vo (virtual organization name) and role (his role in the organization) for Multi-VO Role-Based caching. When a query request is made with vo and role, first the service will look for any cached files in the cache folder created(using sha1) for the specific Virtual Organization and the cache file will be retrieved rather than making a call to the database server. If the cache file doesn't exist in the specified VO cache folder then the service will make a database call and the query result will be cached (sha1) so that the next time the same request comes in the cache file will be retrieved. An authenticated and authorized user has the ability to get, list, delete and recreate the Multi-VO Role-Based cache by using OrbiterCacheFileService and by passing virtual organization name and role.

For a detailed explanation how it was implemented, path-to-cache-storage denotes the site specific disk storage location, cache denotes the top level auto-caching directory (this directory will have list access removed a-x), grid unique id (guid) denotes a 64-128 character based string value stored as a site specific cache key value, VO denotes the Virtual Organization short name, role denotes the users mapped role, and md5sum denotes the cached query result data file. This storage scheme will provide private access to VO role-based authorized users as without the knowledge of the site specific cache key value a user cannot access the file system cached query results as the top level directory list access has been removed.

The Orbiter Query Service is shown in Figure 35.

Figure 35: Orbiter Query Service

```
Service Attributes:

- /orbiter/service/star/OrbiterSTARQueryService.php/format/{XML|JSON}/host/{local|remote}/database/
  {database name}/query/{query string}/cache/on/vo/{virtual organization}/role/{admin/user}

- Service attribute definitions.

path-to-cache-storage
- Will be taken from define('ORBITERCACHEFILELOCATION', '/tmp/cache');

/vo/{virtual organization}/
- default value: null
- Virtual Organization short name.

/role/{admin/user}/
- default value: null
- Requesting user role.

/guid/{will be the Sha1 of vo + role}/
- default value: null.

Example to use private cache using service OrbiterSTARQueryService:

- Service attributes:

path-to-cache-storage
- Will be taken from define('ORBITERCACHEFILELOCATION', '/tmp/cache')
/vo/{virtual organization}/
- value: star.

/role/{admin/user}/
- value: admin

/guid/{will be the Sha1 of vo + role}/
- value: 664uncgtg33g9mng6ons38lj11.
/cache/on

- When cache is set to "on", it caches the response to the cache location set in the directory structure.

The result of the query is cached in /
tmp/cache/cache/664uncgtg33g9mng6ons38lj11/star/admin/869uncgtg33g9mng6ons38lj38
(Query result Sha1).
```

2.5 Task 5. Develop Dynamic On-Demand Data Resource Access

Objective: This on-demand service will provide a STAR MySQL database instance using the Virtual Workspaces infrastructure, Sun Grid Utility Computing resources, and investigate Grid deployments. Based on the load balancing and performance optimization study conducted in the Phase I project an on-demand data resource CWS4DB system capability is desired. This on-demand service will provide a STAR MySQL database instance using the Virtual Workspaces infrastructure (Nimbus) developed at Argonne National Laboratory. This database can then be integrated with CWS4DB data resource nodes as a data resource for providing additional load balancing capabilities.

Summary:

a) Develop Dynamic On-Demand Data Resource CWS4DB system capability
b) Integrate STAR MySQL database with CWS4DB data resource nodes as a data resource for providing additional load balancing capabilities.
c) Investigate expanding this on-demand service by developing infrastructure and tools for deploying STAR MySQL database servers on Sun Utility Grid Computing resources
d) Integrate these database servers within the CWS4DB system as OGSA-DAI data resources and evaluate their performance and on-demand capabilities.
e) Utilize the Virtual Workspaces execution environment developed at Argonne National Laboratory and supported by the NSF Computer Systems Research (CSR) Virtual Playgrounds project
f) Investigate deploying and configuring an optimized STAR MySQL database server utilizing the Virtual Workspaces infrastructure
g) Evaluate the databases on their performance and on-demand capabilities
h) Find a load balancer algorithm to calculate the ranking.
i) Set up a virtual machines infrastructure in place at Buffalo to perform testing.

Collaboration with Argonne National Laboratory

Our Argonne National Laboratory (ANL) collaborator, Dr. Keahey, participated in further developing the dynamic STAR on-demand resource that is the main focus of this task. We obtained accounts within the Nimbus cloud-computing environment for testing our CWS4DB infrastructure and we installed and tested the base Nimbus infrastructure on our Tech-X test bed.

We made significant progress on this task with Tech-X personnel as out ANL collaborator had significant prior committments. This has enabled the dynamic STAR on-demand resource that is the main focus of this task. We obtained accounts within the Nimbus cloud-computing environment for testing our CWS4DB infrastructure and we have installed and tested the base Nimbus infrastructure on our Tech-X test bed. We completed the design and development of this task and determined the bandwidth limitation of the Nimbus infrastructure. Due to the size of the STAR collaboration database the required Virtual Machine (VM) image size was problematic when starting the on-demand VMs.

We have created a table which will have all the available database resources with type, rank and the status. An authorized user can add a new db resource to the list by using the Query DB Connection String Service and the load balancer sets the ranking based on weighted score and make it available to use by the query service and other services. That way every time a new database resource node is added to the virtual work stations on demand it will be made available for all the services to use.

In the Task 2 we have setup our physical infrastructure to have a cluster-like architecture with a front-end, and a set of cluster nodes and installed OpenNebula. We have integrated STAR database resources with the OpenNebula distributed virtual infrastructure resource nodes as a data resource, so the Orbiter Load Balancer balances the service requests load among these nodes. A brief example of how to use these two services are explained below.

Orbiter Query Database Load Balancer Service

In conjunction with this task we have developed a Query Load Balancer Orbiter web service. This service is responsible for load balancing the query and other databases. Also used for updating the database rank and status. The service response will return connection strings of the type specified by the user based on the rank and status.

Brief example of use: The Orbiter Load Balancer Service is shown in Figure 36.

Orbiter Query Database Connection String Service

We have also developed a Query Database Connection String Orbiter web service to facilitate work on this task. This service provides the user an ability to add, update, and delete the database connection string information so that the user can manage database resources on demand.

Brief example of use: The Orbiter Query Db Connection String Service is shown in Figure 37.

Figure 36: Orbiter Load Balancer Service

```
// Service name
OrbiterQueryDbLoadBalancerService.php

// Service API
/operation/api/
    - output a service API listing only

// Service operations
/operation/get/
    - Gets a list of connection strings
/operation/update/
    - Updates the rank of a given connection string
/operation/schema/
    - Displays the schema.
/operation/wsdl/
    - Displays the wsdl.

// Service attribute definitions
/operation/{get|update|schema|wsdl}
    - Performs the user specified operation
    - default value : api
/html/{on|off}/
    - default value : off
    - When on, displays the schema or wsdl in the html format
/type/{star|orbiter|cyber}/
    - default value : star
    - type of the database
/endpoints/{integer value}
    - default value : 3
    - Number of connection strings to be retrieved
/rank/{integer value}/
    - default value : 1
    - database rank
/constr/{string value}/
    - default value : null
    - Connection string
/status/{1|0}/
    - default value : 1
    - connection string status
/format/{json|api|schema}/
    - default value :json
    - Output format
```

Figure 37: Orbiter Query Db Connection String Service

```
// Service name
OrbiterQueryDbConnectionStringService.php

// Service API
/operation/api/
    - output a service API listing only

// Service operations
/operation/insert/
    - Insert the connection string
/operation/update/
    - Updates the rank and status of a given connection string
/operation/delete/
    - Deletes the connection string
/operation/schema/
    - Displays the schema.
/operation/wsdl/
    - Displays the wsdl.

// Service attribute definitions
/operation/{api|schema|wsdl|insert|update|delete}
    - default value: api
    - Performs the user specified operation
/html/{on|off}/
    - default value: off
    - When on, displays the schema or wsdl in the html format
/constr/{string value}/
    - default value: null
    - Connection string
/type/{star|orbiter|cyber}/
    - default value: star
    - type of the database
/status/{1|0}/
    - default value= 1
    - connection string status
/rank/{integer value}/
    - default value: 0
    - database rank
/format/{json|api|schema}/
    - default value: api
    - Output format
```

2.6 Task 6. Develop Fault Resilient Data Resource Pathways

Objective: Investigate eliminating a single point of failure for the STAR C++ API bound codes database query requests.

Summary:

a) Develop a fault resilient CWS4DB capability that will automatically tolerate a failed OGAS-DAI data resource and satisfy data resource queries from a designated secondary data resource
b) Incorporate the fault resilient CWS4DB capability into the CWS4DB Data Resource Node service level
c) Investigate failure modes of the OGSA-DAI data resource
d) Implement several recovery mechanisms including timeout retry, secondary data resource utilization, and finally local data resource usage in order to develop a robust fault resilient data resource pathway

We have met our goals for this task in conjunction with this effort. The Principal Investigator, Dr. Green has defined the fault resilient data resource pathway. We have successfully tested this proof-of-concept in the Orbiter RESTful infrastructure and we plan to implement this on Eucalyptus once it is in place at BNL. This proof-of-concept is shown in Figure 38.

Figure 38: Proof-of-Concept Fault Resilient Data Resource Pathway for the Orbiter RESTful Service Infrastructure

```
try {
   $this->_activeQueryDbs[$dbConstant] =
      new mysqli($constArray[0], $constArray[1], $constArray[2], $constArray[3], $this->_dbPort);
} catch {
   // Throw a warning to the user and return the existing array if not empty
   trigger_error('Database Connection Warning!', E_USER_WARNING);
   if (!empty($this->_activeQueryDbs)) {
      return $this->_activeQueryDbs;
   } else {
      trigger_error("Orbiter Error, Connect error");
   }
}
```

This will tolerate a failed data resource and satisfy the queries from existing secondary data resources. The original OGSA-DAI SOAP services were quite rigid with respect to establishing fault resilient pathways, as the services tend to hang when a data resource was not available. Our RESTful service design is much more flexible when dealing with data resource failures. We have the ability to fully configure the number of attempts that the RESTful service will make based on the available secondary resources. Furthermore, the warnings are logged such that the resource administrator can extract the required information form the services logs for further assess the resource reliability.

We have implemented 3 levels of fault resilient mechanism in our services. Our query service uses Load Balancer to get the top three high ranked active database resources available

to make a connection. If the first resource fails to connect for some reason it will try connect second and third resources and keeps them as an array of available data resources for querying. When the database query fails at one of the resources it will try the other resources available in the database resources array. The service will trigger an error only if all the fault resilient mechanisms are failed. The number of database resources can be changed and can be set in the requesting query service to make it more fault resilient system when more db resources are available for use.

An example of the developed fault resilient mechanism is shown in Figure 39. This will tolerate a failed data resource and satisfy the queries from an existing secondary data resource.

Figure 39: Example of a Fault Resilient Mechanism for Orbiter Services

```
foreach($dbQueryList as $val) {
    $this->_activeQueryDbs = $this->setQueryDb($val);
}
if ( !empty($this->_activeQueryDbs) ) {
    return $this->_activeQueryDbs;
} else {
    trigger_error(
        "Orbiter Error 111:1425 E_USER_ERROR
         Query database connection object list is empty.",
        E_USER_ERROR);
}

$connObj = new mysqli($constArray[0], $constArray[1],
 $constArray[2], $constArray[3], $this->_dbPort);
if ( mysqli_connect_error() ) {
    trigger_error(
        "Database Connection Warning! Can not connect
         to Database server.",
        E_USER_WARNING);
} else {
    $this->_activeQueryDbs[] = $connObj;
}
return $this->_activeQueryDbs;
```

2.7 Task 7. Develop a Prototype On-Demand Data Resource Node

Objective: Investigate and prototype the deployment of an on-demand data resource node to meet the dynamic data demands of the STAR collaboration.

Summary:

a) Investigate a Level 4 deployment of the STAR CWS4DB data resource node
b) Work with the Virtual Workspaces team in deploying and testing the performance of on-demand data resource nodes
c) Utilize the major features of the Virtual Workspaces infrastructure in achieving this task, specifically remote deployment and lifecycle management, group management capabilities, per-client usage tracking and authentication/authorization
d) Work with the STAR BNL team with the deployment and testing of the Sun Grid Utility Computing group resources
e) Extend the collaboration with the Open Science Grid (OSG)

We have met our goals for this task in conjunction with this effort. Details are discussed in the following subsections.

Virtual Workspaces and OSG Collaboration

Similar to Task 5 the on-demand data resource node provides the capability of scaling the STAR collaboration data delivery by utilizing cloud and grid computing resources that are currently available. The Virtual Workspaces infrastructure and the Sun Grid Utility Computing group resources have been used to investigate the development of the on-demand node for servicing STAR database resource queries. We have created a table which will have all the available database resources with type, rank and the status. An authorized user can add a new db resource to the list by using the Query DB Connection String Service and the load balancer sets the ranking based on weighted score and make it available to use by the query service and other services. That way every time a new database resource node is added to the virtual work stations on demand it will be made available for all the services to use.

We are following the below listed diagram in support of on-demand resource for the STAR collaboration as STAR has worked quite closely with the Open Science Grid and the Nimbus project in providing resources for the collaboration.

We have also been working with the Eucalyptus infrastructure as the DOE Magellan cloud resources are capitalizing on its capabilities. Drs. Lauret and Green have agreed to setup Eucalyptus available resources at BNL and Tech-X for further investigating the limitations of the Virtual Facility described in Figure 40. The following sub-tasks have been undertaken to complete this task.

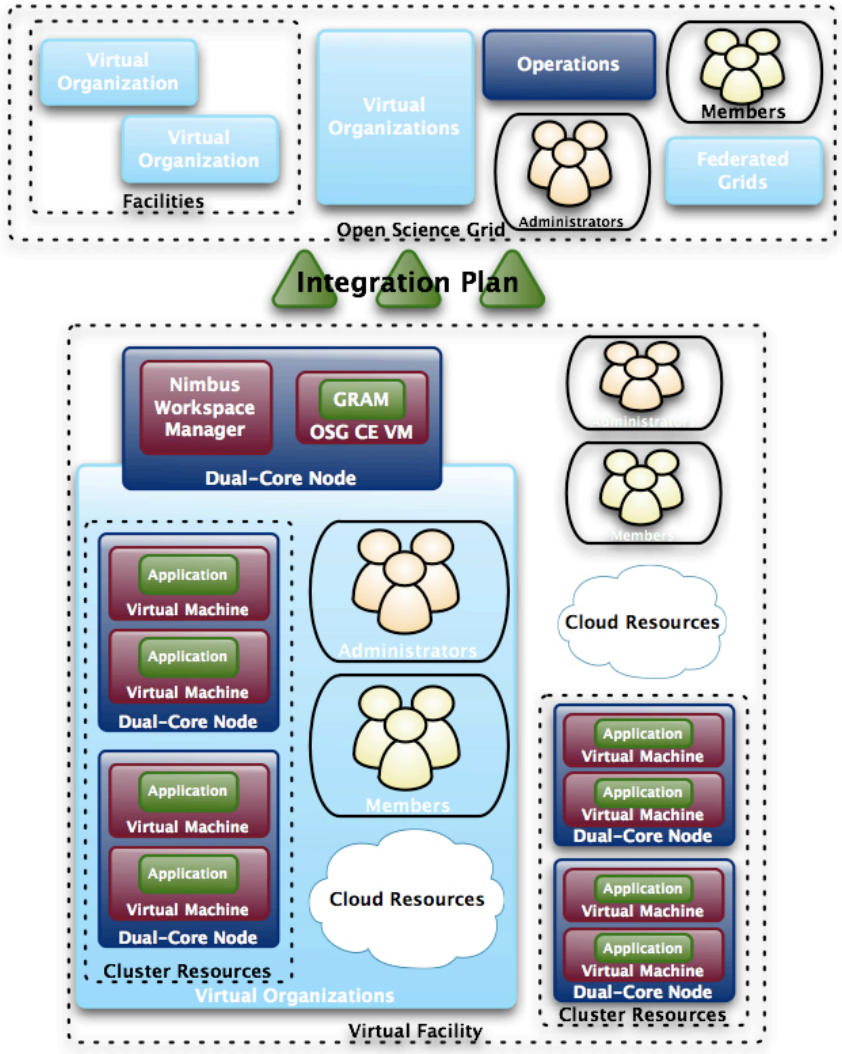


Figure 40: Virtual Facility and Open Science Grid Integration Plan

Database Testing and Timings

Using sequences of SQL operations that are recorded from actual STAR DB usage, we evaluated database performance under load by timing numerous repetitions of these operations against local and remote databases. These sequences are shown in Figures 41-46.

We have implemented code to log performance data for each SQL operation in such sequences. sql_profile.php takes the input filename as an argument. It reads the file line-by-line, attempts to parse each line as a SQL statement, and executes the SQL. It opens a connection to a local database for logging the queries and also opens a separate database connection for the timed SQL operations based on the connection statements in the input file.

The query logging uses three database tables. The 'dataset' table records the input file name and assigns it a unique dataset_id. The 'query' table records distinct query statements per dataset. The table 'query_log' records query duration, JSON size and XML size for every query executed. JSON size and XML size are determined by encoding the query results as JSON and XML strings and taking the size of those strings. Queries that returned a large number of rows and columns resulted in a PHP out of memory error when trying to encode as JSON or XML, so the SQL result set is limited to one million characters prior to encoding.

Automated Resource Load Monitor

This task implemented a basic automated load monitoring system for distributed resources. The LoadChecker system is built on RESTful Web services written in PHP and a Java client. The monitor client can use either a local MySQL database connection or Web services that connect to a remote MySQL host. To monitor a resource such as a server, the client is run on it at 1-minute intervals using the standard GNU scheduler, cron. It reads activity statistics from the operating system and MySQL server, and then writes the statistics to a MySQL database. The client and Web services exchange information using JSON (JavaScript Object Notation) messages, which are generated and handled using standard PHP JSON and json.java.org packages. The get_resource.php and post_stats.php services are RESTful, using URL parameters to identify the resource being monitored. The post_stats.php service also accepts a JSON structure sent via HTTP POST in order to support sending a larger set of statistics than would be possible if the values were URL-encoded. LoadChecker thus provides an automatic, continuous, persistent log of load statistics that is important to support stress testing and load balancing.

Nimbus and Cloud Integration

The cloud client has been downloaded; Nimbus cloud client has been installed; openssl was acquired after which access to diverse clouds has to be requested. Then dependencies have to be gathered and installed on Cyber. Certificates must be created for Nimbus upon which a p12 file would be created. This file is moved to the .globus directory, which is in the home directory of the user. The cloud allows access through an RSA public and private key pair. Those two keys are Cyber's keys and should be copied to the ".ssh" folder into the home directory of whoever intends to use the cloud. The said keys can be found in

File Name: star.pp500.full.sql					
Host	Address	Number of Trails Averaged	Total number of queries	Total size of queries	Total query time
Remote	https://cyber.txcorp.com/ orbiter/service/star	1	6549	38963175	644.20057988167
Remote	http://cyber.txcorp.com/ orbiter/service/star	1	6549	38963293	307.41246199608
Remote	https://64.240.154.24/orbiter/service/star	1	6549	38963293	487.29793095589
Remote	http://64.240.154.24/orbiter/service/star	1	6549	38963293	138.62204384804
Local	https://cyber.txcorp.com/ orbiter/service/star	1	6549	38963293	622.77593588829
Local	http://cyber.txcorp.com /orbiter/service/star	1	6549	38963293	291.01330494881
Local	https://64.240.154.24/orbiter/service/star	1	6549	38962872	478.29859399796
Local	http://64.240.154.24/orbiter/service/star	1	6549	38963293	139.37202596664
File Name: star.dau200.full.sql					
Remote	https://cyber.txcorp.com/ orbiter/service/star	1	8690	27774786	793.21806406975
Remote	http://cyber.txcorp.com/ orbiter/service/star	1	8690	27789042	513.18452215195
Remote	https://64.240.154.24/orbiter/service/star	1	8690	27789042	574.45676898956
Remote	http://64.240.154.24/orbiter/service/star	1	8690	27789042	120.59467601776
Local	https://cyber.txcorp.com/ orbiter/service/star	1	8690	27788925	798.5506029129
Local	http://cyber.txcorp.com /orbiter/service/star	1	8690	27789042	286.73853397369
Local	https://64.240.154.24/orbiter/service/star	1	8690	27788924	580.46977496147
Local	http://64.240.154.24/orbiter/service/star	1	8690	27789042	119.05681300163
File Name: star.auau200.full.sql					
Remote	https://cyber.txcorp.com/ orbiter/service/star	1	8810	35507540	819.88809108734
Remote	http://cyber.txcorp.com/ orbiter/service/star	1	8810	35559589	305.7815489769
Remote	https://64.240.154.24/orbiter/service/star	1	8810	35559523	592.56079316139
Remote	http://64.240.154.24/orbiter/service/star	1	8810	35559589	139.08594894409
Local	https://cyber.txcorp.com/ orbiter/service/star	1	8810	35559589	914.83851408958
Local	http://cyber.txcorp.com /orbiter/service/star	1	8810	35559589	296.07430291176
Local	https://64.240.154.24/orbiter/service/star	1	8810	35557603	601.53532004356
Local	http://64.240.154.24/orbiter/service/star	1	8810	35559589	140.23222184181

Figure 41: Overall Database Performance and Timings

Name	#Operations
db-perf-test.txt	6,667
offline.auau200.full.sql	8,911
offline.dau200.full.sql	8,784
offline.pp500.full.sql	6,667

Figure 42: Sample SQL Sequences for Database Performance Timings

DB Host	# Repetitions	Avg Time (sec)
dbx.star.bnl.gov	10	921.66
orbiter.txcorp.com	10	3.42
cyber.txcorp.com	10	11.63
dbx.star.bnl.gov	20	922.02
orbiter.txcorp.com	20	3.49
cyber.txcorp.com	20	11.7
dbx.star.bnl.gov	30	898.57
orbiter.txcorp.com	30	3.61
cyber.txcorp.com	30	11.88

Figure 43: Timing Results for *db-perf-test.txt*

DB Host	# Repetitions	Avg Time (sec)
orbiter.txcorp.com	5	4.16
dbx.star.bnl.gov	5	1134.09
orbiter.txcorp.com	10	4.14
dbx.star.bnl.gov	10	1090.29
orbiter.txcorp.com	15	4.1
dbx.star.bnl.gov	15	1616.98

Figure 44: Timing Results for *offline.auau200.full.sql*

DB Host	# Repetitions	Avg Time (sec)
orbiter.txcorp.com	5	2.56
dbx.star.bnl.gov	5	993.93
orbiter.txcorp.com	10	2.66
dbx.star.bnl.gov	10	1256.02
orbiter.txcorp.com	15	2.66
dbx.star.bnl.gov	15	999.29

Figure 45: Timing Results for *offline.dau200.full.sql*

DB Host	# Repetitions	Avg Time (sec)
orbiter.txcorp.com	5	4.2
dbx.star.bnl.gov	5	921.82
orbiter.txcorp.com	10	3.52
dbx.star.bnl.gov	10	921.82
orbiter.txcorp.com	15	3.39
dbx.star.bnl.gov	15	907.23

Figure 46: Timing Results for *offline.pp500.full.sql*

/usr/local/nimbus-testing/keys. In that same directory we can figure a file named password.txt. Inside of that can be found a password relevant to the cloud's initialization.

Before deploying an image, the cloud requires the user to request a lease first. All commands are located in /usr/local/nimbus-testing/NIMBUS-cloud-client/bin. This is achieved by typing:

./grid-proxy-init.sh

A Nimbus script could allow the user to perform a myriad of operations such as listing, upload deploy and delete images. Setting up custom images demands the utilization of Xen on the machine intended to be used in the creation of the image. After installation of Xen, subsequent Xen-tools such as Ubuntu, Lenny, Etch (for Debian based system users); Fedora, SuSe, Mandriva, the Yellow Dog Update Manager (for RPM based systems) also need to be installed. In addition to installing Xen, a Xen patched kernel has to be run.

Initial user interface for the on-demand data resources has several components utilizing the YUI Yahoo interface and the Eclipse Rich Client Platform (RCP) that has the capabilities for managing and running the STAR production job simulations on the available data resources.

2.8 Task 8. Prototype Pre-Cache Capabilities for Production Job Workflow

Objective: Prototype Pre-Cache Capabilities for Production Job Workflow.

Summary:

a) Run a site specific test suite of pre-defined data resource node test and perform validation queries
b) Investigate how to add capabilities for a authenticated and authorized user to invalidate the auto-caching date resource node query cache so that it can be flushed remotely.
c) Investigate adding the capability for the authenticated and authorized user to invalidate execution node query caches on a node by node basis or on a entire cluster wide basis.
d) Provide a pathway for an authenticated and authorized user upon configuration of the CWS4DB system to execute the customizable site specific test suite for pre-caching production job query caches
e) Provide this capability also for an existing and configured CWS4DB data resource nodes.

We have met our goals for this task in conjunction with this effort. We have created a web service OrbiterResourcePreCacheService that is used to pre-cache computational job query results on each resource node, which runs a star sql file and pre-populates the cache files at every node level when a resource is added to the resource list. When an on-demand data resource node is added for a STAR production and run the site specific test suite for the validation of queries, the pre-cache service will pre-cache the subsequent production run query results at the each node.

When a request comes in at a local server and if the query result was not found in cached files list on this requested server and if we set the host to remote and the address to remote server address, it will look at the pre-cache file directory and will retrieve from the proxy server that was pre-cached on the server's hard disk so that the result can be quickly retrieved.

The Orbiter Resource Pre-Cache Service is shown in Figure 47.

Figure 47: OrbiterResourcePreCacheService API

```
Service Name: OrbiterResourcePreCacheService.php
- Manages the pre-caching

Service attributes:
- /orbiter/service/star/OrbiterResourcePreCacheService.class.php/format/
  {XML|JSON}/operation/runprecache/resource/database resource address
- /orbiter/service/star/OrbiterResourcePreCacheService.class.php/format/
  {XML|JSON}/operation/flushprecache/resource/database resource address

- Service attribute definitions.
/runprecache/{use this to run the pre-cache}/
- value: on
/flushprecache/{use this to flush the pre-cache}/
- value: on
/resource/{database resource address}/
- value: database resource address
```

2.9 Task 9. Develop a Customizable Site Specific Test Suite

Objective: Develop customizable site-specific test suite

Summary:

a) Develop tests to validate and verify the performance and data delivery capabilities of the CWS4DB system
b) Use the information mined from MySQL server query log files generated by STAR production and user job queries to test and validate the CWS4DB system and available data resources
c) Provide the tools necessary for investigating the performance and tuning of specific data resource nodes
d) Work with our collaborators in determining a customizable site specific test suite to complete this task

In order to deliver a high quality of service infrastructure, a customizable and site specific test suite is required to validate and verify the performance and data delivery capabilities of the CWS4DB system. We have met our goals in conjunction with this task, discussed in more detail in the following subsections.

Continuous Integration

In conjunction with this task we have installed and maintained *phpUnderControl* as the primary tool for ensuring the quality and integrity of Orbiter services and infrastructure. This tool facilitates the continuous integration of Orbiter code to ensure that changes do not adversely impact other components within the system. *phpUnderControl*, built upon the Java *CruiseControl*, provides build statistics, metrics, test coverage analysis, code browsing, code quality analysis, automated documentation generation, and unit test execution, which composed a solid foundation for building quality services for the Orbiter SOA. A brief overview of *phpUnderControl* functionality is shown below.

1. 'Overview' shows the details like date and time of last build, date and time when it is last changed and its log entry; Number of Unit Tests passed or failed. It also displays the PHPUnit PMD rule showing the errors/warnings in the files and PHP CodeSniffer violation.
2. 'Tests' displays the status - whether success or failure and time elapsed for each test. It also shows how the unit tests and test suites are organized.
3. 'Metrics' displays the summary of project metrics as the following
   (a) Number of Build Attempts
   (b) Number of Broken Builds
   (c) Number of Successful Builds, and
   (d) Breakdown of Build Types
   (e) Scatter Plot of Good and Broken Builds across Time and Date
   (f) Unit Coverage
   (g) Area Chart of Executable and Covered Tests against Lines of Code and Build
   (h) Unit Tests
   (i) Area Chart of Executable and Covered Tests against Tests and Build
   (j) Test to Code Ratio
   (k) Area Chart of Classes, Methods,Test Classes and Methods against Classes or Methods
   (l) Coding Violations
   (m) Area Chart of PHPCodeSniffer, PHPUnit PMD and PHPDoc against Violations and Build
   (n) Test Execution
   (o) Time-Area chart of Execution time across Builds
   (p) Duplicated Code- Lines and Tokens across Builds
4. 'Coverage' displays the percentage of Coverage of Lines of Code, Functions/Methods and Classes in the current directory and colorizes the status based on a legend.
5. 'Code Browser' lets user browse the code. It also displays a summary of errors and notices against each file.
6. 'Documentation' generates documentation based on PHPDoc-formatted comments and the structure of the source code itself. It also categorizes API into the corresponding packages of source code. It also displays Todo List. The Orbiter API has been developed using PHPDoc template *'Earthli'* and has been classified into corresponding packages the classes belong to. An API for a class includes class description, authors, version, copyright, link, todo, license, method summary, methods, parameters, and exceptions. This is illustrated in the figure 48.
7. 'CodeSniffer' sniffs PHP files to detect violations of a defined coding standard. It also displays the summary of PHP CodeSniffer violation and detailed level violations. This guarantees a high quality code base adhering to the defined standards for the CWS4DB project.
8. 'PHPUnit PMD (Project Mess Detector)' scans code, looks for potential problems and reports violations of each PMD rule. The PHPUnit PMD displays the Errors/Warnings across the corresponding files identified based on the rules namely PHPUnit PMD /

CodeCoverage, PHPUnit PMD / CRAP, PHPUnit PMD / NPathComplexity, PHPUnit PMD / CyclomaticComplexity, PHPUnit PMD /TooManyFields, PHPUnit PMD / ExcessiveMethodLength and PHPUnit PMD / ExcessivePublicCount.

9. 'PHP-CPD-Copy Paste Detector' shows the duplication of lines of code in different files.
10. 'Changeset' shows the set of modifications made since the last successful build.

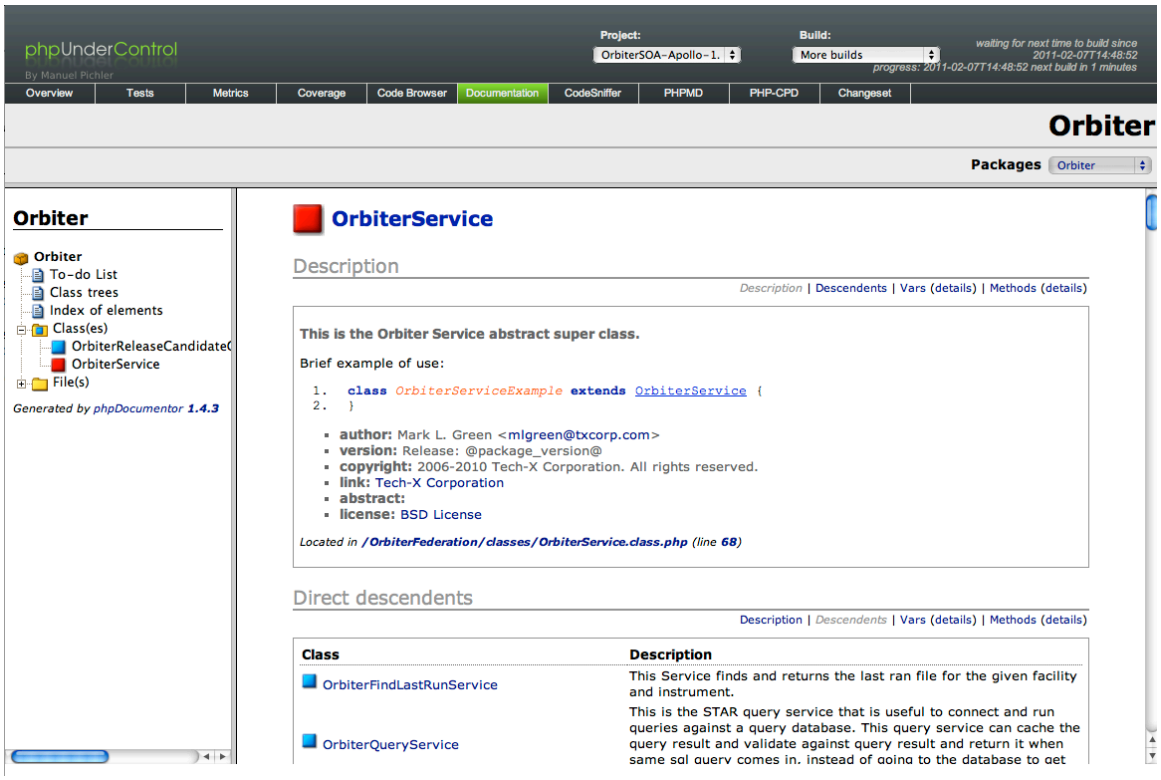


Figure 48: Orbiter phpUnderControl Documentation

The versioned releases and branching implemented in our Orbiter Infrastructure as explained in detail under Task 2 are built on different builds in Cruise Control. A screen capture of the same is shown in figure 49. This allows us to manage all of our Orbiter major versions, ensuring that each release independently passes our testing infrastructure and quality control measures.

We have developed more than 106 unit tests for Orbiter and about 81 unit tests for CWS4DB classes and the related infrastructure. These tests ensured the stability of the system as we continued to add functionality to the Orbiter SOA infrastructure over the course of this effort. Unit testing in this case examines the core functionality of the Orbiter SOA implementation, ensuring that individual and atomic functionality is consistent and functions as expected as the code base evolves over time. In addition to these unit tests we have also developed a number of service-level tests that test the overall API and expected results from calling the front-end RESTful services. Integrated with *phpUnderControl* and its automated testing through continuous integration, our service-level tests ensured that CWS4DB deployments

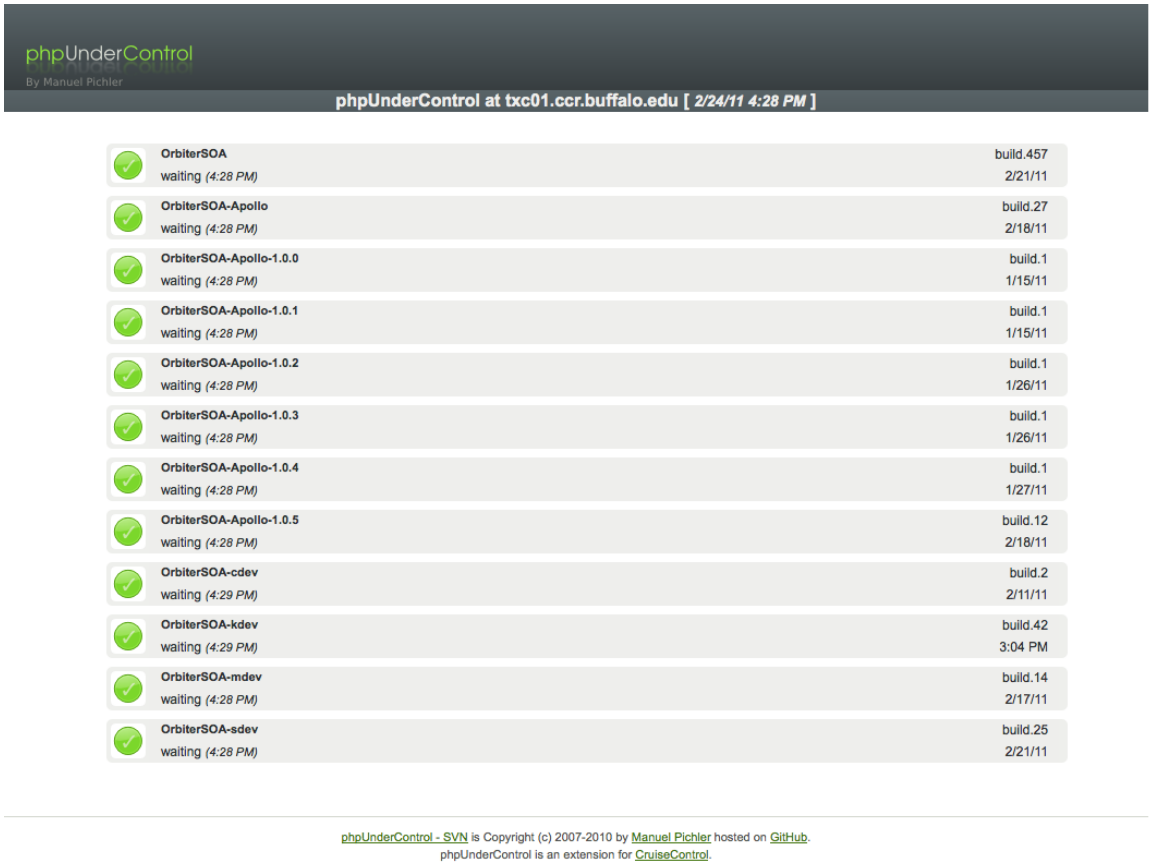


Figure 49: Versioning and Branching in PHPUnderControl

of Orbiter services maintained a stable set of services, exercising the operations of each one using PHP *cURL*.

Timing based testing using the latest STAR database snapshot

The stress testing of the CWS4DB infrastructure is done through a series of scripts and files that simulate STAR production jobs. The script listed in listed in Figure 50 simulates the STAR reconstruction database access pattern for our development and testing environment.

We set up a server at Buffalo Center for Computational Research that has a large capacity with very good connectivity that we have used to test the performance of the services. We ran tests for all the 4 sql files provided against the latest production quality database. We used the Orbiter SOA services developed over the course of this effort to test the caching, pre-caching and querying capabilities of the system on this server and were able to retrieve timings with which to benchmark other deployments. We wrapped the functionality of the above stress testing script as an Orbiter service that can be accessed through a remote host for deployed testing capabilities. An example run of the Simulator service on this server is included in the following pages, exercising the output formats and IP versus DNS lookups.

Figure 50: Script Simulating STAR Production Jobs

```
<?php
function get_time() {
   list($usec,$sec) = explode(' ', microtime());
   return ((float)$usec + (float)$sec);
}

$start = get_time();
$conn = false;
$cnt = 0;
while ($line = trim(fgets(STDIN))) {
   $handled = false;
   if (!strncasecmp("connect", $line, 7)) {
      // try to parse line as 'CONNECT user@host AS dbuser ON db'
      $strarr = split("[\t ]+", $line);
      if (sizeof($strarr) == 6) {
         $user = $strarr[3];
         $db = $strarr[5];
         $str = $strarr[1];
         $ha = split("[@]", $str);
         if (sizeof($ha) == 2) {
            $host = $ha[1];
            echo "Connecting to ".$host." as ".$user."\n";
            $conn = mysql_connect($host, $user, "");
            if (!$conn) die ('Connection error: '.mysql_error());
            $handled = true;
         }
      }
   } else if (!strncasecmp("use", $line, 3)) {
      // try to parse line as 'USE DB dbname'
      $strarr = split("[\t ]+", $line);
      if (sizeof($strarr) == 3) {
         $db = $strarr[2];
         echo "USE ".$db."\n";
         $link = mysql_select_db($db);
         if (!$link) {
            die("Can't select DB ".$db);
            $handled = true;
         }
      }
   } else if (!strncasecmp("select", $line, 6) || !strncasecmp("show", $line, 4)) {
      // process line as query
      $result = mysql_query($line);
      if (!$result) { echo "Query: ".$line."\nError: ".mysql_error()."\n";
      } else {
         $rowcnt = 0;
         while ($row = mysql_fetch_array($result, MYSQL_NUM)) { $rowcnt++;  }
         echo "Query result: ".$rowcnt." rows\n";
      }
      $handled = true;
   } else if (!strncasecmp("quit", $line, 4)) {
      // disconnect from DB if ($conn) {
      if ($conn) {
         mysql_close($conn);
         $conn = false;
      }
      $handled = true;
   }
   if (!$handled) {
      echo "\nSkipping line:\n".$line."\n";
      $cnt++;
   }
}

$elapsed = get_time()-$start;
echo "Total time: ".$elapsed." sec";
?>
```

```
python ./testOrbiterREST.py http://128.205.41.182/orbiter/kdev
/service/webservice/OrbiterSimulatorService.php/operation/
runfile/file//tmp/testfiles/auau7_log.sql.rtf/format/json/cache/on

Number of trials averaged: 1
Total number of queries: 2918
Total size of queries: 1361776
Total query time: 114.83887505531

python ./testOrbiterREST.py http://128.205.41.182/orbiter/kdev
/service/webservice/OrbiterSimulatorService.php/operation/
runfile/file//tmp/testfiles/auau11_log.sql.rtf/format/json/cache/on

Number of trials averaged: 1
Total number of queries: 2918
Total size of queries: 1359417
Total query time: 116.78107690811

python ./testOrbiterREST.py http://128.205.41.182/orbiter/kdev
/service/webservice/OrbiterSimulatorService.php/operation/
runfile/file//tmp/testfiles/auau39_log.sql.rtf/format/json/cache/on

Number of trials averaged: 1
Total number of queries: 2918
Total size of queries: 1362013
Total query time: 117.37054586411

python ./testOrbiterREST.py http://128.205.41.182/orbiter/kdev
/service/webservice/OrbiterSimulatorService.php/operation/
runfile/file//tmp/testfiles/auau200_log.sql.rtf/format/json/cache/on

Number of trials averaged: 1
Total number of queries: 2918
Total size of queries: 1362013
Total query time: 117.68230295181

--------------------------------------------------------------------------------
python ./testOrbiterREST.py http://128.205.41.182/orbiter
/kdev/service/webservice/OrbiterSimulatorService.php/operation
/runfile/format/xml/cache/on/file//tmp/testfiles/auau7_log.sql.rtf

Number of trials averaged: 1
Total number of queries: 2918
Total size of queries: 2267497
Total query time: 129.42278599739

python ./testOrbiterREST.py http://128.205.41.182/orbiter
/kdev/service/webservice/OrbiterSimulatorService.php/operation
/runfile/format/xml/cache/on/file//tmp/testfiles/auau11_log.sql.rtf

Number of trials averaged: 1
Total number of queries: 2918
Total size of queries: 2267497
```

```
Total query time: 134.40248394012

python ./testOrbiterREST.py http://128.205.41.182/orbiter
/kdev/service/webservice/OrbiterSimulatorService.php/operation
/runfile/format/xml/cache/on/file//tmp/testfiles/auau39_log.sql.rtf

Number of trials averaged: 1
Total number of queries: 2918
Total size of queries: 2267497
Total query time: 134.59920406342

python ./testOrbiterREST.py http://128.205.41.182/orbiter
/kdev/service/webservice/OrbiterSimulatorService.php/operation
/runfile/format/xml/cache/on/file//tmp/testfiles/auau200_log.sql.rtf

Number of trials averaged: 1
Total number of queries: 2918
Total size of queries: 2267497
Total query time: 132.66522622108

--------------------------------------------------------------------------------

python ./testOrbiterREST.py http://txc02.ccr.buffalo.edu
/orbiter/kdev/service/webservice/OrbiterSimulatorService.php
/operation/runfile/debug/on/file//tmp/testfiles/auau7_log.sql.rtf

RESPONSE:

Number of trials averaged: 1
Total number of queries: 2918
Total size of queries: 1362013
Total query time: 251.5587079525

python ./testOrbiterREST.py http://txc02.ccr.buffalo.edu
/orbiter/kdev/service/webservice/OrbiterSimulatorService.php
/operation/runfile/debug/on/file//tmp/testfiles/auau11_log.sql.rtf

RESPONSE:

Number of trials averaged: 1
Total number of queries: 2918
Total size of queries: 1362013
Total query time: 252.59035301208

python ./testOrbiterREST.py http://txc02.ccr.buffalo.edu
/orbiter/kdev/service/webservice/OrbiterSimulatorService.php
/operation/runfile/debug/on/file//tmp/testfiles/auau39_log.sql.rtf

RESPONSE:

Number of trials averaged: 1
Total number of queries: 2918
Total size of queries: 1362013
Total query time: 245.11031603813
```

```
python ./testOrbiterREST.py http://txc02.ccr.buffalo.edu
/orbiter/kdev/service/webservice/OrbiterSimulatorService.php
/operation/runfile/debug/on/file//tmp/testfiles/auau200_log.sql.rtf

RESPONSE:

Number of trials averaged: 1
Total number of queries: 2918
Total size of queries: 1362013
Total query time: 253.29887604713

--------------------------------------------------------------------------------

python ./testOrbiterREST.py http://128.205.41.182
/orbiter/kdev/service/webservice/OrbiterSimulatorService.php
/operation/runfile/debug/on/file//tmp/testfiles/auau7_log.sql.rtf

RESPONSE:

Number of trials averaged: 1
Total number of queries: 2918
Total size of queries: 1362013
Total query time: 255.44179987907

python ./testOrbiterREST.py http://128.205.41.182
/orbiter/kdev/service/webservice/OrbiterSimulatorService.php
/operation/runfile/debug/on/file//tmp/testfiles/auau11_log.sql.rtf

RESPONSE:

Number of trials averaged: 1
Total number of queries: 2918
Total size of queries: 1362013
Total query time: 255.52873706818

python ./testOrbiterREST.py http://128.205.41.182
/orbiter/kdev/service/webservice/OrbiterSimulatorService.php
/operation/runfile/debug/on/file//tmp/testfiles/auau39_log.sql.rtf

RESPONSE:

Number of trials averaged: 1
Total number of queries: 2918
Total size of queries: 1362013
Total query time: 255.93114089966

python ./testOrbiterREST.py http://128.205.41.182
/orbiter/kdev/service/webservice/OrbiterSimulatorService.php
/operation/runfile/debug/on/file//tmp/testfiles/auau200_log.sql.rtf

RESPONSE:

Number of trials averaged: 1
```

```
Total number of queries: 2918
Total size of queries: 1362013
Total query time: 254.40869903564

--------------------------------------------------------------------------------

python ./testOrbiterREST.py http://txc02.ccr.buffalo.edu
/orbiter/kdev/service/webservice/OrbiterSimulatorService.php
/operation/runfile/noop/on/debug/on/file//tmp/testfiles/auau7_log.sql.rtf

RESPONSE:

Number of trials averaged: 1
Total number of queries: 2918
Total size of queries: 250269
Total query time: 117.55900597572

python ./testOrbiterREST.py http://txc02.ccr.buffalo.edu
/orbiter/kdev/service/webservice/OrbiterSimulatorService.php
/operation/runfile/noop/on/debug/on/file//tmp/testfiles/auau11_log.sql.rtf

RESPONSE:

Number of trials averaged: 1
Total number of queries: 2918
Total size of queries: 250263
Total query time: 117.02989602089

python ./testOrbiterREST.py http://txc02.ccr.buffalo.edu
/orbiter/kdev/service/webservice/OrbiterSimulatorService.php
/operation/runfile/noop/on/debug/on/file//tmp/testfiles/auau39_log.sql.rtf

RESPONSE:

Number of trials averaged: 1
Total number of queries: 2918
Total size of queries: 250238
Total query time: 117.36743593216

python ./testOrbiterREST.py http://txc02.ccr.buffalo.edu
/orbiter/kdev/service/webservice/OrbiterSimulatorService.php
/operation/runfile/noop/on/debug/on/file//tmp/testfiles/auau200_log.sql.rtf

RESPONSE:

Number of trials averaged: 1
Total number of queries: 2918
Total size of queries: 250250
Total query time: 117.95971107483

--------------------------------------------------------------------------------

python ./testOrbiterREST.py http://128.205.41.182
/orbiter/kdev/service/webservice/OrbiterSimulatorService.php
```

```
/operation/runfile/noop/on/debug/on/file//tmp/testfiles/auau7_log.sql.rtf

RESPONSE:

Number of trials averaged: 1
Total number of queries: 2918
Total size of queries: 250260
Total query time: 114.50704908371

python ./testOrbiterREST.py http://128.205.41.182
/orbiter/kdev/service/webservice/OrbiterSimulatorService.php
/operation/runfile/noop/on/debug/on/file//tmp/testfiles/auau11_log.sql.rtf

RESPONSE:

Number of trials averaged: 1
Total number of queries: 2918
Total size of queries: 250376
Total query time: 115.08740878105

python ./testOrbiterREST.py http://128.205.41.182
/orbiter/kdev/service/webservice/OrbiterSimulatorService.php
/operation/runfile/noop/on/debug/on/file//tmp/testfiles/auau39_log.sql.rtf

RESPONSE:

Number of trials averaged: 1
Total number of queries: 2918
Total size of queries: 250293
Total query time: 115.82376503944

python ./testOrbiterREST.py http://128.205.41.182
/orbiter/kdev/service/webservice/OrbiterSimulatorService.php
/operation/runfile/noop/on/debug/on/file//tmp/testfiles/auau200_log.sql.rtf

RESPONSE:

Number of trials averaged: 1
Total number of queries: 2918
Total size of queries: 250297
Total query time: 114.05878996849

--------------------------------------------------------------------------------

python ./testOrbiterREST.py http://txc02.ccr.buffalo.edu
/orbiter/trunk/service/webservice/OrbiterSimulatorService.php
/operation/runfile/debug/on/file//tmp/testfiles/auau7_log.sql.rtf

RESPONSE:

Number of trials averaged: 1
Total number of queries: 2918
Total size of queries: 1362013
Total query time: 269.27799105644
```

```
python ./testOrbiterREST.py http://txc02.ccr.buffalo.edu
/orbiter/trunk/service/webservice/OrbiterSimulatorService.php
/operation/runfile/debug/on/file//tmp/testfiles/auau11_log.sql.rtf

RESPONSE:

Number of trials averaged: 1
Total number of queries: 2918
Total size of queries: 1362013
Total query time: 253.02259516716

python ./testOrbiterREST.py http://txc02.ccr.buffalo.edu
/orbiter/trunk/service/webservice/OrbiterSimulatorService.php
/operation/runfile/debug/on/file//tmp/testfiles/auau39_log.sql.rtf

RESPONSE:

Number of trials averaged: 1
Total number of queries: 2918
Total size of queries: 1362013
Total query time: 253.49148702621

python ./testOrbiterREST.py http://txc02.ccr.buffalo.edu
/orbiter/trunk/service/webservice/OrbiterSimulatorService.php
/operation/runfile/debug/on/file//tmp/testfiles/auau200_log.sql.rtf

RESPONSE:

Number of trials averaged: 1
Total number of queries: 2918
Total size of queries: 1362013
Total query time: 254.53757119179
```

2.10 Task 10. Write Progress and Final Reports

100% of progress and continuation reports are complete.

3 Products Developed

3.1 Presentations and Publications

In conjunction with this effort we have produced a number of presentations and publications, listed below. We have attached these presentations and publications in the following pages.

Green, Mark L. "CWS4DB: A Customizable Web Service for Efficient Access to Distributed Nuclear Physics Relational Databases." SBIR/STTR Exchange Meeting. September 13-14, 2010, Gaithersburg MD, USA.

Ruby, Catherine L., Green, Mark L., and Miller, Stephen D. (2010) "Orbiter Commander: A Flexible Application Framework for Service-Based Scientific Computing Environments," Grid Computing Environments Workshop, 2010. GCE'10 Nov. 14, 2010, New Orleans LA, USA.

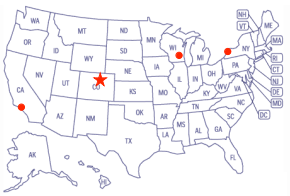
Green, Mark L. "CWS4DB: A Customizable Web Service for Efficient Access to Distributed Nuclear Physics Relational Databases." SBIR/STTR Exchange Meeting. October 24-25, 2011, Gaithersburg MD, USA.

CWS4DB: A Customizable Web Service for Efficient Access to Distributed Nuclear Physics Relational Databases

FY 2008 SBIR Phase II Proposal Award Number: DE-FG02-07ER84757

Mark L. Green, PI

Tech-X Corporation, Buffalo Office
Systems Integration Group



TECH-X CORPORATION

CWS4DB Project

A customizable Web Service for Efficient Access to Distributed Nuclear Physics Relational Databases
DOE NP Phase I and II – Manouchehr Farkhondeh

Tech-X: Mark L. Green (PI), Catherine L. Ruby, Krishna Kantam, Srilakshmi Ramireddy

Need: As the size of NP data grows and the collaborative nature of HENP experiments increases, the ability to access differently organized relational databases remotely, efficiently, and yet in a user-friendly and interoperable manner is becoming very important.

Partners: Jerome Lauret (STAR project at BNL), Kate Keahey (Nimbus project at ANL), Doug Olson (Open Science Grid), Alexandre Vaniachine (ATLAS project ANL/ CERN)

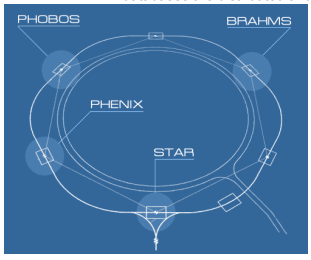
DOE Beneficiaries: Nuclear and high energy physics communities, national laboratories, and collaborative projects

Commercial Beneficiaries: Companies requiring efficient web service access to distributed relational databases with high-level database and user APIs

TECH-X CORPORATION

Problem Identification

- The importance of this project comes from the fact that a large fraction of the ever-growing data generated by Nuclear Physics (NP) experiments is stored in relational databases. For example:
  - The BNL Relativistic Heavy Ion Collider (RHIC) supports STAR (Solenoidal Tracker at the at the RHIC) which composed of 52 institutions from 12 countries, with a total of 529 collaborators;
  - relational databases (such as Condition databases, Calibration databases, and Geometry databases) are heavily used in the STAR experiment;
  - while accessing data in such databases is convenient and available for local users who are familiar with a particular database, the situation becomes more complicated when the databases are distributed and heterogeneous.

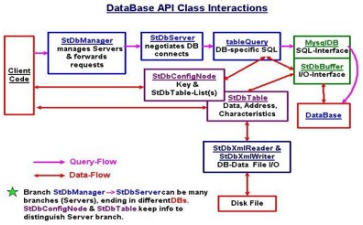


- Tech-X therefore proposes a system to overcome the outlined challenges by bridging relational databases with high-level APIs through Web services.
  - In particular, the distributed and heterogeneous nature of the databases will be addressed by creating Web services in the Orbiter Federation Service Oriented Architecture (SOA), which provides mechanisms coordinating access to diversified data resources through ReST (Representational State Transfer) services, caching, authentication, and authorization.

TECH-X CORPORATION

CWS4DB Technical Objectives

- Tech-X proposes to develop a customizable Web service for efficient access to distributed NP databases. The proposed system will consist of:
  - a generic Web service for accessing arbitrary distributed relational databases,
  - a reference client implemented at the Relativistic Heavy Ion Collider (RHIC) at Brookhaven National Laboratory (BNL), for the Solenoidal Tracker at the at the RHIC (STAR) experiment, and
  - a tool for creation of the high-level and domain-specific clients required by particular applications.
- The Phase II objectives include:
  - Take into account what was learned from the research in Phase I and extend the CWS4DB prototype into a production-quality, load-balanced, auto-caching, grid-enabled, fault-tolerant, and on-demand system.



  - Use a flexible work plan involving a separate piece of technical functionality that can be implemented in a way that can be exercised in the STAR computing environment, yet developed in a general way for application's from other NP projects.
  - ***The ultimate goal is to produce a set of software tools and services that can be easily adapted by the NP application developer.***

TECH-X CORPORATION

CWS4DB Tasks

- Task 1: Determine CWS4DB System and Load Balancing Additional Requirements and Properties (Tech-X & BNL) ☆
  - Extend the Phase I developed requirements and properties and continue prototype work with our partners.
- Task 2: Design and Implement Tiered Deployment Capabilities (Tech-X)
  - Develop a tiered deployment based protocol for the CWS4DB system.
- Task 3: Design and Implement Auto-Caching Infrastructure (Tech-X & BNL)
  - Provide a sophisticated auto-caching mechanism in order to increase the effective system performance based on work with our partners.
- Task 4: Enable Multi-Virtual Organization Role-Based Capabilities (Tech-X)
  - Develop the CWS4DB infrastructure required for user-friendly management and caching capabilities.
- Task 5: Develop Dynamic On-Demand Data Resource Access (Tech-X)
  - This on-demand service will provide a STAR MySQL database instance using the Virtual Workspaces infrastructure, Virtual Machine Computing resources, and investigate Grid deployments.

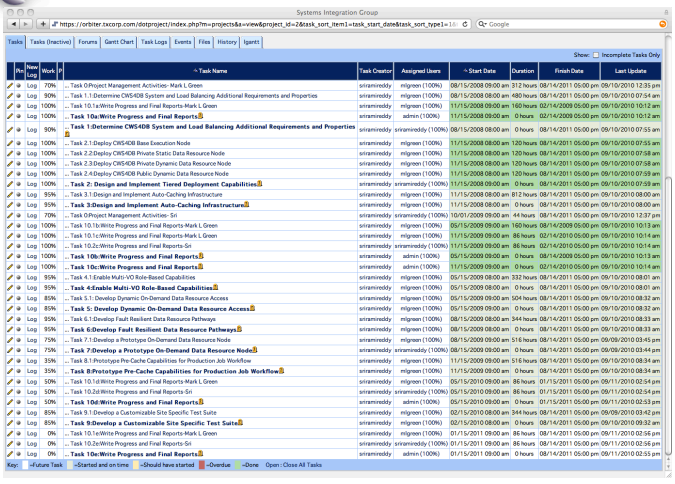
CWS4DB Tasks Continued

- Task 6: Develop Fault Resilient Data Resource Pathways (Tech-X)
  - Investigate eliminating a single point of failure for the STAR C++ API bound codes database query requests.
- Task 7: Develop a Prototype On-Demand Data Resource Node (Tech-X & BNL) ☆
  - Investigate and prototype the deployment of a on-demand data resource node to meet the dynamic data demands of the STAR collaboration.
- Task 8: Prototype Pre-Cache Capabilities for Production Job Workflow (Tech-X & BNL) ☆
  - We will provide a pathway for an authenticated and authorized user upon configuration of the CWS4DB system to execute the customizable site specific test suite for pre-caching production job queries.
- Task 9: Develop a Customizable Site Specific Test Suite (Tech-X) ☆
  - In order to deliver a high quality of service infrastructure a customizable and site specific test suite is required to validate and verify the performance and data delivery capabilities of the CWS4DB system.

Project Management

- Subversion Repositories
  - Multiple readers and committers
- Redmine, Trac, and Wiki Sites
  - Integrates ticketing system, repositories, milestones, and roadmap
- Eclipse Integrated Development Environment
  - Tracks code modifications based on Redmine and Trac tickets
- Zend Studio, Development Server, and Server
  - Commercial PHP development and enterprise level server
- Content Management System (Drupal)
  - Offsite collaborator access to project information
- Knowledgebase Manager
  - Coding best practices, design patterns, systems and integration information
- MacA&D Developer
  - Analysis and Design (A&D) with requirements management and use case development
- dotProject
  - Open source PHP based project management software

Project Status



Task: DB Timings (DONE)
Using sequences of SQL operations that are recorded from actual STAR DB usage, we evaluated database performance under load by timing numerous repetitions of these operations against local and remote databases. The sample SQL sequences are:

Name	# Operations
db-perf-test.txt	6,667
offline.auau200.full.sql	8,911
offline.dau200.full.sql	8,784
offline.pp500.full.sql	6,667

Timing results for *db-perf-test.txt*:

DB Host	# Repetitions	Avg Time (sec)
dbx.star.bnl.gov	10	921.66
orbiter.txcorp.com	10	3.42
cyber.txcorp.com	10	11.63
dbx.star.bnl.gov	20	922.02
orbiter.txcorp.com	20	3.49
cyber.txcorp.com	20	11.7
dbx.star.bnl.gov	30	898.57
orbiter.txcorp.com	30	3.61
cyber.txcorp.com	30	11.88

Timing results for *offline.auau200.full.sql*:

DB Host	# Repetitions	Avg Time (sec)
orbiter.txcorp.com	5	4.16
dbx.star.bnl.gov	5	1134.09
orbiter.txcorp.com	10	4.14
dbx.star.bnl.gov	10	1090.29
orbiter.txcorp.com	15	4.1
dbx.star.bnl.gov	15	1616.98

Timing results for *offline.dau200.full.sql*:

DB Host	# Repetitions	Avg Time (sec)
orbiter.txcorp.com	5	2.56
dbx.star.bnl.gov	5	993.93
orbiter.txcorp.com	10	2.66
dbx.star.bnl.gov	10	1256.02
orbiter.txcorp.com	15	2.66
dbx.star.bnl.gov	15	999.29

Timing results for *offline.pp500.full.sql*:

DB Host	# Repetitions	Avg Time (sec)
orbiter.txcorp.com	5	4.2
dbx.star.bnl.gov	5	921.82
orbiter.txcorp.com	10	3.52
dbx.star.bnl.gov	10	921.82
orbiter.txcorp.com	15	3.39
dbx.star.bnl.gov	15	907.23

CWS4DB Database Query Caching and Optimization

- Network bandwidth is important and depends on the last mile normally
- Database server load is minimal
- Investigate the database service payload size
- Wrote a custom ReSTful PHP database service with a JSON (JavaScript Object Notation) payload to compare with the XML payload

TECH-X CORPORATION

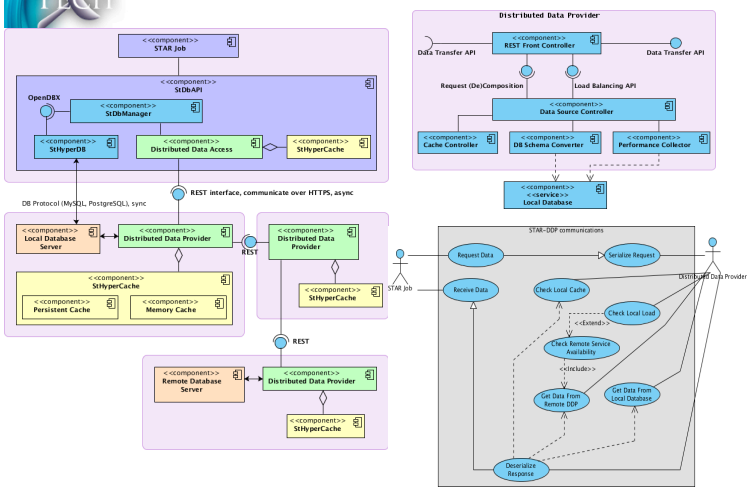
CWS4DB Database Query Caching and Optimization

- Log performance data for each SQL operation
- Calculate and log JSON and XML payload size

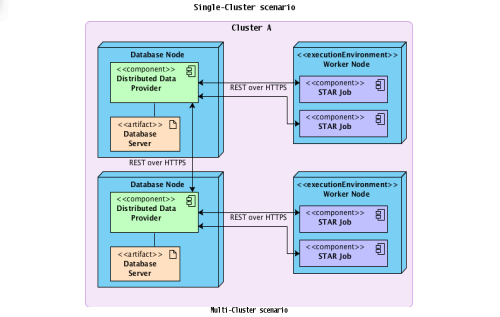
- On average over a dataset the equivalent JSON payload is 8.8 – 10.1 times smaller
- In general an order of magnitude lower bandwidth loading is required with the JSON PHP service

entry_id	query_id	json_size	xml_size	duration	timestamp
44880	5757	97	273	0.00026798248291016	2009-08-11 14:46:58
44881	5758	396	996	0.00047802925109863	2009-08-11 14:46:58
44882	5759	99	275	0.00019717216491699	2009-08-11 14:46:58
44883	5758	467	1205	0.00057792663574219	2009-08-11 14:46:58
44884	5760	94	270	0.00020289421081543	2009-08-11 14:46:58
44885	5761	3310	7498	0.0028619766235352	2009-08-11 14:46:58
44886	5762	62	190	0.00020694732666016	2009-08-11 14:46:58
44887	5763	102	278	0.00020194053649902	2009-08-11 14:46:58
44888	5764	126	312	0.00022315979003906	2009-08-11 14:46:58
44889	5765	7	63	0.00011920928955078	2009-08-11 14:46:58
44890	5766	102	278	0.00022387504577637	2009-08-11 14:46:58
44891	5767	126	312	0.00022315979003906	2009-08-11 14:46:58
44892	5768	7	63	0.00012707710266113	2009-08-11 14:46:58
44893	5769	102	278	0.00021219253540039	2009-08-11 14:46:58
44894	5770	127	313	0.00026917457580566	2009-08-11 14:46:58
44895	5771	7	63	0.00011920928955078	2009-08-11 14:46:58
44896	5772	103	279	0.00022912025451660	2009-08-11 14:46:58
44897	5773	127	313	0.00023388862609863	2009-08-11 14:46:58
44898	5774	7	63	0.00012397766113281	2009-08-11 14:46:58
44899	5775	103	279	0.00020599365234375	2009-08-11 14:46:58
44900	5776	127	313	0.00024199485778809	2009-08-11 14:46:58
44901	5777	7	63	0.00016498565673828	2009-08-11 14:46:58
44902	5778	103	279	0.0022598762512207	2009-08-11 14:46:58
44903	5779	127	313	0.00022602081298828	2009-08-11 14:46:58
44904	5780	7	63	0.00013303756713867	2009-08-11 14:46:58
44905	5781	103	279	0.00022983551025391	2009-08-11 14:46:58
44906	5782	127	313	0.00027108192443848	2009-08-11 14:46:58
44907	5783	7	63	0.00012493133544922	2009-08-11 14:46:58
44908	5784	103	279	0.00025486946105957	2009-08-11 14:46:58
44909	5785	127	313	0.00022006034851074	2009-08-11 14:46:58

CWS4DB Load Balancing Design



CWS4DB Proxy Implementation

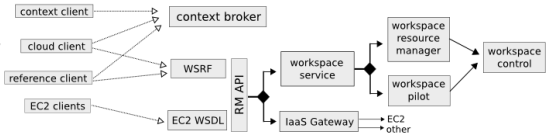


TECH-X CORPORATION

CWS4DB Cloud On-Demand Resources

- Tech-X has installed Nimbus and utilized the Nimbus client with the available science clouds in support of the STAR on-demand database service.
  - The Nimbus infrastructure provided limited upload/download bandwidth consistently.
  - The required STAR image is relatively large due to the size of the MySQL database.
  - We investigated several ways of populating the STAR database and tested query performance with our ReSTful PHP JSON database service successfully.
  - The Open Grid Services Architecture - Database Access and Integration (OGSA-DAI) XML database services could not be loaded on the Nimbus science cloud due to memory constraints.
  - We are still investigating utilizing Eucalyptus and the cloud enabled MySQL database Drizzle

Nimbus components (Keahey, ANL)



CWS4DB Summary

File Name : star.pp500.full.sql

https://cyber.txcorp.com/orbiter/service/star/OrbiterStarSimulatorService.php
/cache/off/
/format/XML/
/host/local/
/file//tmp/testfiles/star.pp500.full.sql/
/address/http://64.240.154.24/orbiter/service/star/

Result:

Number of trials averaged: 1
Total number of queries: 6549
Total size of queries: 38,926,201 bytes
Total query time: 76.9 seconds
Total query rate: 85.1 query/second.

CWS4DB Summary

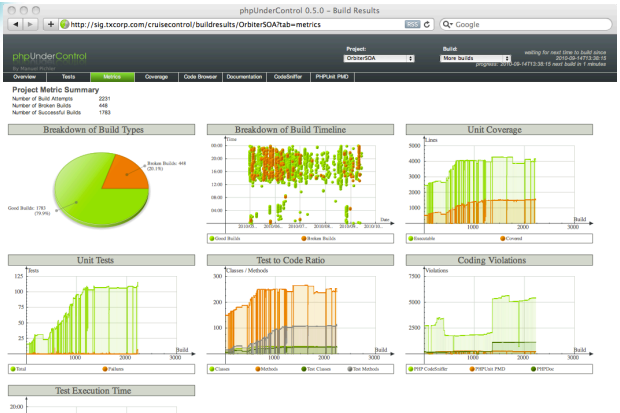
New class files and services developed to accomplish the above tasks:

-- orbiterAutoLoader.php (150)
-- OrbiterAttributeParser.class.php (147)
-- OrbiterCacheFileService.php (723)
-- OrbiterCacheManager.class.php (236)
-- OrbiterDatabaseConnection.class.php (212)
-- OrbiterErrorHandler.class.php (509)
-- OrbiterErrorHandlerMessageService.class.php (526)
-- OrbiterMailer.class.php (187)
-- OrbiterMasterSlaveDatabaseValidationService.class.php (439)
-- OrbiterQueryDbConnectionStringStarService.class.php (467)
-- OrbiterQueryDbLoadBalancerStarService.class.php (399)
-- OrbiterRestAuth.class.php (655)
-- OrbiterServiceAttributes.class.php (132)
-- OrbiterServiceLogger.class.php (234)
-- OrbiterStarQueryService.class.php (530)
-- OrbiterStarSimulatorService.php (489)

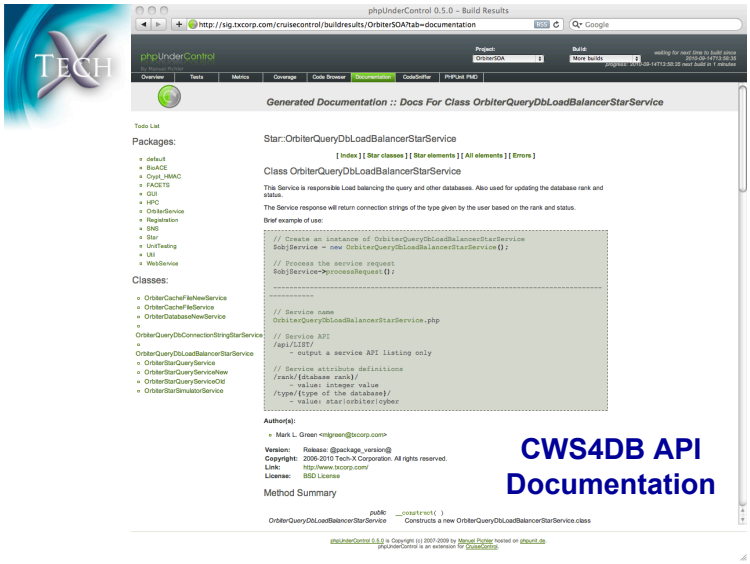
Services developed:
-- OrbiterCacheFileService.php
-- OrbiterQueryDbConnectionStringStarService.php
-- OrbiterQueryDbLoadBalancerStarService.php
-- OrbiterQueryDBService.php
-- OrbiterStarQueryService.php
-- OrbiterStarSimulatorService.php

Unit Test scripts developed:

-- OrbiterAttributeParserStubTest.php
-- OrbiterAttributeParserTest.php
-- OrbiterServiceTestSuite.php
-- OrbiterAutoLoaderTest.php
-- OrbiterCacheManagerFileTest.php
-- OrbiterCacheManagerTest.php
-- OrbiterDatabaseConnectionMasterTest.php
-- OrbiterDatabaseConnectionSlaveTest.php
-- OrbiterDatabaseConnectionTest.php
-- OrbiterDataProcessManagerTest.php
-- OrbiterErrorHandlerEmailTest.php
-- OrbiterErrorHandlerErrorDetailedTest.php
-- OrbiterErrorHandlerLogTest.php
-- OrbiterErrorHandlerNotifyTest.php
-- OrbiterErrorHandlerSampleTest.php
-- OrbiterErrorHandlerShowContentsOutputTest.php
-- OrbiterMailerTest.php
-- OrbiterQueryDbLoadBalancerStarServiceTest.php
-- OrbiterRestAuthTest.php
-- Services_JSONTest.php

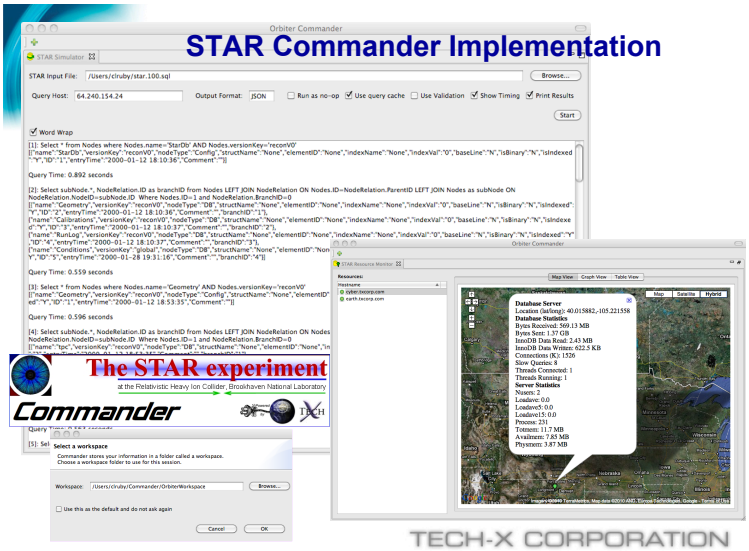


CWS4DB Continuous Integration



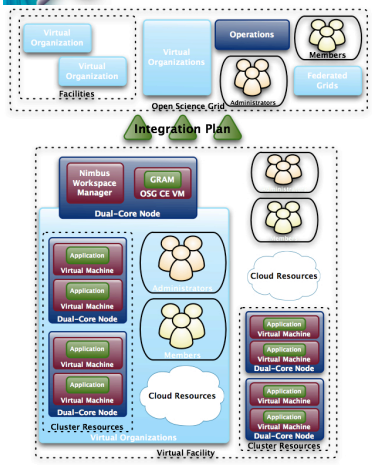
CWS4DB API Documentation

STAR Commander Implementation



TECH-X CORPORATION

Future Directions

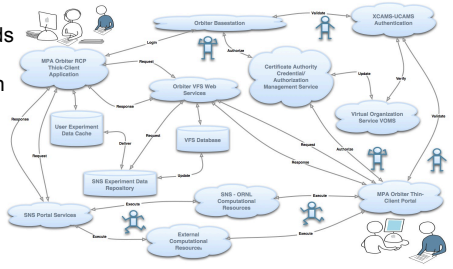


- Integrate On-Demand Application Resources (O-DAR) within the Open Science Grid.
- This is a new type of OSG virtual facility that can be used for cycle scavenging usage on hardware that is idle or migrated out of a production environment and might not even have OSG stack installed.
- It can represent a lightweight method of deploying OSG worker nodes and building more capacity for scientific application usage.
- Will support NP, HEP, Neutron Science, etc.

TECH-X CORPORATION

Orbiter Federation SOA via ReSTful Services

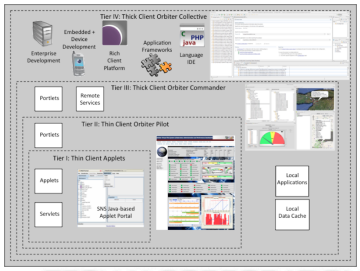
- Orbiter Infrastructure serves capabilities via ReSTful web services
- Services are standards-based and are scalable, reusable, and extensible
- Robust security standards using access keys and private-key authentication
- Reusable to ensure consistent and reliable Quality of Service



TECH-X CORPORATION

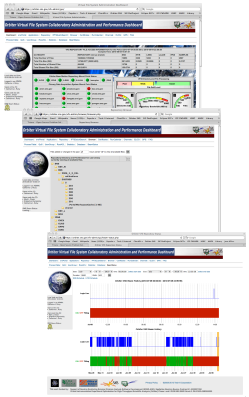
Orbiter Multitier Portal Architecture (MPA)

- Framework for delivering capabilities to thin- and thick-clients using the Orbiter Federation ReSTful SOA
- Flexible and re-usable architecture for developing capabilities for thin web clients and thick local clients
- Comprised of four tiers:
  - Orbiter Federation SOA
    - Thin-Client Applets
  - Orbiter Pilot
    - Thin-Client Portlets
  - Orbiter Commander
    - Thick-Client Applications
  - Orbiter Collective
    - Thick-Client Eclipse IDE



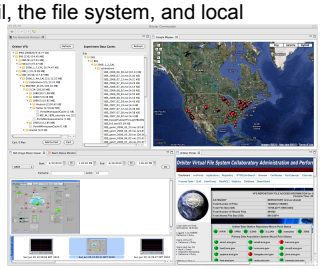
Orbiter Pilot – Thin Client

- Built on top of the Orbiter Federation SOA
- Tier II of the Orbiter Multitier Portal Architecture
- Accessible to users with accounts and internet access (via a web browser)
- Build upon the services provided by the Orbiter SOA infrastructure
- Capabilities are seamlessly integrated using these well-defined ReSTful web services



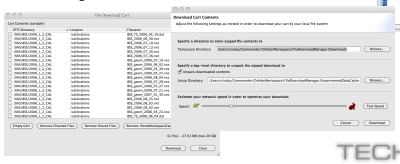
Orbiter Commander – Thick Client

- Built on top of the Orbiter Federation SOA
- Integrates Orbiter Pilot
- Tier III of the Orbiter Multitier Portal Architecture
- Run locally on user work stations or personal computers
- Uses Eclipse RCP (Rich Client Platform) to deliver a robust and powerful GUI to the end user, also allows Commander to integrate with other local resources like e-mail, the file system, and local applications.
- Build upon the services provided by the Orbiter Federation SOA infrastructure
- Allows users to run complex simulations or computationally-intensive tasks on their local machines, relieving Quality of Service concerns on web service providers



Orbiter Commander – Thick Client (continued)

- Atomic capabilities are provided as *modules* that can be installed as needed from a central module repository
- The Orbiter Federation ReSTful SOA provides robust access to diverse capabilities, such as:
  - Multi-threaded streaming downloads of repository files
  - Live status monitoring of the beam
  - Slideshows of instrument application screenshots
  - Organization of modules into "*Suites*"



Orbiter Collective (future capabilities)

- Modules will be continuously added to Commander to provide new capabilities, including:
  - A *collaboratory* providing live chat and data sharing capabilities
  - Opportunistic file slicing to support the retrieval and management of very large data sets
  - Real-time and offline scientific data visualization capabilities
  - Integration with other open-source tools such as data analysis and workflow management for computational, data movement, and visualization jobs
  - Support for 3rd party module contributions as well as user integrated applications (MPA Tier IV Orbiter Collective)

Orbiter Federation SOA: Python Client Service Access Example

```
#!/usr/bin/python
import os, sys, base64, hmac, commands, time
from hashlib import sha1 as sha
from urllib import urlencode
from urllib import urlopen
from urllib import quote_plus

myhome = os.environ.get('HOME')
os.environ['TZ']='GMT'
time.tzset()

idfile = open(myhome + "/.orbiter/my.id")
ACCESS_KEY = idfile.read().strip()
idfile.close()
keyfile = open(myhome + "/.orbiter/user.key")
PRIVATE_KEY = keyfile.read()
keyfile.close()

URI = sys.argv[1]
EXPIRES = str(int(time.mktime(time.localtime(time.time()+60))))
str = URI + '/OrbiterAccessKeyId/' + ACCESS_KEY + '/Expires/' + EXPIRES
SIGNATURE = base64.b64encode(hmac.new(PRIVATE_KEY, str, sha).digest()).strip()
print urlopen(str + '/Signature/' + SIGNATURE, params).read()
```


Related Publications

- Lynch, V. E., Cobb, J. W., Green, M. L., Kohl, J. A., Miller, S. D., Ren, S., Smith, B., Vazhkudai, S. S.; "Experience with Remote Job Execution", NOBUGS 2008 Conference, Australian Nuclear Science and Technology Organization (ANSTO), Sydney, Australia, 3-5 November 2008 in proceedings.
- Green, Mark L.; Alexander, David A.; Pundaleeka, Roopa; Matykiewicz, James. "Automatic Certificate Based Account Generation and Secure AJAX Calls in a Grid Portal", Grid Computing Environments Workshop, 2008. GCE'08 Volume , Issue , 12-16 Nov. 2008 Page (s):1 - 8 DOI 10.1109/GCE.2008.4738444
- Green, Mark L, Miller, Stephen D, Vazhkudai, Sudharshan S, Trater, James R; "Doing Your Science While You're in Orbit", International Conference on Neutron Scattering 2009, Knoxville, TN, 3-7 May 2009. Submitted to Journal of Physics Conference Series.
- Miller, Stephen D., Herwig, Kenneth W., Ren, Shelly, Vazhkudai, Sudharshan S., Jemian, Pete, Luitz, Steffen, Salnikov, Andrei A., Gaponenko, Igor, Proffen, Thomas, Lewis, Paul, Green, Mark L.; "Data Management and Science at DOE BES User Facilities - Past, Present, and Future", SciDAC 2009, San Diego, CA, 14-18 June 2009.
- Green, Mark L. and Miller, Stephen D. (2007) "Multitier Portal Architecture for Thin- and Thick-client Neutron Scattering Experiment Support." Grid Computing Environments (GCE) workshop, Nov. 11-12, 2007, Reno, NV, http://casci.rit.edu/proceedings/gce2007 .
- Green, Mark L., Alexander, David, Pundaleeka, Roopa, and Matykiewicz, James (2008) "Automatic Certificate Based Account Generation and Secure AJAX Calls in a Grid Portal." Grid Computing Environments Workshop, 2008. GCE '08, Nov. 12-16, 2008, pages 1 – 8, Austin, TX

Related Posters

- Green, Mark L., Miller, Stephen D., Ren, Shelly X., Peterson, Peter F.; "Scalable Web Services for Experiment Repository Virtual File System Access", NOBUGS 2008 Conference, Australian Nuclear Science and Technology Organisation (ANSTO), Sydney, Australia, 3-5 November 2008.
- Green, Mark L., Miller, Stephen D., Cobb, John W., Trater, Jim R.; "Enlightened Cybersecurity to Enable Collaborative Research Using Virtual Organizations", NOBUGS 2008 Conference, Australian Nuclear Science and Technology Organisation (ANSTO), Sydney, Australia, 3-5 November 2008.
- Lynch, V. E., Cobb, J. W., Green, M. L., Kohl, J. A., Miller, S. D., Ren, S., Smith, B., Vazhkudai, S. S.; "Experience with Remote Job Execution", NOBUGS 2008 Conference, Australian Nuclear Science and Technology Organisation (ANSTO), Sydney, Australia, 3-5 November 2008.
- Miller, S.D., Kohl, J.A., Vazhkudai, S.S., Green, M.L.; "NSSD Neutron Science Portal architecture", NOBUGS 2008 Conference, Australian Nuclear Science and Technology Organisation (ANSTO), Sydney, Australia, 3-5 November 2008.
- Green, Mark L, Miller, Stephen D, Vazhkudai, Sudharshan S, Trater, James R; "Doing Your Science While You're in Orbit", International Conference on Neutron Scattering 2009 (ICNS 2009), Knoxville, TN, 3-7 May 2009.
- Miller, Stephen D., Herwig, Kenneth W., Ren, Shelly, Vazhkudai, Sudharshan S., Jemian, Pete, Luitz, Steffen, Salnikov, Andrei A., Gaponenko, Igor, Proffen, Thomas, Lewis, Paul, Green, Mark L.; "Data Management and Science at DOE BES User Facilities - Past, Present, and Future", SciDAC 2009, San Diego, CA, 14-18 June 2009.

Related Presentations

- Green, Mark L. and Miller, Stephen D.; "Orbiter Service Oriented Architecture at SNS", DANSE Developer Meeting, CalTech, Pasadena, CA, 24-27 August 2008.
- Green, Mark L; "A Multi-tiered Portal Architecture Overview: Emphasizing the Orbiter Thick-client Tier", NOBUGS 2008 Conference, Australian Nuclear Science and Technology Organisation (ANSTO), Sydney, Australia, 3-5 November 2008.
- Green, Mark L.; "A Service Oriented Architecture for the SNS", DANSE Developer Meeting, CalTech, Pasadena, CA, 15 December 2008.
- Green, Mark L. and Miller, Stephen D.; "Demonstration of the Orbiter Service Oriented Architecture at SNS", DANSE Developer Meeting, CalTech, Pasadena, CA, 25-29 January 2009.
- Miller, S.D. and Green, Mark L.; "Toward Federated Services and Infrastructure for SNS Researchers", Composing Collaboratories Meeting, Chicago, IL, 24-26 February 2009.
- Green, Mark L.and Lauret, Jerome; "STAR & Virtualization, looking beyond: Integrating Scientific, Grid, and Cloud Computing Infrastructures", Open Science Grid All Hands Meeting, Virtual Technology Workshop, LIGO Livingston Observatory, Louisiana, 2-5 March 2009.
- Miller, S.D. and Green, Mark L.; "Current and Future Data Intensive Computing at DOE BES User Facilities", Workshop on Enabling Data-Intensive Computing: from Systems to Applications, Pittsburgh, PA, 30-31 July 2009.

Sponsored Workshop

- Green, Mark L.; "Orbiter Workshop for DANSE Project Integration with SNS", Oak Ridge National Laboratory, Spallation Neutron Source, JICS Auditorium, Oak Ridge, TN, 30 April 2009 – 1 May 2009.

For More Information

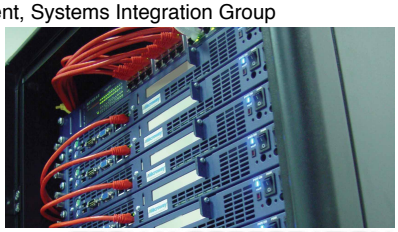
Contact:

Mark L. Green, Vice President, Systems Integration Group

716-204-8690

mlgreen@txcorp.com

http://www.txcorp.com



Orbiter Commander: A Flexible Application Framework for Service-Based Scientific Computing Environments

Catherine L. Ruby and Mark L. Green
Systems Integration Group
Tech-X Corporation
Williamsville, New York 14221
Email: clruby@txcorp.com, mlgreen@txcorp.com

Stephen D. Miller
Spallation Neutron Source
Oak Ridge National Lab
Oak Ridge, Tennessee 37831-6475
Email: millersd@ornl.gov

***Abstract*—Gateway computing environments face several challenges in providing robust, scalable, and sustainable capabilities to a wide range of users. Principles of encapsulation and cohesion have been applied in emerging trends of application framework development, where modular designs and abstraction layers allow these systems to remain flexible and agile as requirements evolve over time.**

Orbiter Commander is a modular and extensible application framework that leverages the Orbiter Federation Service Oriented Architecture to deliver fast and secure capabilities in an Eclipse RCP desktop application. Commander provides suites of modules that can be seamlessly delivered to end users on multiple platforms, enabling rapid component development through a flexible design and well-defined extension points. This paper presents our collaboration with the Spallation Neutron Source Neutron Experiment and Theory Hub (NExTHUB) and the Solenoidal Tracker at the at the RHIC (STAR) experiment, two suites of capabilities tailored to serve the needs of users at Oak Ridge National Laboratory and Brookhaven National Laboratory.

***Index Terms*—Gateway Computing Environments, Graphical User Interfaces, Application frameworks, Scalability, Reusability, Client-server architectures, Integrated Development Environments, SOA.**

I. Introduction

Best practices in software engineering have trended toward the development of application frameworks as systems grow in size and complexity. That is, principles of cohesion and encapsulation from Object Oriented Programming (OOP) have been leveraged to produce problem-solving environments that promote the reuse of a method of solving a problem, rather than the reuse of specific solutions. Application frameworks organize their capabilities in well-defined abstraction layers and provide points for extending functionality for future development. These structured frameworks strive to reduce software costs by minimizing design, development, and testing time.

Framework-based development is becoming increasingly common in industry today. The success of Apple's iPhone Developer Program [1] is an example of the popularity and commercial viability of this model. Apple's flexible API is used to provide a wide range of applications, allowing the users of Apple's hardware to customize their devices for their individual use. Google's Android [2] uses a similar model. By allowing modular content to be produced at-will by third-party developers, Apple and Google can focus their efforts on maintaining their infrastructure while ensuring that the user-visible capabilities are driven by popular demand.

Application frameworks are becoming increasingly common among gateway computing environments as well. These environments must address several challenges as they integrate disparate capabilities and technologies across distributed systems and deliver these capabilities to a wide variety of users across many scientific disciplines. These environments must also deliver these services securely while producing scalable and maintainable implementations. Framework-based development mitigates many of these challenges by promoting component reuse and abstraction layers in order to remain agile in response to ever-changing requirements. Initiatives such as the Open Grid Computing Environments (OGCE) [3] and the GridSphere Project [4] use these methodologies to produce frameworks for gateway computing environments [5].

This paper presents Orbiter Commander, an Eclipse Rich Client Platform (RCP) [6] desktop application framework. Commander provides a robust and highly customizable user interface built on the Orbiter Federation Service Oriented Architecture (SOA) [7], [8], while also providing the flexibility for offline computing and integration with other local applications. The following section discusses the Orbiter Federation SOA and how it provides robust services to support Commander. Section 3 discusses the Multitier Portal Architecture (MPA) [9], the architecture that serves as the foundation of the Commander framework. Section 4 describes the design of the Commander framework and its modular capabilities that support flexibility and rapid component development. Sections 5 and 6 present Commander's NExTHUB and STAR Suites, two sets of capabilities that support science at Oak Ridge National Lab and Brookhaven National Lab. Section 7 includes related work, and section 8 presents conclusions and future directions.

II. Orbiter Federation SOA

Commander is built on a three-tier client-server architecture consisting of a Data Tier, Logic Tier, and Presentation Tier. These tiers support scalability and flexibility by partitioning system responsibilities and providing abstraction layers with well-defined interfaces [10]. The Data and Logic Tiers are represented by Orbiter Federation, a Service Oriented Architecture (SOA) that provides functionality through remote services.

Orbiter's MySQL [11] database uses master/slave replication to ensure fault-tolerance and reliability. This database holds the basic functional data for Orbiter, including service logs, user records, and user certificates. Using its database Orbiter provides federated user access and identity management, as well as data for additional functionality such as virtual file management and resource monitoring.

Orbiter services are implemented as Representational State Transfer (RESTful) [12] web services that deliver functionality through a well-defined API. These services employ robust security standards including SSL [13] and signed requests that ensure client identities, the integrity of their RESTful calls, and the privacy of their transmissions. An overview of the Orbiter Federation SOA is featured in Figure 1.

At the time of this writing, Orbiter provides 27 services implemented as 66 PHP [14] classes that connect to the Data Tier to provide functionality to their clients. Service calls to Orbiter Federation SOA use the following format:

$$https://\{ServiceProvider\}/\{ResourceAddress\}/\{Attributes\}/\{ID\}/\{ExpirationTime\}/\{Signature\}$$

ServiceProvider: The service provider that hosts the Orbiter Federation SOA services.
ResourceAddress: The service endpoint, implemented as a PHP service stub and underlying object oriented implementation.
Attributes: The arguments to the service, using "/" separators.

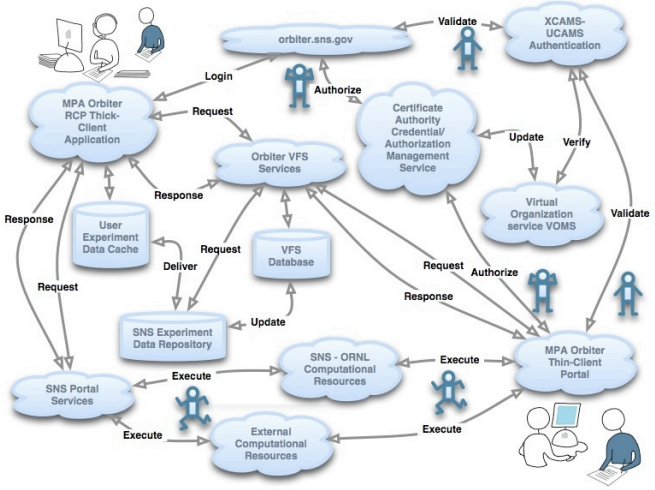


Fig. 1. Orbiter SOA Overview

ID: The requestor's Orbiter Access Key ID. This unique identifier binds a service request to a registered Orbiter user.
ExpirationTime: The expiration time for the service request in GMT epoch time (seconds).
Signature: The digital signature of the request.

The last three elements of an Orbiter service request are what ensure the security of Orbiter service transactions. Users are provided with an RSA [15] private key upon registering with the system, which is used to digitally sign their requests[1]. Signatures are calculated on the request's canonical string resource identifier, which includes the hostname, resource address, attributes, Orbiter access key identifier, and expiration time. The signature is appended to the end to complete the call. Each Orbiter RESTful POST operation is signed in a similar manner ensuring the POST data integrity.

This method of signing Orbiter service requests ensures the identity of the client as well as the integrity of the data. Orbiter Federation services use the RSA private key bound to the given Orbiter access key id to re-sign the request, comparing the provided signature with the calculated one. If these signatures don't match then Orbiter rejects the call outright, and will also reject a call that arrives more than 60 seconds[2] after the client-provided expiration time. Because the full service call including the timestamp is included in both the client-side and server-side calculation of the signature, a malicious third party cannot alter or replay a captured message as an authorized user.

These security measures, along with the use of SSL for secure communications, has allowed the Orbiter Federation SOA to pass the Oak Ridge National Lab security audit for usage and deployment at their site. These measures address 8 of the 9 top security threats facing web services as identified by Web Services Interoperability Organization [17], [18], which include:

Message alteration: Attackers cannot alter an Orbiter request without breaking the RSA SHA hash signature. Orbiter will reject a request that does not match canonical string signed resource identifier for the specified Orbiter access key ID.

Loss of confidentiality: The SSL protocol ensures that Orbiter service transactions are handled privately and provides transport-level encryption.

Falsified messages: Secure Orbiter services cannot be reached without a signed canonical string resource identifier that matches the signature for the specified Orbiter Federation SOA resource address.

Man in the middle: The SSL protocol prevents an attacker from reviewing requests and responses send securely between the Orbiter Federation SOA web services and their clients.

Principal spoofing: The Orbiter infrastructure is the only provider of valid Orbiter access key identifiers and RSA private keys that are authorized to use Orbiter Federation SOA secure web services.

Forging claims: Attackers cannot create valid Orbiter Fed-

[1]Signatures are produced using HMAC SHA [16] encoding.
[2]This value is configurable on the client side and has per-request granularity.

eration SOA service requests without obtaining an Orbiter access key identifier and valid RSA private key from the Orbiter Federation SOA authentication/authorization infrastructure.

Replay of message: Attackers cannot repeat a RESTful request to secure Orbiter Federation services, as subsequent identical requests will be rejected. Attackers cannot alter the user-provided expiration time without breaking the RSA signature.

Replay of message parts: An Orbiter RESTful service request is not complete without a valid signature that is applied to all other message parts. Attackers cannot construct a new request from any part of a previous request without altering the service request canonical string resource identifier and generating a valid signature.

Denial of service: The denial of service propensity is greatly reduced by the listed security measures in place at the current time within the Orbiter Federation SOA. Furthermore, more specific measures are planned which will ban specific offending IP addresses to further reduce the threat.

The Orbiter Federation SOA is able to provide these security measures without a significant impact on quality of service or turnaround time. An "empty" service with full validation, authorization, and connection to the underlying data layer has been timed at $1x10^{-6}$ seconds. The average Orbiter Federation full service call has been observed to take roughly $2x10^{-2}$ seconds to complete. The Orbiter Virtual File System (VFS), built on Orbiter Federation services, has fielded 31.8 million of these secure service requests since February of 2010.

The flexible and secure infrastructure of the Orbiter Federation SOA allows it to provide robust Software-as-a-Service (SaaS) capabilities to its clients. Orbiter is capable of providing complex objects, dynamic charts, raw data, and data files, forming a solid foundation of valuable resources upon which end-user interfaces can be built. These services enable the Multitier Portal Architecture, a layered approach to building scalable and robust end-user applications. This architecture is discussed in the next section.

III. Multitier Portal Architecture

The Multitier Portal Architecture (MPA) implements the Presentation Tier of Orbiter's three-tier client-server architecture. The MPA addresses challenges in providing scalable and sophisticated user interfaces by using the Orbiter Federation SOA to build increasingly complex and customizable applications. This layered approach maximizes the reuse of the developed infrastructure and services in successive tiers. An overview of the MPA is included in Figure 2.

Tier I of the MPA is implemented by the Orbiter Federation Service Oriented Architecture and thin-client applets that utilize the portal server for file transfers, visualization caching, and file meta-data transfers. This browser-based thin-client applet allows lightweight user access to Orbiter Federation services.

Orbiter Pilot implements tier II of this architecture, providing a thin-client and portlet layer of the MPA for accessing Orbiter services. Pilot provides a number of capabilities such as a dashboard for reviewing overall system usage and services for virtual file system browsing and downloads. A screenshot of the Orbiter Pilot dashboard is featured in Figure 3.

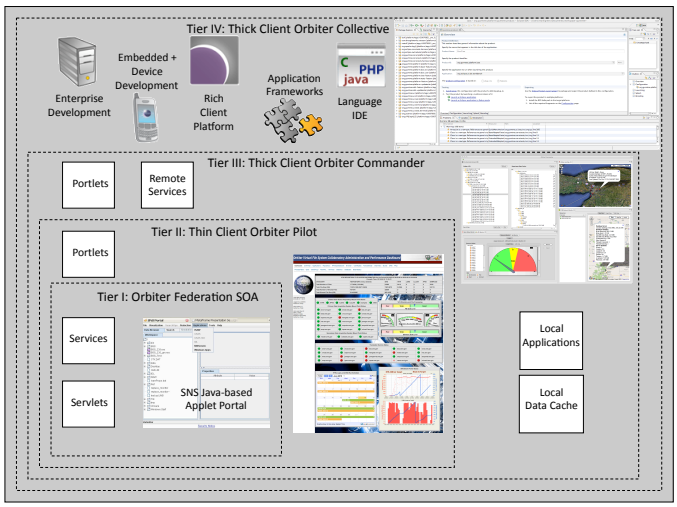


Fig. 2. Multitier Portal Architecture

Orbiter Commander implements tier III of this architecture by providing a sophisticated desktop application to end-users. Like in the lower tiers of the MPA Commander utilizes the Orbiter Federation SOA to provide services to end users, while providing the capability for offline computing and integration with other local desktop applications. Commander implements its capabilities as modular components that produce a highly customizable plug-and-play end-user application.

The flexible and modular framework provided by Orbiter Commander and the underlying Orbiter Federation SOA enables tier IV. This tier, the Orbiter Collective, is a collaboratory for sharing information and data and for applying Orbiter capabilities in new ways. That is, Commander's growing toolkit and well-defined extension points allow for the rapid development of new Commander capabilities either in-house

Fig. 3. Orbiter Pilot Dashboard

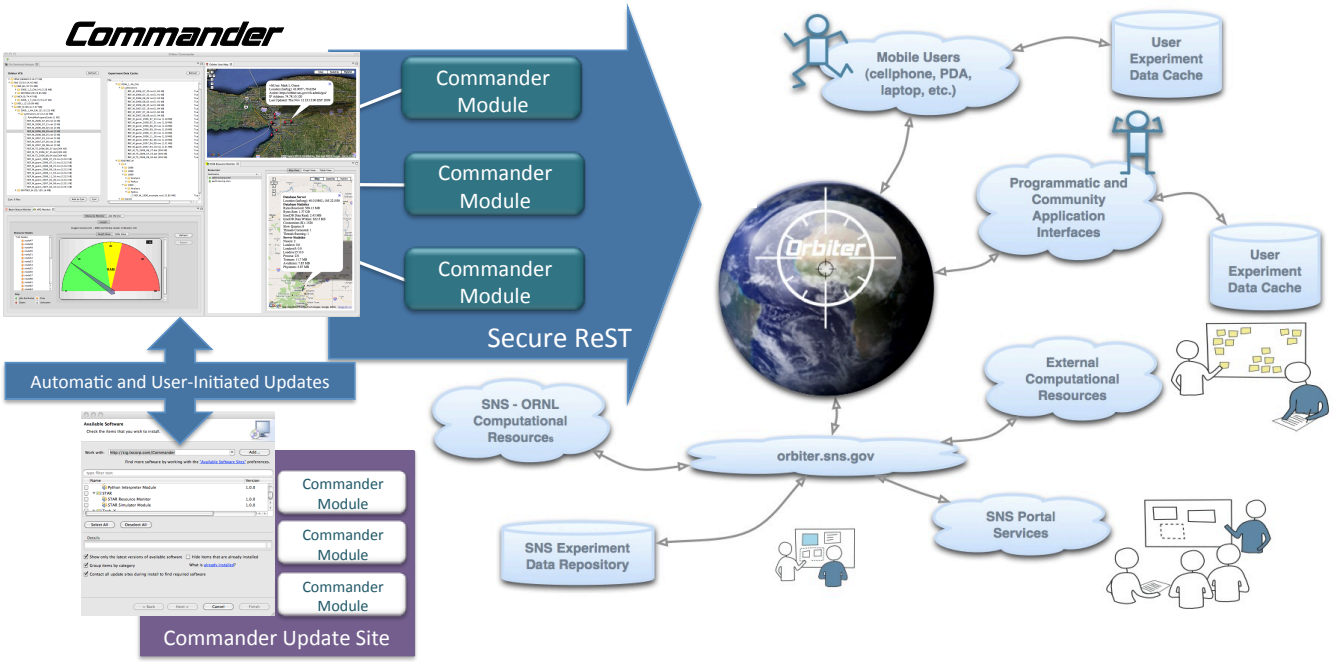


Fig. 4. Orbiter Commander Framework Overview

or alternatively by third-party developers. In addition, tools to facilitate integrating the flexible and easy-to-use Orbiter Federation services into a variety of applications, such as workflow managers, will allow Orbiter capabilities to be accessible to a wide range of users. This collection of sophisticated Federation services and Commander modules will facilitate an ever-growing set of capabilities that will make the Orbiter infrastructure highly attractive to a broad range of potential users.

Orbiter Commander, and its flexible architecture that enables the Orbiter Collective, is discussed in the following section.

IV. Commander Framework

Commander is a thick-client Java [19] desktop application supported by the Orbiter Federation SOA and its underlying Data Tier. It is built on the Eclipse Rich Client Platform (RCP), an architecture that allows a robust and powerful Graphical User Interface (GUI) to be developed much more rapidly than through traditional Swing [20] libraries. Eclipse RCP is built on the Eclipse Equinox runtime, an implementation of the OSGi specification [21] for modular Java applications. Eclipse RCP's embedded product branding, user help system, and Standard Widget Toolkit (SWT) [22] provide for a sophisticated interface and a rich overall user experience, and the underlying OSGi standard allows for the fruits of development initiatives to be interoperable and easily integrated into other applications.

Commander's capabilities are implemented as discreet functional components called modules. A module is an OSGi-compliant bundle in the form of an Eclipse plug-in that inherits functionality from the Commander application. Modules encapsulate the implementation of a specific functional component and are completely independent from other Commander modules. Related capabilities are grouped together in suites that fulfill the needs of a particular project or initiative.

The Commander framework, on the other hand, takes responsibility for only the most basic capabilities of the application, such as providing the main application window and launching and managing individual modules at runtime, while making no assumptions about the behavior of a particular module. Commander defines a set of plug-in extension points for defining Commander-compliant suites and modules. Superclasses that perform much of the basic module functionality enhance these extension points, and a growing set of utilities and visual widgets provide a solid base for rapidly developing sophisticated Commander modules. An overview of this framework is included in Figure 4.

The flexibility introduced by this design allows Commander to adopt a plug-and-play paradigm that yields a highly customizable product. That is, modules can be dynamically introduced into the Commander application without requiring the core application to be adjusted, or even rebuilt, to handle them. Their functional independence allows users to choose which capabilities they will integrate to enhance their application, yielding a product that can be easily customized to suit individual requirements. The Commander interface and a few of its modules are shown in Figures 5 and 6.

Commander's implementation as a desktop application allows it achieve the best of both worlds in terms of online and offline computing. The Orbiter Federation SOA on its Logic Tier ensures secure web service interaction and allows users

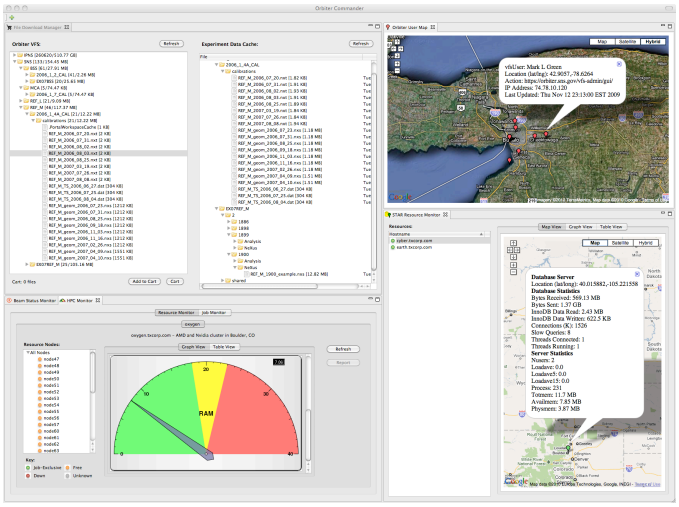


Fig. 5. Commander's Rich Interface via Eclipse RCP

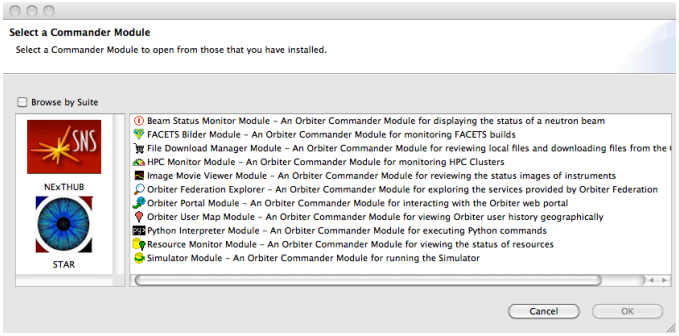


Fig. 6. Commander Modules and Suites

to fully leverage the resources available via the Orbiter infrastructure. However, because Commander is accessed locally, offline capabilities can be provided when networks are poor or unavailable, such as while users are traveling or working remotely on slow connections. That is, service calls can be cached for when the network becomes available again, and Commander may also poll commonly used services to built a usable system state that can be referred to in the event that connectivity is lost. In addition, computationally- or data-intensive capabilities can be provided locally such that users can continue their work offline. These are areas of future work.

In addition to these benefits, as a desktop application Commander is also able to more easily interact with other applications and functionality in the user's local environment. For example, future Commander capabilities could allow users to access their e-mail, local files, or local applications and integrate those capabilities with other Commander modules.

Whereas web-based gateways are lightweight and instantly deliverable, a core challenge of providing a robust desktop application lies in distributing the application on multiple platforms and keeping it updated as new revisions are produced. The Eclipse framework, however, provides tools that address these issues and allow Commander to be easily updated and managed by end-users. By leveraging the Eclipse Equinox p2

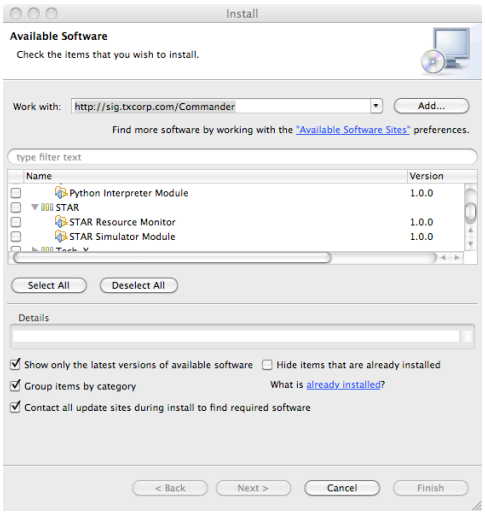


Fig. 7. Commander Modules and the Equinox p2 Provisioning System

provisioning system, Commander modules can be posted to an external site where updated or newly developed capabilities can be seamlessly pushed out to end-users. From this interface, featured in Figure 7, users can find new versions of their installed capabilities and also discover new modules that would assist with their projects. This feature, combined with Eclipse's cross-platform compatibility support through its Delta Pack[3], leads to a significant reduction in the cost and effort associated with delivering Commander's capabilities to end-users.

The generalized design of the core Commander framework and seamless cross-platform distribution also supports the rapid development of new capabilities. Commander-compatible suites and modules can be generated simply by extending the core Commander plug-in at its pre-defined extension points, which the Eclipse IDE uses to automatically generate classes that can be loaded by the Commander framework. Module stubs for completely new functionality can be created in minutes, and finished modules need only be added to the established update site in order to be instantly distributable to all Commander users. This decoupled and distributed provisioning model enables the Orbiter Collective, an ever-growing set of Commander capabilities developed both in-house and by third-party providers. Developers will be able leverage Commander's extension points and toolkit to develop and distribute new Commander modules to end-users. In many of the Commander capabilities presented in this in this paper it was observed that new modules could be conceptualized, designed, developed, and deployed in a matter of days, demonstrating the flexibility and power of this model.

Enabling the rapid development and dissemination of new capabilities ensures that Commander will remain agile as new requirements are presented. Furthermore, the ability to quickly produce valuable new functionality helps facilitate widespread

[3]Currently Linux, Mac OS X, Windows, Solaris, AIX, and HP-UX are supported.

community adoption. Two such suites of capabilities, NExTHUB and STAR, are discussed in the following sections.

V. The Commander NExTHUB Suite

The Spallation Neutron Source (SNS) [23] located at Oak Ridge National Lab (ORNL) is an accelerator-based neutron source that supports several scientific domains such as material sciences, structural biology, and superconductivity. At this one-of-a-kind $1.4 billion facility a world-class suite of instruments is being developed to facilitate science in this wide range of disciplines.

Conceptually similar to but preceding the Orbiter Pilot web interface, SNS had developed a Neutron Sciences web based portal to support the data and computing needs of its users. This portal leverages a job and data management infrastructure custom produced to utilize resources available to the facility including both intra- and extra-ORNL computing resources. Though still in production use, this portal has also enabled solicitation of requirements for a next generation research support cyber-environment. In an evolutionary process, the concept of a more robust environment conceived to better support a broader base for collaboration coupled with better facilitation to computing, data, and software application resources, the concept of the Neutron Experiment and Theory Hub (NExTHUB) has emerged from the SNS scientists. The goal of NExTHUB is to facilitate higher impact science while shortening the publication lifecycle via cyber-enabled collaborative science and publication tools and environments. In principle, NExTHUB would be readily extensible to support other data producing user facilities particularly other neutron scattering and light source user facilities.

Though still in the conception phase, NExTHUB itself would be a multi-tiered, multi-faceted SOA system within ORNL comprised of supporting infrastructure and resources orchestrated to provide services and access to resources. Utilizing this SOA approach enables an open arena to broadly facilitate communities of developers to produce service consuming applications and tools suited to their interests and needs while at the same time complying with ORNL cybersecurity policies incorporating per-user, per-service granularity for user authentication, authorization, and access to resources. To this end, Commander is developing the NExTHUB (Neutron Experiment and Theory Hub) Suite, a set of modules designed to integrate with the work which has been done and is still underway at SNS. This suite, currently consisting of a File Download Manager, Beam Status Monitor, and several other modules, brings Orbiter functionality directly to the desktops of end-users.

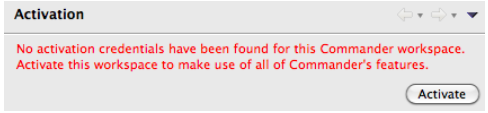


Fig. 8. Commander's Activation Page

NExTHUB users begin by activating Commander. The activation process enables secure Orbiter Federation RESTful service communications by binding the Commander instance to their Orbiter access key identifier and private key information. Until users activate Commander by using their ORNL XCAMS/UCAMS [24] login information to securely download their Orbiter credentials, Commander will restrict the use of its modules. A screenshot of this state is featured in Figure 8.

The File Download Manager is a module that facilitates real-time remote file browsing as well as rapid multi-threaded downloads. These files are served through the Orbiter Virtual File System (VFS), which houses scientific data files across instruments at several neutron sciences facilities including SNS, HFIR [25], LENS [26], Lujan [27], and IPNS. At the time of this writing, the Orbiter VFS supports 956 user accounts with over 808,525 directories and 2,695,387 repository files totaling over 22.33 TB.

The File Download Manager uses RESTful calls to Orbiter Federation services to build a file directory structure for the Commander user. These directories are organized in a hierarchical tree similar to what would be expected in a typical operating system visual file browser. Directories show their total sizes and numbers of files, reporting this live from the Orbiter VFS.

Users can download their remote files directly to their local systems by using the file cart provided by the module. Users point and click to add files and directories to their carts, and can review the contents of their carts to ensure that they are getting only the files they want for their systems. Users may download up to 20 GB of VFS files at a time, where this limit is server-side configurable on a per-user basis.

Users can initiate their downloads through their carts once they have selected all of their files. Users may choose to receive their files as a zipped archive, or alternatively have the module automatically decompress them and write them

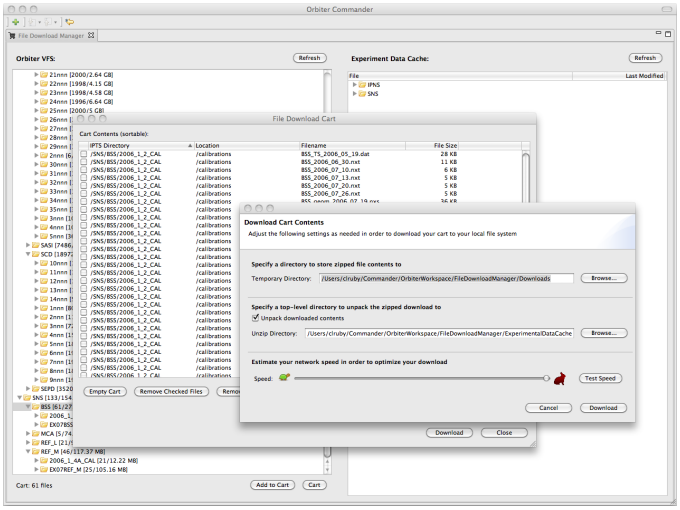


Fig. 9. Commander's File Browser and Download Cart

to the local system in the same hierarchical data structure used on the local repository. They may also designate the top-level directory where files should be written. A screenshot of the File Download Manager and the Download Cart dialog is shown in Figure 9.

The download interface of the File Download Manager also allows the user to specify the network speed for the transmission. This speed should correlate to the relative performance of the user's current network, and dictates the performance of the download service; i.e., the number of download threads to initiate. Future work will also use this metric to optimize file compression for the transfer. That is, the Orbiter Federation service will compress a larger percentage of VFS files for slower networks, while saving time at the service level by streaming uncompressed files to users on faster connections.

The network speed can either be specified directly by the user, or it can be calculated automatically. The File Download Manager can estimate the current network speed by timing the act of calling an Orbiter service that returns a requested number of bytes and comparing it to the maximum bandwidth observed from the Orbiter VFS. Allowing the network speed to vary for each individual download request using this heuristic method permits the module's performance to be tailored to a user's environment at any given time, whether on a professional-quality network, at home, or while traveling when the network may be slow.

The multi-threaded download capability of this module allows large amounts of data to be retrieved quite rapidly from the remote VFS; this has been observed to be an order of magnitude faster than traditional single-threaded downloads. Figure 10 features a multi-threaded download in progress where the download threads are run as separate background processes, allowing the user to continue using Commander without waiting for the download to complete. When the download threads are finished, the user can browse local file directories directly through Commander, reviewing the downloaded files and opening them on his or her local system.

The Beam Status Monitor is a Commander module that tracks the current status of the neutron beam at SNS. This module receives live data via calls to the Orbiter Federation SOA and retrieves not only the most recent beam status but

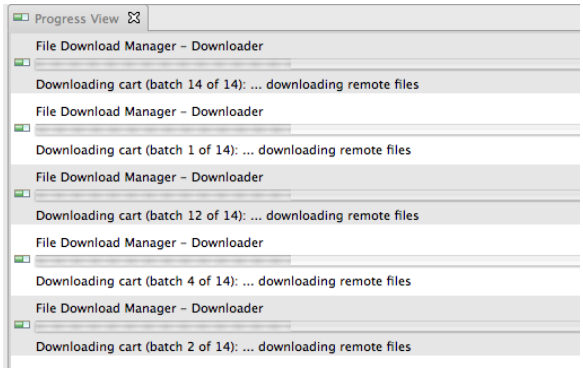


Fig. 10. Commander's Multi-Threaded File Download

Fig. 11. Accessing Live Beam Status and Historical Data

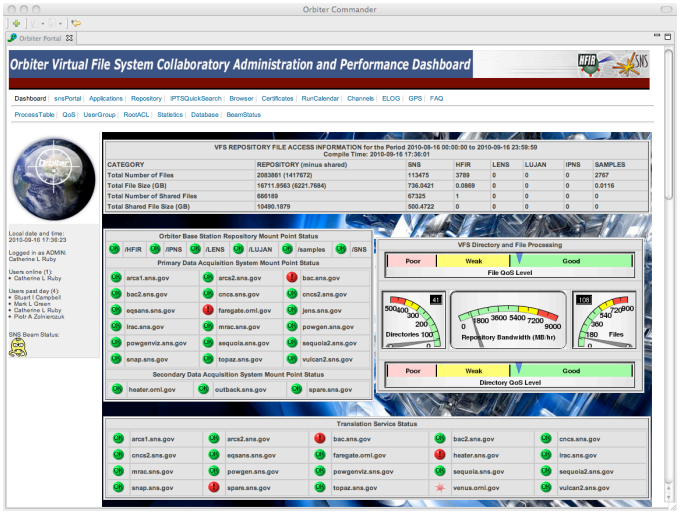


Fig. 12. Accessing Pilot Functionality

also historical charts illustrating the beam status at any given time. This module automatically handles making periodic updates to this view to deliver the most up-to-date information to the user in Figure 11.

Commander provides other capabilities that facilitate work done at SNS as well. Commander's Portal module brings users to Orbiter Pilot, allowing them to access its functionality from within the application. Because Pilot is loaded directly into Commander itself, future work can add additional functionality that integrates Pilot's capabilities with other Commander modules. This module is shown in Figure 12. Commander also provides a module that shows the historical use of the Orbiter VFS geographically on an embedded interactive Google map [28], shown in Figure 13.

This suite of capabilities provides considerable value, where users can browse their remote repositories, download files to their local systems, review live and historical beam sta-

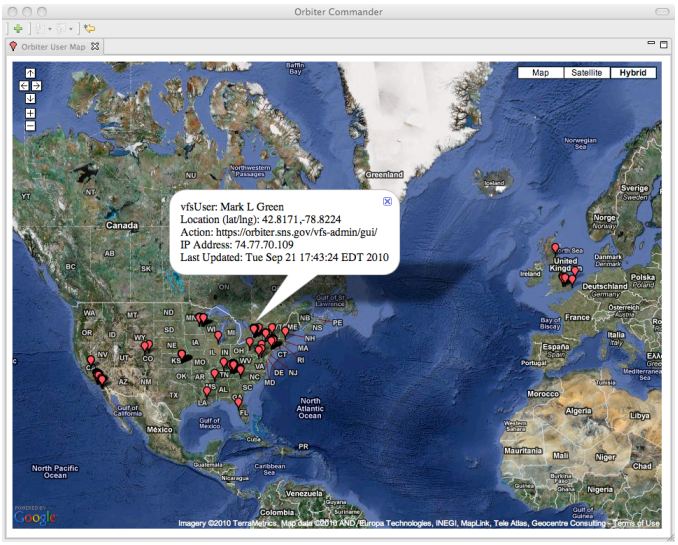


Fig. 13. Orbiter VFS Historical Information

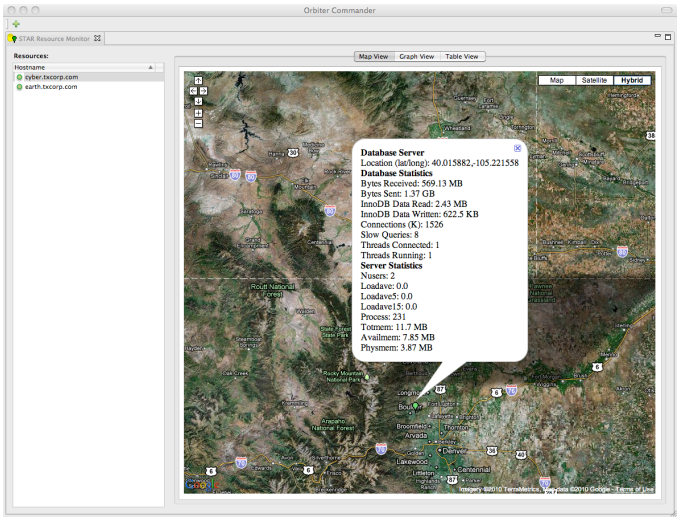


Fig. 14. Capturing Live STAR Resource Statistics

tus information, and access Orbiter Pilot and interactive geographically-arranged historical usage information. These capabilities, in addition to future work planned for this suite, bring Orbiter VFS capabilities seamlessly to end-users.

VI. The Commander STAR Suite

The Solenoidal Tracker at the RHIC (STAR) [29] experiment studies quark-gluon plasma (QGP), a state of matter believed to exist at sufficiently high energy densities [30]. This experiment, at the Relativistic Heavy Ion Collider at Brookhaven National Laboratory, conducts simultaneous studies using several types of specialized detectors. The experiment is composed of 52 institutions from 12 countries, with a total of 529 collaborators. To facilitate this experiment, Commander is developing the STAR Suite, a set of modules to support the development of a customizable set of Orbiter Federation web services for efficient access to arbitrary and distributed relational Nuclear Physics (NP) databases.

The STAR Resource Monitor module helps users track the live status of STAR databases. Using Orbiter Federation RESTful web services, this module lists available resources and displays an interactive Google map displaying detailed resource information, such as bytes sent and received, available memory, numbers of users and processes, and other valuable information. This module also provides real-time charts and tables presenting this information, allowing users to choose the best presentation in order to understand the states of available STAR NP database resources. A screenshot of this capability is featured in Figure 14.

Commander also provides a STAR Simulator module for running queries against available NP database resources. Users direct the module to a local STAR query file and specify the resource to run the queries against. The queries, their results from the query database, and their total times are then displayed. Figure 15 displays the STAR Simulator and the results of several queries.

Together these modules add value to the STAR web service development initiative. Using STAR Resource Monitor, resources can be chosen from the monitor and used in the Simulator module, allowing users to systematically test the performance of the available STAR NP databases. These capabilities facilitate the development of these services, which in turn will ultimately benefit collaborators at the STAR experiment.

VII. Related Work

The Orbiter Federation web services are related to several industry initiatives involved in developing SOA solutions and portal technologies. WSO2, for example, produces enterprise middleware as componentized products, allowing their services to be customized for the architectures it is adopted by [31]. IBM's WebSphere encompasses application infrastructures and portal solutions for SOA environments [32],

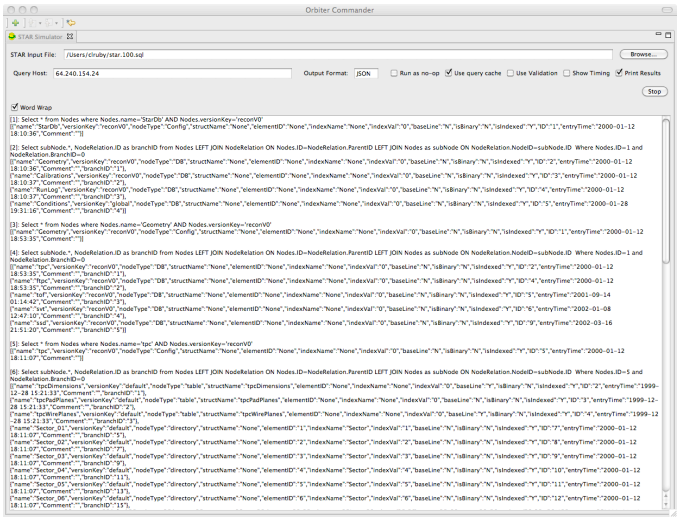


Fig. 15. Commander's STAR Simulator Module

providing a similar foundation to what supports the Orbiter Pilot. Orbiter Federation's method for providing signed and secure web service transactions can be compared to the model used by Amazon Web Services (AWS) [33]. This robust security method allows Orbiter Federation to provide a scalable service-based platform, comparable to the enterprise efforts of Oracle [34] or Microsoft's Azure [35].

Eclipse RCP applications are being continuously developed to solve a wide variety of problems. Eclipse RCP Eclipse Integrated Development Environment (IDE) itself, and has been adopted by many open source and commercial applications across several disciplines, including endeavors by major organizations such as Apache and NASA. Used to manage Dutch railway schedules, perform DNA sequence scanning, and analyze web performance, to name a few examples, these use-cases demonstrate that Eclipse RCP is a robust and enterprise-quality platform that is suitable for a wide range of applications. The flexible OSGi specification, upon which the Eclipse Equinox runtime is built, has been adopted by several organizations including Apache and Concierge, which produces an OSGi-compliant framework implementation for mobile and embedded devices [36], [37].

The Commander framework as a whole unifies the power of the Orbiter Federation service-based platform with the rapid application development hooks provided by Eclipse RCP. By leveraging the secure Orbiter Federation services the Commander application is able to streamline the process of connecting users to the wide variety of Orbiter services and capabilities. From managing user credentials to automatically signing and handling Orbiter Federation service requests and responses, users are provided with a seamless application that bridges the gap between web-based capabilities and local system resources. That is, Commander serves as a live thick-client gateway to Orbiter infrastructure, providing access to the remote resources and services that Orbiter offers while allowing users to leverage their local resources as well, such as local GPUs, editors, and viewers. Future work on Commander will provide additional fault tolerance for spotty network accessibility or periods of offline work, by providing the ability to queue service requests for the next time the network becomes available. This will offer a significant advantage over standard browser-based thin-client gateways.

The NExTHUB and STAR Commander suites presented in this paper are proofs-of-concept for the Orbiter Collective layer, which strives to provide a rapid development environment for new Commander capabilities. The core Commander framework provides simple hooks for authenticating to and using Orbiter Federation services, and basic functionality can be rapidly extended to produce highly customized suites of tools for a wide variety of uses. Whether developed in-house or eventually by third-party users, this model for organic growth requires a stable and flexible core framework which Commander provides by leveraging Orbiter Federation services and Eclipse RCP. Like the Apple and Android development models mentioned in the introduction of this paper, this Collective approach allows Commander's capabilities to be driven by popular demand and facilitates the widespread community adoption of the core Orbiter gateway infrastructure by ensuring that new capabilities can be rapidly tailored for a wide range of potential uses.

VIII. Conclusion

The NExTHUB and STAR Suites both add value to their supporting initiatives, where Commander is capable of facilitating the work done at SNS as well as with the STAR experiment. Using the robust and secure web services offered by the Orbiter Federation SOA, and by implementing tiers III and IV of the Multitier Portal Architecture, Commander is able to provide flexible and sophisticated functionality to its users. This includes facilitating the rapid download of scientific data files, live delivery of important data and statistics, and simulation capabilities that will ultimately benefit these programs.

Future work on Commander will incorporate live chat capabilities, integration with workflow management tools, and several tailored capabilities for NExTHUB and STAR. Additional suites such as a set of tomography modules for Rapid Image Processing (RIP) are planned as well. Commander's well-defined set of extension points and expanding library of utilities and widgets will facilitate the Orbiter Collective's rapid development of these new capabilities, and the Equinox p2 provisioning system provided by the Eclipse framework will allow for the seamless delivery of new modules both by in-house and by third party developers in the future.

Commander's structured framework mitigates the common issues associated with developing and sustaining complex scientific environment capabilities. Commander leverages the fast and secure Orbiter Federation RESTful web services to provide a sophisticated desktop application while enabling offline computing capabilities and integration with other local applications. Commander's ability to grow organically with new suites and modules will facilitate community adoption and the continuing relevance of its capabilities. Furthermore, its customizable interface and plug-and-play delivery model makes the Commander framework and its modules attractive to the widest range of potential users. These capabilities allow Orbiter Commander to present a robust, scalable, and sustainable scientific gateway solution.

Acknowledgment

We would particularly like to acknowledge Dr. Mark E. Hagen at the Spallation Neutron Source for his input and guidance on the Neutron Experiment and Theory Hub.

This work is supported by the Neutron Scattering Science Division Analysis Software Development (NSSD-ASD), Spallation Neutron Source Contract #: 4000057260, Virtual Instrumentation Experiment Optimization for High-Throughout Scientific Analysis (Orbiter) Phase I and II DOE BES SBIR Grant #: DE-FG02-08ER85000, and CWS4DB: A Customizable Web Service for Efficient Access to Distributed Nuclear Physics Relational Databases Phase I and II DOE BES SBIR Grant #: DE-FG02-07ER84757.

REFERENCES

[1] Apple iphone developer program. [Online]. Available: http://developer.apple.com/programs/iphone/

[2] Android developers. [Online]. Available: http://developer.android.com/index.html

[3] Open grid computing environment (OGCE). [Online]. Available: http://www.collab-ogce.org/ogce/

[4] Gridsphere portal framework. [Online]. Available: http://www.gridsphere.org/gridsphere/gridsphere

[5] J. Alameda, M. Christie, G. Fox, J. Futrelle, D. Gannon, M. Hategan, G. Kandaswamy, G. von Laszewski, M. A. Nacar, M. Pierce, E. Roberts, C. Severance, and M. Thomas, "The open grid computing environments collaboration: Portlets and services for science gateways." *Concurrency and Computation*, vol. 19, no. 6, pp. 921–942, April 2007.

[6] Eclipse rich client platform. [Online]. Available: http://www.eclipse.org/rcp/

[7] S. Jones and M. Morris, "A methodology for service architectures," OASIS Draft, October 2005. [Online]. Available: http://www.oasis-open.org/committees/download.php/15071/

[8] Orbiter virtual file system service oriented architecture interfaces. [Online]. Available: https://orbiter.sns.gov

[9] M. L. Green and S. D. Miller, "Multitier portal architecture for thin- and thick-client neutron scattering experiment support," M. Pierce, Ed. International Workshop on Grid Computing Environments (GCE), Nov. 11-12 2007.

[10] W. W. Eckerson, "Three tier client/server architecture: Achieving scalability, performance, and efficiency in client server applications," *Open Information Systems*, vol. 10, 1995.

[11] MySQL: The world's most popular open source database. [Online]. Available: http://www.mysql.com/

[12] R. T. Fielding, "Architectural styles and design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000. [Online]. Available: http://www.ics.uci.edu/ fielding/pubs/dissertation/top.htm

[13] P. C. K. Alan O. Freier, Philip Karlton, "The SSL protocol - version 3.0," Internet Draft, Transport Layer Security Working Group, November 1996.

[14] Php: Hypertext preprocessor. [Online]. Available: http://www.php.net/

[15] B. Kaliski, "PKCS #1: RSA encryption version 1.5," March 1998, network Working Group, RSA Laboratories East. [Online]. Available: http://tools.ietf.org/html/rfc2313

[16] *Keying Hash Functions for Message Authentication*, vol. 1109/1996. Advances in Cryptology - Crypto '96, 1996.

[17] A. Singhal, T. Winograd, and K. Scarfone, "Guide to secure web services: Recommendations of the national institute of standards and technology," NIST, U.S. Department of Commerce Special Publication 800-95, August 2007. [Online]. Available: http://csrc.nist.gov/publications/nistpubs/800-95/SP800-95.pdf

[18] Web services interoperability organization. [Online]. Available: http://www.ws-i.org/

[19] Java. [Online]. Available: http://www.java.com/en/

[20] A swing architecture overview. [Online]. Available: http://java.sun.com/products/jfc/tsc/articles/architecture/

[21] OSGi alliance. [Online]. Available: http://www.osgi.org/Main/HomePage

[22] SWT: The standard widget toolkit. [Online]. Available: http://www.eclipse.org/swt/

[23] Spallation neutron source. [Online]. Available: http://www.sns.gov

[24] Open resource collaboration. [Online]. Available: http://www.ornl.gov/xcams/xcamsfaq.htm

[25] High flux isotope reactor: Overview. [Online]. Available: http://neutrons.ornl.gov/facilities/HFIR/

[26] LENS: The low energy neutron source. [Online]. Available: http://www.indiana.edu/ lens/index.html

[27] Los Alamos National Laboratory - materials science at LANSCE. [Online]. Available: http://lansce.lanl.gov/lujan/index.shtml

[28] Google maps API family. [Online]. Available: http://code.google.com/apis/maps/index.html

[29] The STAR experiment at the relativistic heavy ion collider, Brookhaven National Lab. [Online]. Available: http://www.star.bnl.gov/

[30] The STAR experiment at the relativistic heavy ion collider, Brookhaven National Lab: The physics of star. [Online]. Available: http://www.star.bnl.gov/central/physics/

[31] WSO2. [Online]. Available: http://wso2.com/

[32] IBM websphere software. [Online]. Available: http://www-01.ibm.com/software/websphere/

[33] Amazon web services. [Online]. Available: http://aws.amazon.com/

[34] Oracle cloud computing. [Online]. Available: http://www.oracle.com/us/technologies/cloud/index.htm

[35] Windows azure. [Online]. Available: http://www.microsoft.com/windowsazure/

[36] Eclipse rich client platform (RCP) applications. [Online]. Available: http://www.eclipse.org/community/rcp.php

[37] J. McAffer, P. Vanderlei, and S. Archer, *OSGi and Equinox: Creating Highly Modular Java Systems*. Addison-Wesley Publishing Company, 2010.

Orbiter Commander

A Flexible Application Framework for Service-Based Scientific Computing Environments

Catherine L. Ruby, Mark L. Green
Systems Integration Group, Tech-X Corporation
clruby@txcorp.com, mlgreen@txcorp.com

Stephen D. Miller
Spallation Neutron Source, Oak Ridge National Lab
millersd@ornl.gov

GCE10 – November 14, 2010

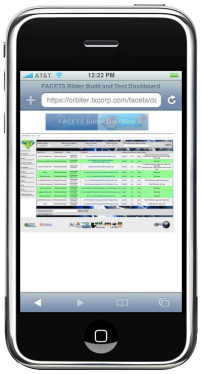
Application Frameworks

- Application Frameworks define ***how*** to solve common problems, not solutions themselves
  - Using well placed abstraction layers
  - Defining points for extending functionality
- Notable Gateway Computing Environments using this methodology:
  - Open Grid Computing Environments (OGCE)
  - GridSphere
  - WSO2

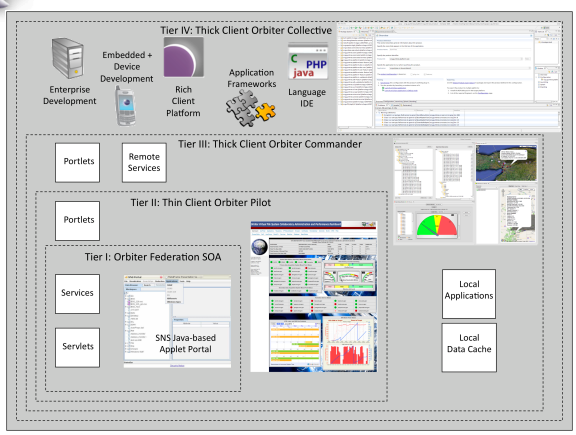
Other Framework Examples

- Apple's iPhone
- Google's Android

Stable and extensible frameworks permit **third-party development** of new capabilities, assuring relevance and **widespread community adoption**



Multitier Portal Architecture



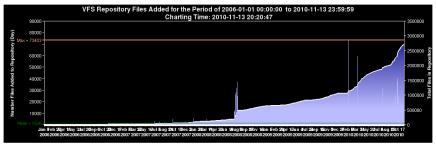
Orbiter Federation Services

- RESTful web services implemented in PHP offer extremely rapid turnaround
- Services validate user identities as well as request integrity, securing the underlying data and capabilities
- Underlying MySQL database uses master/slave database replication to ensure fault tolerance and reliability

Since February 2010…
- **31.8M** service requests were logged by Orbiter Virtual File System
- **9.2M** cataloging/monitoring/QoS tasks were performed

Orbiter Virtual File System

- Catalogs and provides access to scientific files from SNS, HFIR, LENS, LUJAN, and IPNS
- Orbiter Federation web services facilitate querying for directories, files, and pulling these resources down from their remote storage areas
- User access is controlled using the authentication and validation mechanisms in the secure Orbiter services



***956** user accounts,*
***2.7M** files,*
***22.33TB** of data*
***31.8M** service requests since 02-2010*

Secure Authenticated Services

https://{ServiceProvider}/{ResourceAddress}/{Attributes}/{ID}/{ExpirationTime}/{Signature}

- Orbiter Access Key ID declares user identity
- Expiration Time ensures request lifetime/validity
- RSA Private Key Signature ensures data integrity

Similar to the Amazon AWS Security Model

Secure Authenticated Services (cont.)

- Protects against several well-known attacks
  - Loss of Confidentiality (SSL)
  - Message alteration (signature)
  - Falsified messages (access key + signature)
  - Replay of Message (timestamp)

*Service turnaround for a no-op with full validation and authorization – **$1x10^{-6}$ seconds***

*Generally observed full service responses – **$\sim 2x10^{-2}$ seconds***

Data, Logic, and Presentation

Orbiter Federation Platform

Data Tier — MySQL Database, Master/Slave Replication

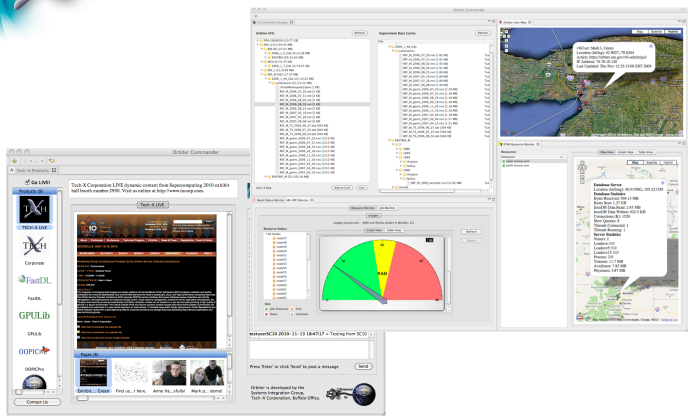
Logic Tier — REST, SSL, Authenticated services
Monitoring/QoS

Presentation Tier – *Pilot, Commander, Collective*

Orbiter Pilot

- Multitier Portal Architecture Tier II
- Thin-Client browser-based application
- Access to Orbiter VFS file repositories
- Dashboards, QoS metrics, and charting tools for data collected by the Orbiter infrastructure
- Driven by Orbiter Federation services

Orbiter Commander



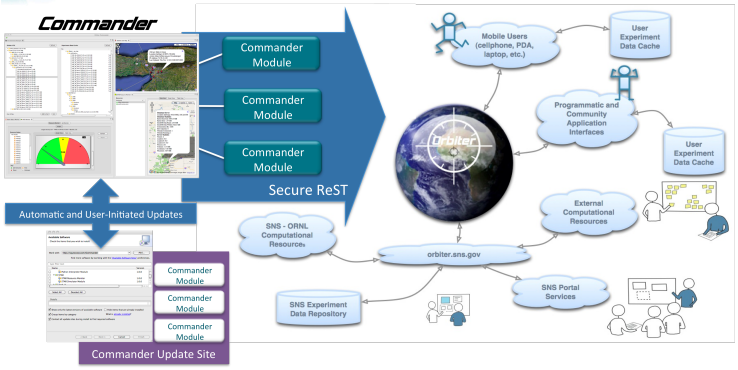
Orbiter Commander (cont.)

- Multitier Portal Architecture Tier III
- Thick-Client RCP Application
- Built on Orbiter Federation Services, providing live gateway capabilities
- Modular design for plug-and-play capabilities
- Well-defined extension points for code-reuse and future development
- Highly customizable interface

Orbiter Commander (cont.)

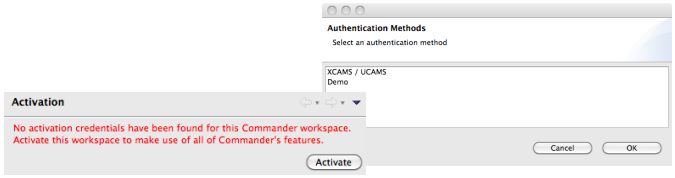
- Eclipse RCP
  - Mature Integrated Development Environment
  - Simplified Standard Widget Toolkit
  - Automatic handling of core GUI implementation
- OSGi specification
  - Modular application “bundles”
  - Decouples application components into individually loaded packages

Commander Architecture



Commander User Identity Management

- Commander requires users to *activate*, prompting them to input their information
- An Orbiter Federation service accepts their credentials and generates any necessary account information
- Orbiter access key and RSA private key information is downloaded securely for the Commander user
- Commander uses these Orbiter parameters to securely perform all subsequent Federation service transactions

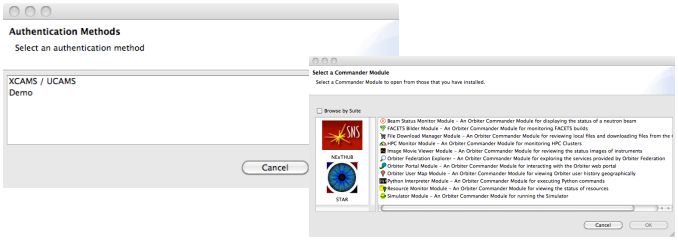


Why a Thick-Client?

- Integration with local resources/ applications
- Possibility for:
  - Fault-tolerance for spotty network connections
  - Offline mode for planned network outages (travel, etc)
  - Queued/background processes for maintaining up-to-date views – enhanced multitasking

Suites of Modules

- Capabilities, or **modules**, are organized in branded **suites**, each addressing the requirements of a particular problem or domain

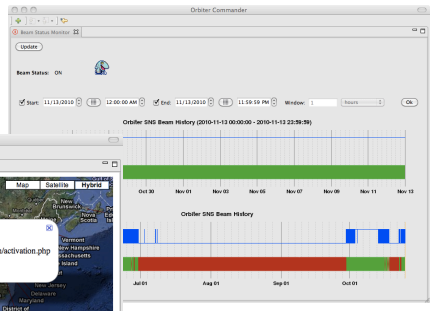


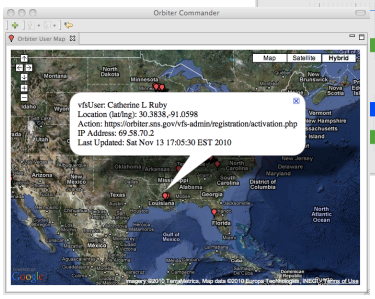
NExTHUB Suite

- Work supporting our efforts with the Neutron Experiment Theory Hub, at the Spallation Neutron Source, Oak Ridge National Laboratory
- Diverse capabilities from remote file retrieval to live status monitoring
- Rapidly developed using the foundation of the Commander framework and the Orbiter Federation services

NExTHUB Suite (cont.)

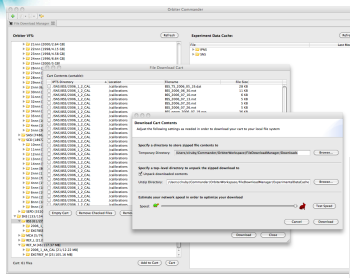
- Live minute-by-minute updates of the current status of the beam as well as historical data



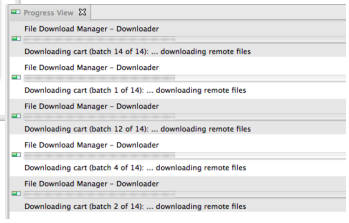


- Usage statistics served by the Orbiter Federation services

NExTHUB Suite (cont.)

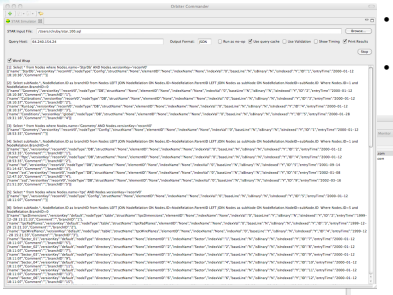


- On-demand seamless access to remote Orbiter VFS files
- File permissions based on Federated Orbiter identities
- 22.33 TB of data, 2.7M repository files

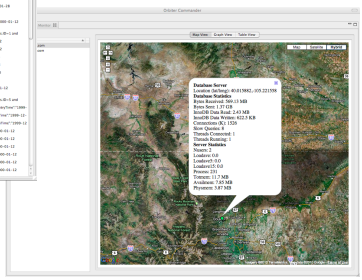


- Cart-based file download paradigm
- Dynamically set per-user limit provided from Orbiter Federation Services (normally 20GB per cart)
- Multi-threaded download as a background process

STAR Suite



- Modules supporting our work with members of the STAR experiment
- Simulate STAR database queries against enabled remote resources



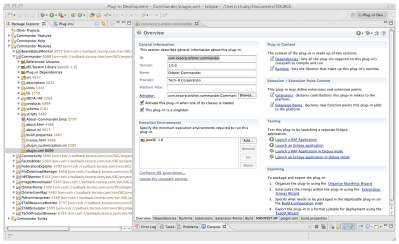
- Review load and other Monitoring/ QoS metrics on STAR database query endpoints

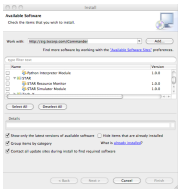
Orbiter Collective

- Multitier Portal Architecture Tier IV
- Orbiter Commander provides Eclipse extension points for:
  - New suites
  - New modules
  - Orbiter Federation authenticated clients
  - User activation methods
- Developers can quickly leverage these to create Commander-compliant modules

Orbiter Collective (cont.)

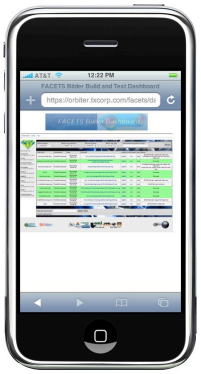
- Eclipse IDE provides the development environment for extensions to the Commander framework
- Eclipse p2 provisioning system allows third-party content to be directly consumed by Commander users – *no reinstall required*





Orbiter Collective (cont.)

- Organic growth of new functionality
- Agile development based on evolving requirements – focus on ***what to build*** not ***how***
- Ensures continuing use and community adoption, extending the lifetime of the effort



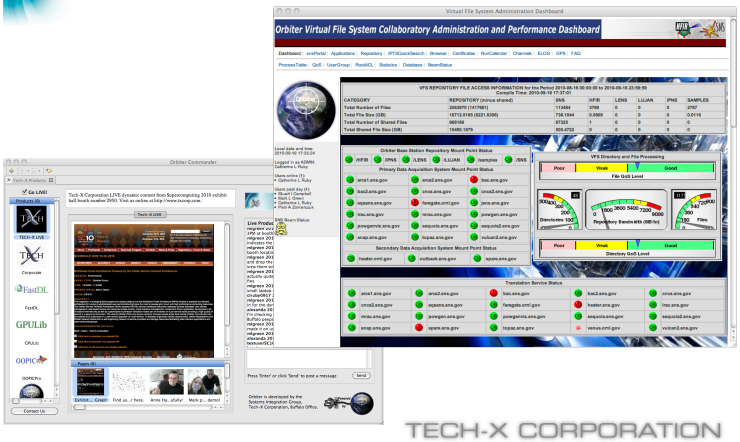
Conclusions

- **Multitier Portal Architecture** creates a layered approach for building increasingly complex user interfaces
- **Orbiter Federation** services provide fast and secure access to scientific data and resources
- **Orbiter Pilot** uses these service to deliver a thin client portal – a browser-based gateway
- **Orbiter Commander** delivers these services as end-user capabilities in a rich and customizable set of interfaces – a thick-client gateway

Conclusions (cont.)

- Commander's modular design and well-defined extension points enable the **Orbiter Collective**, a method for third-party development of Commander capabilities
- Orbiter Collective allows the Orbiter infrastructure be **easily customized and adapted** to the needs of a large and diverse scientific user community

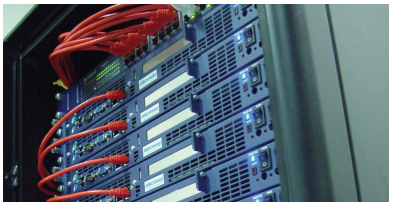
Thank you for your attention!



For More Information

Mark L. Green
Systems Integration Group
Tech-X Corporation, Buffalo Office
Telephone: 716-204-8686

Email: orbiter@txcorp.com
URL: https://orbiter.txcorp.com

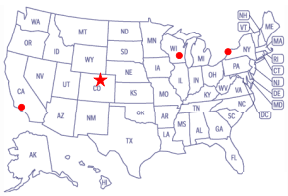


CWS4DB: A Customizable Web Service for Efficient Access to Distributed Nuclear Physics Relational Databases

FY 2008 SBIR Phase II Proposal Award Number: DE-FG02-07ER84757

Dr. Mark L. Green, PI

Tech-X Corporation, Buffalo Office
Systems Integration Group



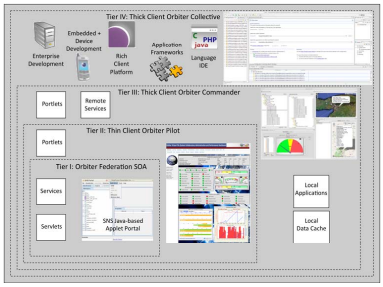
Tech-X Orbiter Project

- **Orbiter is an end-to-end framework** delivering fast and secure solutions through both thin-client web access and thick-client desktop application suites and modules. These applications leverage the information-sharing capabilities of Orbiter in providing powerful and personalized web-accessible components.
- **Service Oriented Architectures (SOAs)** have been proven to be a popular design for building reliable and scalable large-scale software systems, borrowing from earlier Object Oriented Programming (OOP) techniques of encapsulation, cohesion, and the use of abstraction layers behind well-defined public APIs. Orbiter Federation services, built upon industry standards, offer fast and secure access to a wide range of capabilities.

https://orbiter.txcorp.com

Orbiter Multitier Portal Architecture (MPA)

- Through the Multitier Portal Architecture (MPA) Orbiter Federation services are delivered directly to end-users via a variety of rich interactive interfaces. The MPA allows increasingly sophisticated capabilities to be rapidly developed to suit a wide range of user requirements, and the foundation provided by Orbiter Federation enables these capabilities to be delivered swiftly and securely to end-users.
- Framework for delivering capabilities to thin- and thick-clients using the Orbiter RESTful SOA
- Flexible and re-usable architecture for developing capabilities for thin web clients and thick local clients
- Comprised of four tiers:
  - Tier I: Orbiter Federation SOA
    - Low-level RESTful services
  - Tier II: Thin-Client Orbiter Pilot
    - Light weight client access
  - Tier III: Thick-Client Orbiter Commander
    - Fully capable installed application
  - Tier IV: Thick-Client Orbiter Collective
    - IDE for Orbiter development



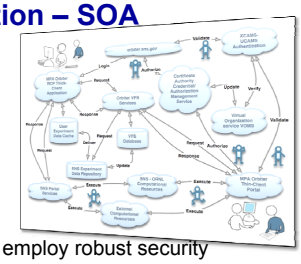
https://orbiter.txcorp.com

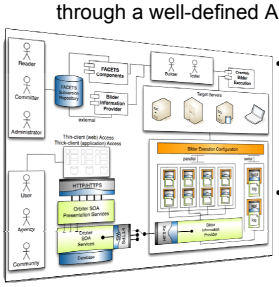
Tech-X Orbiter Project

- **Federation** provides a Service Oriented Architecture (SOA) of web services, delivering powerful, lightweight, secure, and scalable capabilities.
- **Pilot** delivers Federation web services through web-accessible thin-clients. These gateway and portal clients deliver Orbiter capabilities through easy-to-use web interfaces.
- **Commander** is a rich cross-platform desktop application that provides access to Federation services while allowing Orbiter systems to interact directly with local compute resources.
- **Collective** opens the door to advanced collaboration across a wide range of associations, facilities, and institutions. Orbiter meets the needs of these organizations through the development of integrated cross-platform applications that enable the full value of third party products and services.

Orbiter Federation – SOA

- Tier I of the Orbiter Multitier Portal Architecture
- Orbiter services are implemented as Representational State Transfer (RESTful) web services that deliver functionality through a well-defined API.





- These services employ robust security standards including SSL and signed requests that ensure client identities, the integrity of their RESTful service calls, and the privacy of their transmissions.
- Orbiter Web Services use SSL encryption, access key identifiers, timestamps, and private key signatures, ensuring the privacy, authorization, and request integrity of all interactions.

Tech-X Orbiter Project

- Orbiter Pilot, Orbiter Commander, and the Orbiter Collective demonstrate how access to Orbiter Federation resources and services can be provided through **reliable, scalable, and interactive scientific gateways**.
- Its **flexible cross-platform desktop solutions** present the user with a rich and customizable interface to data, information, computational resources, and enterprise application bases.
- Orbiter **solutions are inherently scalable**, where Federation, Pilot, Commander, and Collective each build modular capabilities that are focused on particular needs.
- Orbiter solutions have been routinely used for the **management and retrieval of large amounts of data** and information.

CWS4DB Project

A customizable Web Service for Efficient Access to Distributed Nuclear Physics Relational Databases
DOE NP Phase I and II – Manouchehr Farkhondeh

Tech-X: Mark L. Green (PI), Catherine L. Ruby, Sean Burley, Krishna Kantam, Srilakshmi Ramireddy

Need: As the size of NP data grows and the collaborative nature of HENP experiments increases, the ability to access differently organized relational databases remotely, efficiently, and yet in a user-friendly and interoperable manner is becoming very important.

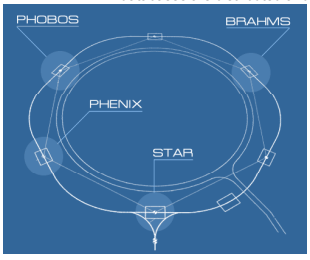
Partners: Jerome Lauret, Dmitry Arkhipkin (STAR project at BNL), Kate Keahey (Nimbus project at ANL), Doug Olson (Open Science Grid), Alexandre Vaniachine (ATLAS project ANL/CERN)

DOE Beneficiaries: Nuclear and high energy physics communities, national laboratories, and collaborative projects

Commercial Beneficiaries: Companies requiring efficient web service access to distributed relational databases with high-level database and user APIs

Problem Identification

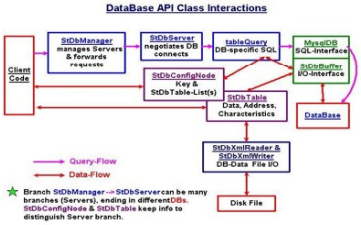
- The importance of this project comes from the fact that a large fraction of the ever-growing data generated by Nuclear Physics (NP) experiments is stored in relational databases. For example:
  - The BNL Relativistic Heavy Ion Collider (RHIC) supports STAR (Solenoidal Tracker at the at the RHIC) which composed of 52 institutions from 12 countries, with a total of 529 collaborators;
  - relational databases (such as Condition databases, Calibration databases, and Geometry databases) are heavily used in the STAR experiment;
  - while accessing data in such databases is convenient and available for local users who are familiar with a particular database, the situation becomes more complicated when the databases are distributed and heterogeneous.



- Tech-X therefore proposed a system to overcome the outlined challenges by bridging relational databases with high-level APIs through Web services.
  - In particular, the distributed and heterogeneous nature of the databases will be addressed by creating Web services in the Orbiter Federation Service Oriented Architecture (SOA), which provides mechanisms coordinating access to diversified data resources through REST (Representational State Transfer) services, caching, authentication, and authorization.

CWS4DB Technical Objectives

- Tech-X proposes to develop a customizable Web service for efficient access to distributed NP databases. The proposed system will consist of:
  - a generic Web service for accessing arbitrary distributed relational databases,
  - a reference client implemented at the Relativistic Heavy Ion Collider (RHIC) at Brookhaven National Laboratory (BNL), for the Solenoidal Tracker at the at the RHIC (STAR) experiment, and
  - a tool for creation of the high-level and domain-specific clients required by particular applications.
- The Phase II objectives include:
  - Take into account what was learned from the research in Phase I and extend the CWS4DB prototype into a production-quality, load-balanced, auto-caching, grid-enabled, fault-tolerant, and on-demand system.



  - Use a flexible work plan involving a separate piece of technical functionality that can be implemented in a way that can be exercised in the STAR computing environment, yet developed in a general way for application's from other NP projects.
  - ***The ultimate goal is to produce a set of software tools and services that can be easily adapted by the NP application developer.***

CWS4DB Tasks

- Task 1: Determine CWS4DB System and Load Balancing Additional Requirements and Properties (Tech-X & BNL)
  - Extend the Phase I developed requirements and properties and continue prototype work with our partners.
- Task 2: Design and Implement Tiered Deployment Capabilities (Tech-X)
  - Develop a tiered deployment based protocol for the CWS4DB system.
- Task 3: Design and Implement Auto-Caching Infrastructure (Tech-X & BNL)
  - Provide a sophisticated auto-caching mechanism in order to increase the effective system performance based on work with our partners.
- Task 4: Enable Multi-Virtual Organization Role-Based Capabilities (Tech-X)
  - Develop the CWS4DB infrastructure required for user-friendly management and caching capabilities.
- Task 5: Develop Dynamic On-Demand Data Resource Access (Tech-X)
  - This on-demand service provides a STAR MySQL database instance using the Amazon EC2 deployments.

CWS4DB Tasks Continued

- Task 6: Develop Fault Resilient Data Resource Pathways (Tech-X)
  - Eliminated a single point of failure for the STAR C++ API bound codes database query requests.
- Task 7: Develop a Prototype On-Demand Data Resource Node (Tech-X & BNL)
  - Prototyped the deployment of a on-demand data resource node to meet the dynamic data demands of the STAR collaboration.
- Task 8: Prototype Pre-Cache Capabilities for Production Job Workflow (Tech-X & BNL)
  - Pathway for an authenticated and authorized user upon configuration of the CWS4DB system to execute the customizable site specific test suite for pre-caching production job queries is complete.
- Task 9: Develop a Customizable Site Specific Test Suite (Tech-X)
  - In order to deliver a high quality of service infrastructure a customizable and site specific test suite is required to validate and verify the performance and data delivery capabilities of the CWS4DB system.

CWS4DB Tasks

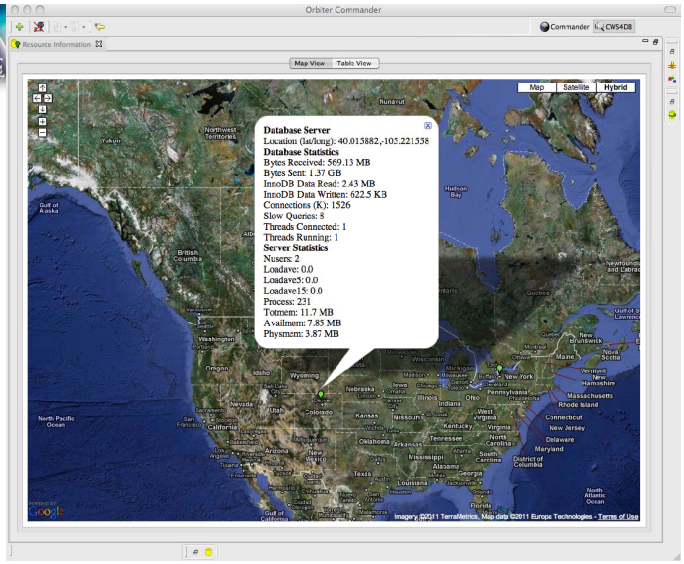
- Task 1: Determine CWS4DB System and Load Balancing Additional Requirements and Properties (Tech-X & BNL)
  - Extend the Phase I developed requirements and properties and continue prototype work with our partners.
- Task 2: Design and Implement Tiered Deployment Capabilities (Tech-X)
  - Develop a tiered deployment based protocol for the CWS4DB system.
- Task 3: Design and Implement Auto-Caching Infrastructure (Tech-X & BNL)
  - Provide a sophisticated auto-caching mechanism in order to increase the effective system performance based on work with our partners.
- Task 4: Enable Multi-Virtual Organization Role-Based Capabilities (Tech-X)
  - Develop the CWS4DB infrastructure required for user-friendly management and caching capabilities.
- Task 5: Develop Dynamic On-Demand Data Resource Access (Tech-X)
  - This on-demand service provides a STAR MySQL database instance using the Amazon EC2 deployments.

Orbiter Federation – SOA

https://{ServiceProvider}/{ResourceAddress}/{Attributes}/{ID}/{ExpirationTime}/{Signature}

- Orbiter Access Key *{ID}* declares user identity
- *{Expiration Time}* ensures request lifetime/validity
- RSA Private Key *{Signature}* ensures data integrity

Similar to the Amazon AWS Security Model



CWS4DB Tasks Continued

- Task 6: Develop Fault Resilient Data Resource Pathways (Tech-X)
  - Eliminated a single point of failure for the STAR C++ API bound codes database query requests.
- Task 7: Develop a Prototype On-Demand Data Resource Node (Tech-X & BNL)
  - Prototyped the deployment of a on-demand data resource node to meet the dynamic data demands of the STAR collaboration.
- Task 8: Prototype Pre-Cache Capabilities for Production Job Workflow (Tech-X & BNL)
  - Pathway for an authenticated and authorized user upon configuration of the CWS4DB system to execute the customizable site specific test suite for pre-caching production job queries is complete.
- Task 9: Develop a Customizable Site Specific Test Suite (Tech-X)
  - In order to deliver a high quality of service infrastructure a customizable and site specific test suite is required to validate and verify the performance and data delivery capabilities of the CWS4DB system.

CWS4DB Summary

File Name : txc02.ccr.buffalo.edu.config.inc.php

```
// ********************************* STAR specific config ***************************************
/**
 * @var STRING ORBITERCACHEFILELOCATION Cache file location.
 */
define('ORBITERCACHEFILELOCATION', '/tmp/cache');
/**
 * @var STRING ORBITERHASHTYPE Orbiter Hash type
 */
define('ORBITERHASHTYPE', 'sha1');
/**
 * @var integer ORBITERQUERYCONNECTIONSTRINGS Number of Orbiter Query Connection strings.
 */
define('ORBITERQUERYCONNECTIONSTRINGS', 2);
/**
 * @var boolean ORBITERUSEQUERYDB Defines whether to use Orbiter Query DB.
 */
define('ORBITERUSEQUERYDB', true);
/**
 * @var integer ORBITERQUERYDBSERVICEADDRESS Orbiter Query DB Load balancer service address.
 */
define('ORBITERQUERYDBSERVICEADDRESS', 'http://txc02.ccr.buffalo.edu/orbiter/'.ORBITERVERSION.'/service/webservice');
/**
 * @var string ORBITERSQLFILELOCATION Orbiter Sql file location to run the pre-cache for new resource.
 */
define('ORBITERSQLFILELOCATION', '/tmp/sqlfiles/auau200_log.txt');
```

CWS4DB Tasks

- Task 3: Design and Implement Auto-Caching Infrastructure (Tech-X & BNL)
  - Provide a sophisticated auto-caching mechanism in order to increase the effective system performance based on work with our partners.
- Task 4: Enable Multi-Virtual Organization Role-Based Capabilities (Tech-X)
  - Develop the CWS4DB infrastructure required for user-friendly management and caching capabilities.
- Task 7: Develop a Prototype On-Demand Data Resource Node (Tech-X & BNL)
  - Prototyped the deployment of a on-demand data resource node to meet the dynamic data demands of the STAR collaboration.
- Task 8: Prototype Pre-Cache Capabilities for Production Job Workflow (Tech-X & BNL)
  - Pathway for an authenticated and authorized user upon configuration of the CWS4DB system to execute the customizable site specific test suite for pre-caching production job queries is complete.

Task: DB Timings (DONE)
Using sequences of SQL operations that are recorded from actual STAR DB usage, we evaluated database performance under load by timing numerous repetitions of these operations against local and remote databases. The sample SQL sequences are:

Name	# Operations
db-perf-test.txt	6,667
offline.auau200.full.sql	8,911
offline.dau200.full.sql	8,784
offline.pp500.full.sql	6,667

CWS4DB Database Query Caching and Optimization

- Network bandwidth is important and depends on the last mile normally
- Database server load is minimal
- Investigate the database service payload size
- Wrote a custom ReSTful PHP database service with a JSON (JavaScript Object Notation) payload to compare with the XML payload

Timing results for *db-perf-test.txt*:

DB Host	# Repetitions	Avg Time (sec)
dbx.star.bnl.gov	10	921.66
orbiter.txcorp.com	10	3.42
cyber.txcorp.com	10	11.63
dbx.star.bnl.gov	20	922.02
orbiter.txcorp.com	20	3.49
cyber.txcorp.com	20	11.7
dbx.star.bnl.gov	30	898.57
orbiter.txcorp.com	30	3.61
cyber.txcorp.com	30	11.88

Timing results for *offline.auau200.full.sql*:

DB Host	# Repetitions	Avg Time (sec)
orbiter.txcorp.com	5	4.16
dbx.star.bnl.gov	5	1134.09
orbiter.txcorp.com	10	4.14
dbx.star.bnl.gov	10	1090.29
orbiter.txcorp.com	15	4.1
dbx.star.bnl.gov	15	1616.98

Timing results for *offline.dau200.full.sql*:

DB Host	# Repetitions	Avg Time (sec)
orbiter.txcorp.com	5	2.56
dbx.star.bnl.gov	5	993.93
orbiter.txcorp.com	10	2.66
dbx.star.bnl.gov	10	1256.02
orbiter.txcorp.com	15	2.66
dbx.star.bnl.gov	15	999.29

Timing results for *offline.pp500.full.sql*:

DB Host	# Repetitions	Avg Time (sec)
orbiter.txcorp.com	5	4.2
dbx.star.bnl.gov	5	921.82
orbiter.txcorp.com	10	3.52
dbx.star.bnl.gov	10	921.82
orbiter.txcorp.com	15	3.39
dbx.star.bnl.gov	15	907.23

TECH-X CORPORATION

CWS4DB Database Query Caching and Optimization

- Log performance data for each SQL operation
- Calculate and log JSON and XML payload size
- On average over a dataset the equivalent JSON payload is 8.8 – 10.1 times smaller
- In general an order of magnitude lower bandwidth loading is required with the JSON PHP service

entry_id	query_id	json_size	xml_size	duration	timestamp
44880	5757	97	273	0.00026798248291016	2009-08-11 14:46:58
44881	5758	396	996	0.00047802925109863	2009-08-11 14:46:58
44882	5759	99	275	0.00019717216491699	2009-08-11 14:46:58
44883	5758	467	1205	0.00057792663574219	2009-08-11 14:46:58
44884	5760	94	270	0.00020289421081543	2009-08-11 14:46:58
44885	5761	3310	7498	0.0028619766235352	2009-08-11 14:46:58
44886	5762	62	190	0.00020694732666016	2009-08-11 14:46:58
44887	5763	102	278	0.00020194053649902	2009-08-11 14:46:58
44888	5764	126	312	0.00022315979003906	2009-08-11 14:46:58
44889	5765	7	63	0.00011920928955078	2009-08-11 14:46:58
44890	5766	102	278	0.00022387504577637	2009-08-11 14:46:58
44891	5767	126	312	0.00022315979003906	2009-08-11 14:46:58
44892	5768	7	63	0.00012707710266113	2009-08-11 14:46:58
44893	5769	102	278	0.00021219253540039	2009-08-11 14:46:58
44894	5770	127	313	0.00026917457580566	2009-08-11 14:46:58
44895	5771	7	63	0.00011920928955078	2009-08-11 14:46:58
44896	5772	103	279	0.0002291202545166	2009-08-11 14:46:58
44897	5773	127	313	0.00023388862609863	2009-08-11 14:46:58
44898	5774	7	63	0.00012397766113281	2009-08-11 14:46:58
44899	5775	103	279	0.00020599365234375	2009-08-11 14:46:58
44900	5776	127	313	0.00024199485778809	2009-08-11 14:46:58
44901	5777	7	63	0.00016498565673828	2009-08-11 14:46:58
44902	5778	103	279	0.0002598762512207	2009-08-11 14:46:58
44903	5779	127	313	0.00022602081298828	2009-08-11 14:46:58
44904	5780	7	63	0.00013303756713867	2009-08-11 14:46:58
44905	5781	103	279	0.00022983551025391	2009-08-11 14:46:58
44906	5782	127	313	0.00027108192443848	2009-08-11 14:46:58
44907	5783	7	63	0.00012493133544922	2009-08-11 14:46:58
44908	5784	103	279	0.00025486946105957	2009-08-11 14:46:58
44909	5785	127	313	0.00022006034851074	2009-08-11 14:46:58

CWS4DB Summary

File Name : star.pp500.full.sql

https://cyber.txcorp.com/orbiter/service/star/OrbiterStarSimulatorService.php
/cache/off/
/format/XML/
/host/local/
/file//tmp/testfiles/star.pp500.full.sql/
/address/http://64.240.154.24/orbiter/service/star/

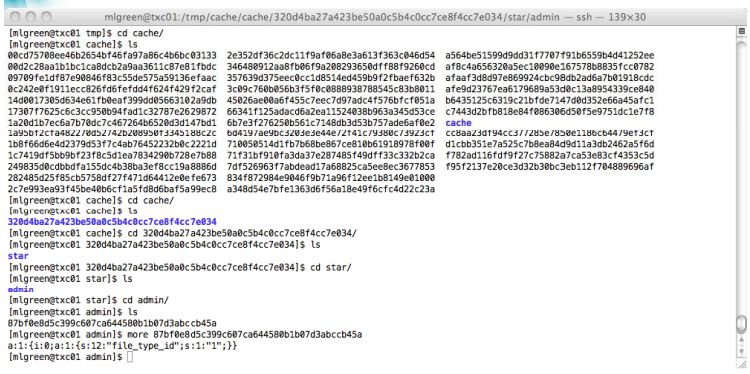
Result:

Number of trials averaged: 1
Total number of queries: 6549
Total size of queries: 38,926,201 bytes
Total query time: 76.9 seconds
Total query rate: 85.1 query/second.

mlgreen@txc01:/tmp/cache/cache/320d4ba27a423be50a0c5b4c0cc7ce8f4cc7e034/star/admin — ssh — 139×30

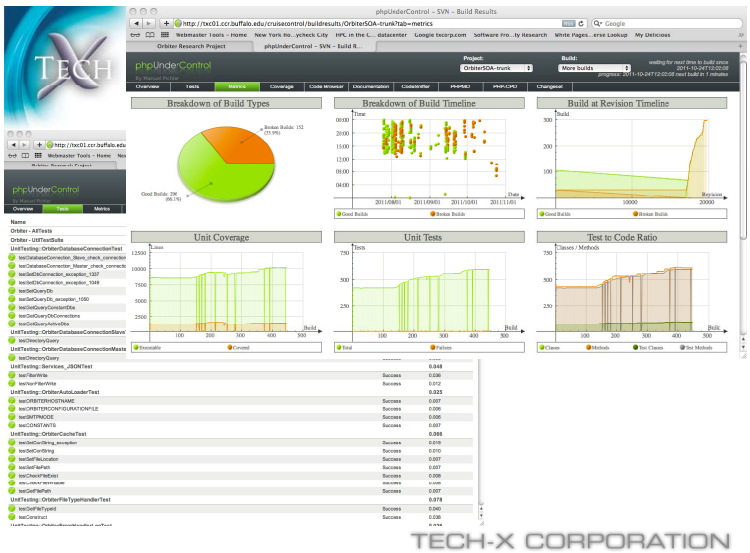
```
[mlgreen@txc01 tmp]$ cd cache/
[mlgreen@txc01 cache]$ ls
00cd75708ee46b2654bf46fa97a86c4b6bc03133  2e352df36c2dc11f9af06a8e3a613f363c046d54  a564be51599d9dd31f7707f91b6559b4d41252ee
00d2c28aa1b1bc1ca8dcb2a9aa3611c87e81fbdc  346480912aa8fb06f9a208293650dff88f9260cd  af8c4a656320a5ec10090e167578b8835fcc0782
09709fe1df87e90846f83c55de575a59136efaac  357639d375eec0cc1d8514ed459b9f2fbaef632b  afaaf3d8d97e869924cbc98db2ad6a7b01918cdc
0c242e0f1911ecc826fd6fefdd4f624f429f2caf  3c09c760b056b3f5f0c0888938788545c83b8011  afe9d23767ea6179689a53d0c13a8954339ce840
14d0017305d634e61fb0eaf399dd05663102a9db  45026ae00a6f455c7eec7d97adc4f576bfcf051a  b6435125c6319c21bfde7147d0d352e66a45afc1
17307f7625c6c3cc950b94fad1c32787e2629872  66341f125adacd6a2ea11524038b963a345d53ce  c7443d2bfb818e84f086306d50f5e9751dc1e7f8
1a20d1b7ec6a7b70dc7c467264b6520d3d147bd1  6b7e3f276250b561c7148db3d53b757ade6af0e2  cache
1a95bf2cfa482270d52742b208950f3345188c2c  6d4197ae9bc3203e3e44e72f41c79380c73923cf  cc8aa23df94cc377285e7850e1186c64479ef3cf
1b8f66d6e4d2379d53f7c4ab76452232b0c2221d  710050514d1fb7b68be867ce810b61918978f00f  d1cbb351e7a525c7b8ea84d9d11a3db2462a5f6d
1c7419df5bb9bf23f8c5d1ea7834290b728e7b88  71f31bf910fa3da37e287485f49dff33c332b2ca  f782ad116fdf9f27c75882a7ca53e83cf4353c5d
249835d0cdbbdfa155dc4b38ba3ef8cc19a8886d  7df526963f7abdead17a68825ca5ee8ec3677853  f95f2137e20ce3d32b30bc3eb112f704889696af
282485d25f85cb5758df27f471d64412e0efe673  834f872984e9046f9b71a96f12ee1b8149e01000
2c7e993ea93f45be40b6cf1a5fd8d6baf5a99ec8  a348d54e7bfe1363d6f56a18e49f6cfc4d22c23a
```

CWS4DB Summary



CWS4DB Tasks Continued

- Task 6: Develop Fault Resilient Data Resource Pathways (Tech-X)
  - Eliminated a single point of failure for the STAR C++ API bound codes database query requests.
- Task 7: Develop a Prototype On-Demand Data Resource Node (Tech-X & BNL)
  - Prototyped the deployment of a on-demand data resource node to meet the dynamic data demands of the STAR collaboration.
- Task 8: Prototype Pre-Cache Capabilities for Production Job Workflow (Tech-X & BNL)
  - Pathway for an authenticated and authorized user upon configuration of the CWS4DB system to execute the customizable site specific test suite for pre-caching production job queries is complete.
- Task 9: Develop a Customizable Site Specific Test Suite (Tech-X)
  - In order to deliver a high quality of service infrastructure a customizable and site specific test suite is required to validate and verify the performance and data delivery capabilities of the CWS4DB system.



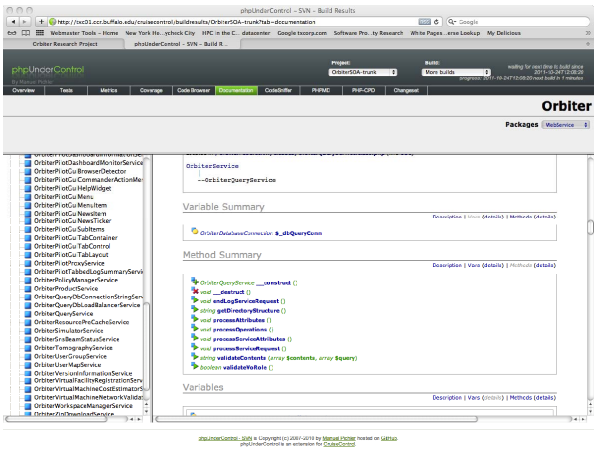
Project Summary

PHP Code:

- 92 classes
- 642 functions/methods
- 10200 lines

API:

- Includes source code links
- Usage
- Dynamically updated



Orbiter Federation SOA: Python Client Service Access Example

```
#!/usr/bin/python
import os, sys, base64, hmac, commands, time
from hashlib import sha1 as sha
from urllib import urlencode
from urllib import urlopen
from urllib import quote_plus

myhome = os.environ.get('HOME')
os.environ['TZ']='GMT'
time.tzset()

idfile = open(myhome + "/.orbiter/my.id")
ACCESS_KEY = idfile.read().strip()
idfile.close()
keyfile = open(myhome + "/.orbiter/user.key")
PRIVATE_KEY = keyfile.read()
keyfile.close()

URI = sys.argv[1]
EXPIRES = str(int(time.mktime(time.localtime(time.time()+60))))
str = URI + '/OrbiterAccessKeyId/' + ACCESS_KEY + '/Expires/' + EXPIRES
SIGNATURE = base64.b64encode(hmac.new(PRIVATE_KEY, str, sha).digest()).strip()
print urlopen(str + '/Signature/' + SIGNATURE, params).read()
```

Scripts and libraries are also available for: C/C++, CURL, Java, Python, PHP that can access the Orbiter Federation.

Orbiter Federation – SOA

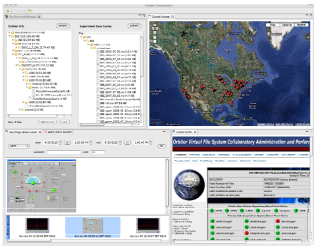
- Orbiter Federation addresses security threats facing web services as identified by Web Services Interoperability Organization which include:
  - **Message alteration:** Attackers cannot alter an Orbiter request without breaking the RSA SHA hash signature. Orbiter will reject a request that does not match canonical string signed resource identifier for the specified Orbiter access key ID.
  - **Loss of confidentiality:** The SSL protocol ensures that Orbiter service transactions are handled privately and provides transport-level encryption.
  - **Falsified messages:** Secure Orbiter services cannot be reached without a signed canonical string resource identifier that matches the signature for the specified Orbiter Federation SOA resource address.
  - **Man in the middle:** The SSL protocol prevents an attacker from reviewing requests and responses send securely between the Orbiter Federation SOA web services and their clients.
  - **Principal spoofing:** The Orbiter infrastructure is the only provider of valid Orbiter access key identifiers and RSA private keys that are authorized to use Orbiter Federation SOA secure web services.

Orbiter Federation – SOA

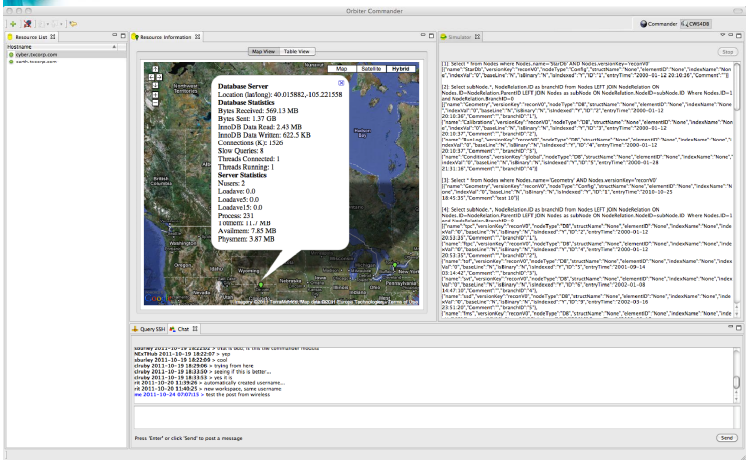
- Orbiter Federation addresses security threats continued:
  - **Forging claims:** Attackers cannot create valid Orbiter Federation SOA service requests without obtaining an Orbiter access key identifier and valid RSA private key from the Orbiter Federation SOA authentication/authorization infrastructure.
  - **Replay of message:** Attackers cannot repeat a RESTful request to secure Orbiter Federation services, as subsequent identical requests will be rejected. Attackers cannot alter the user-provided expiration time without breaking the RSA signature.
  - **Replay of message parts:** An Orbiter RESTful service request is not complete without a valid signature that is applied to all other message parts. Attackers cannot construct a new request from any part of a previous request without altering the service request canonical string resource identifier and generating a valid signature.
  - **Denial of service*:** The denial of service propensity is greatly reduced by the listed security measures in place at the current time within the Orbiter Federation SOA. Furthermore, more specific measures are planned which will ban specific offending IP addresses to further reduce the threat. *Distributed Denial of Service (DDOS) attacks, however, are extremely difficult to defend against utilizing known security measures.

Orbiter Commander – Thick Client

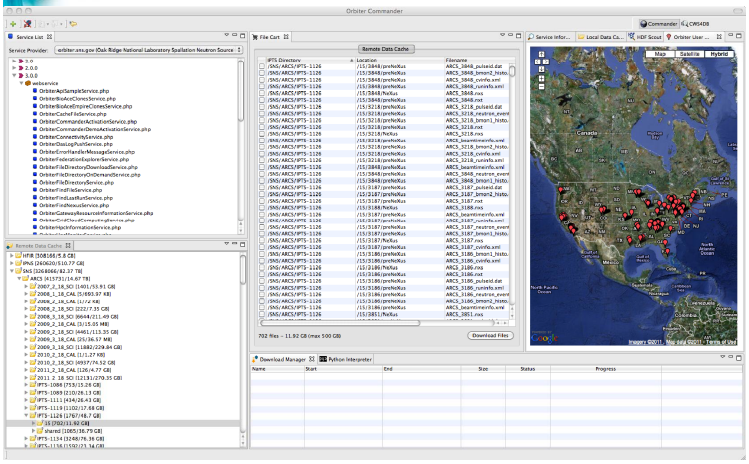
- Tier III of the Orbiter Multitier Portal Architecture
- Orbiter Commander is a Application Framework where a:
  - **Application Framework** defines how to solve common problems, not solutions themselves
    - Using well placed abstraction layers
    - Defining points for extending functionality
- Utilizes Eclipse Rich Client Platform for generating multi-platform applications
  - Mature Integrated Development Environment
  - Simplified Standard Widget Toolkit
  - Automatic handling of core GUI implementation
- Built on Orbiter Federation Services
- Modular design for plug-and-play capabilities
- Well-defined extension points for code-reuse and future development
- Highly customizable interface



STAR Commander Implementation

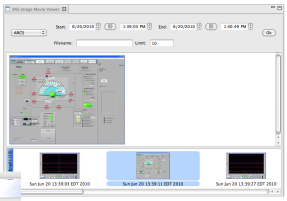


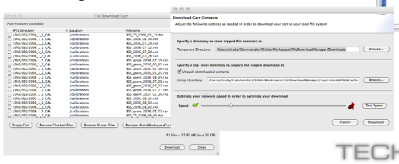
Commander Explorer Implementation



Orbiter Commander – Thick Client (continued)

- Atomic capabilities are provided as *modules* that can be installed as needed from a central module repository
- The Orbiter Federation RESTful SOA provides robust access to diverse capabilities, such as:
  - Multi-threaded streaming downloads of repository files
  - Live status monitoring of the beam
  - Slideshows of instrument application screenshots
  - Organization of modules into "*Suites*"





For More Information

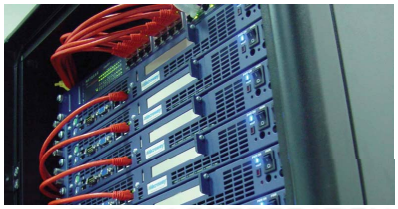
Contact:

Mark L. Green, Vice President of Systems Integration

716-204-8690

mlgreen@txcorp.com

https://orbiter.txcorp.com



3.2 Research Project Website

We have produced a website on the Orbiter project which details our research progress and capabilities. This site is available at https://orbiter.txcorp.com, and is attached in the following pages.

About the Orbiter Project

ORBITER
About the Project
Orbiter Federation Services
Orbiter Pilot
Orbiter Commander
Orbiter Collective
Release Schedule

Orbiter is an end-to-end framework delivering fast and secure solutions through both thin-client web access and thick-client desktop application suites and modules. These applications leverage the information-sharing capabilities of Orbiter in providing powerful and personalized web-accessible components.

Industry standards

Service Oriented Architectures (SOAs) have been proven to be a popular design for building reliable and scalable large-scale software systems, borrowing from earlier Object Oriented Programming (OOP) techniques of encapsulation, cohesion, and the use of abstraction layers behind well-defined public APIs. Orbiter Federation services, built upon industry standards, offer fast and secure access to a wide range of capabilities.

Diverse Applications

Through the Multitier Portal Architecture (MPA) Orbiter Federation services are delivered directly to end-users via a variety of rich interactive interfaces. The MPA allows increasingly sophisticated capabilities to be rapidly developed to suit a wide range of user requirements, and the foundation provided by Orbiter Federation enables these capabilities to be delivered swiftly and securely to end-users

- ***Federation*** provides a Service Oriented Architecture (SOA) of web services, delivering powerful, lightweight, secure, and scalable capabilities. Learn more...
- ***Pilot*** delivers *Federation* web services through web-accessible thin-clients. These gateway and portal clients deliver Orbiter capabilities through easy-to-use web interfaces. Learn more...
- ***Commander*** is a rich cross-platform desktop application that provides access to *Federation* services while allowing Orbiter systems to interact directly with local compute resources. Learn more...
- ***Collective*** opens the door to advanced collaboration across a wide range of associations, facilities, and institutions. Orbiter meets the needs of these organizations through the development of integrated cross-platform applications that enable the full value of third party products and services. Learn more...

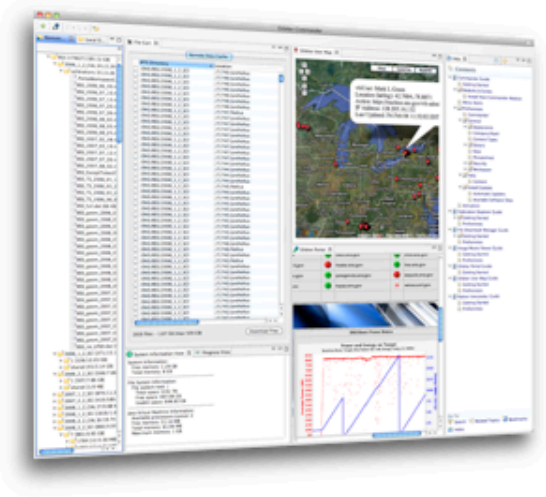
At right: The thin-client ***Orbiter Pilot*** instance at a DOE Laboratory utilizes the *Federation* services to present valuable information to instrument scientists and facility users. By combining these services and graphical user interface modules developers can rapidly produce robust dashboards and monitors for their user base.

Orbiter Pilot, Orbiter Commander, and the Orbiter Collective demonstrate how access to Orbiter Federation resources and services can be provided through reliable, scalable, and interactive scientific

gateways. Its flexible cross-platform desktop solutions present the user with a rich and customizable interface to data, information, computational resources, and enterprise application bases. Orbiter solutions are inherently scalable, where **Federation**, **Pilot**, **Commander**, and **Collective** each build modular capabilities that are focused on particular needs. Orbiter solutions have been routinely used for the management and retrieval of large amounts of data and information.

Versatile Solutions

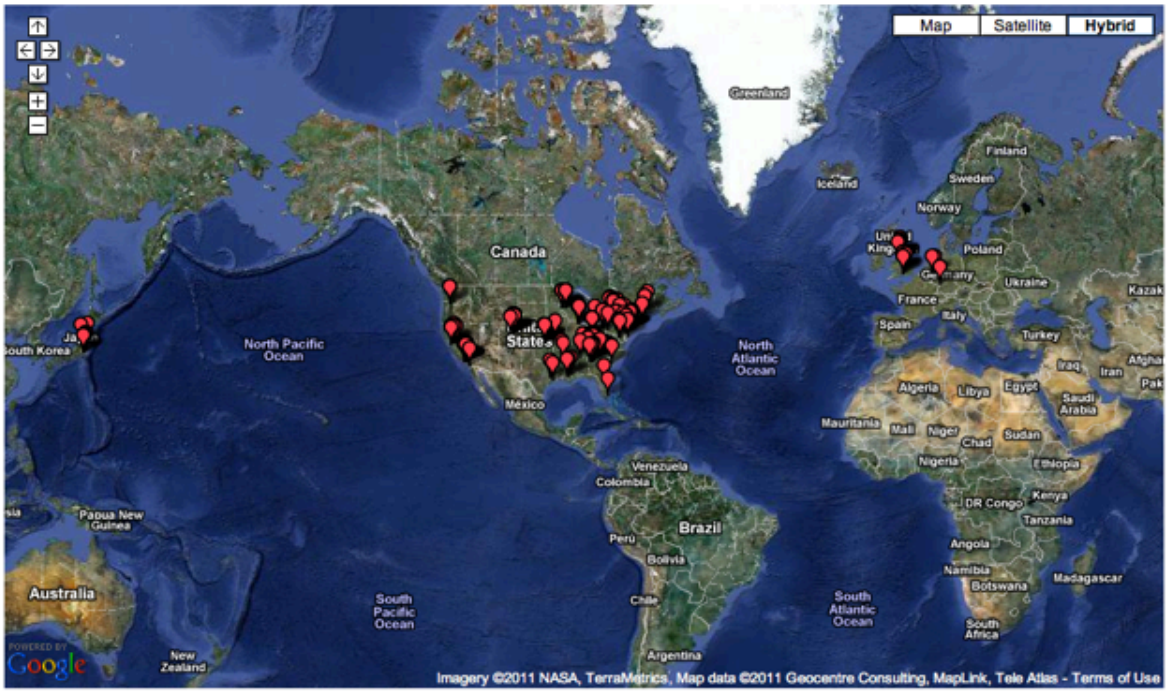
Orbiter is capable of providing a wide variety of solutions, from file management applications to monitoring and information gathering services. Sophisticated dashboards and end-user interfaces can be easily customized for a wide variety of solutions. Orbiter Federation and Pilot run on virtually any architecture that supports modern web servers. Pilot is accessible via any modern web browser, and the Commander desktop application and Collective are supported on Windows, Mac OS X and Linux.



At left: The thick-client ***Orbiter Commander*** deploys a suite of modules in a layout that can be customized by the user. From experiment data cache management and image viewers to process/system information and user-level help, this Commander instance demonstrates the power and flexibility of Orbiter integrated solutions.

Worldwide Accessibility

Orbiter's wide array of capabilities are accessible to users all over the world, where its foundation in industry standards facilitates scalable and global remote access to data and key features. **We currently support over 100 Pilot users and over 50 Commander users on three continents**, and this number continues to grow.



Above: Markers on this map indicate geographic locations of current ***Orbiter Pilot*** and ***Orbiter Commander*** users. As we are continually developing new features for these applications, this number continues to grow.

Find publications relating to the Orbiter Project here.

For project details, please contact Orbiter PI,
Mark L. Green.

Orbiter Federation Services

Orbiter services are implemented as Representational State Transfer (RESTful) web services that deliver functionality through a well-defined API. These services employ robust security standards including SSL and signed requests that ensure client identities, the integrity of their RESTful service calls, and the privacy of their transmissions.

ORBITER

About the Project

Orbiter Federation Services

Orbiter Pilot

Orbiter Commander

Orbiter Collective

Release Schedule

Secure Services

Orbiter Web Services use SSL encryption, access key identifiers, timestamps, and private key signatures, ensuring the **privacy**, **authorization**, and **request integrity** of all interactions. These security measures have allowed the Orbiter Federation SOA to pass a DOE National Laboratory security audit for usage and deployment at their site. These measures completely **address 8 of the 9 top security threats** facing web services as identified by Web Services Interoperability Organization which include:

Message alteration: Attackers cannot alter an Orbiter request without breaking the RSA SHA hash signature. Orbiter will reject a request that does not match canonical string signed resource identifier for the specified Orbiter access key ID.

Loss of confidentiality: The SSL protocol ensures that Orbiter service transactions are handled privately and provides transport-level encryption.

Falsified messages: Secure Orbiter services cannot be reached without a signed canonical string resource identifier that matches the signature for the specified Orbiter Federation SOA resource address.

Man in the middle: The SSL protocol prevents an attacker from reviewing requests and responses send securely between the Orbiter Federation SOA web services and their clients.

Principal spoofing: The Orbiter infrastructure is the only provider of valid Orbiter access key identifiers and RSA private keys that are authorized to use Orbiter Federation SOA secure web services.

Forging claims: Attackers cannot create valid Orbiter Federation SOA service requests without obtaining an Orbiter access key identifier and valid RSA private key from the Orbiter Federation SOA authentication/authorization infrastructure.

Replay of message: Attackers cannot repeat a RESTful request to secure Orbiter Federation services, as subsequent identical requests will be rejected. Attackers cannot alter the user-provided expiration time without breaking the RSA signature.

Replay of message parts: An Orbiter RESTful service request is not complete without a valid signature that is applied to all other message parts. Attackers cannot construct a new request from any part of a previous request without altering the service request canonical string resource identifier and generating a valid signature.

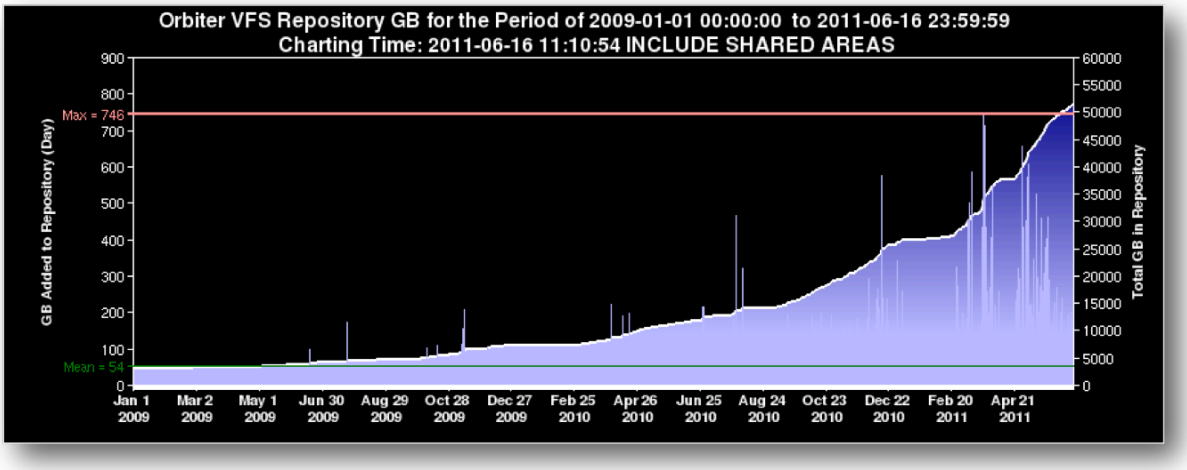
Denial of service*: The denial of service propensity is greatly reduced by the listed security measures in place at the current time within the Orbiter Federation SOA. Furthermore, more specific measures are planned which will ban specific offending IP addresses to further reduce the threat. *Distributed Denial of Service (DDOS) attacks, however, are extremely difficult to defend against utilizing known security measures.

Scalable Infrastructure

The Orbiter Federation SOA is able to provide these security measures without a significant impact on quality of service or turnaround time. A *noop* ("no operation") service with full validation, authorization, and connection to the underlying data layer has been timed at $1x10^{-6}$ seconds. The average Orbiter

Federation full service requests have been observed to take roughly 2.1 seconds to complete, depending on the request. The Orbiter Virtual File System (VFS), built on Orbiter Federation services, has fielded over 50 million of these secure service requests since February of 2010.

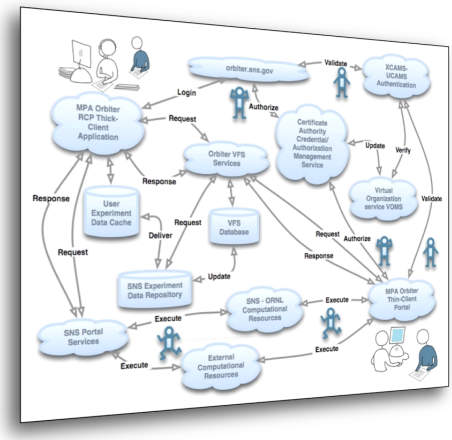
The flexible and secure infrastructure of the Orbiter Federation SOA allows it to provide robust Software-as-a-Service (SaaS) capabilities to its clients. Orbiter is capable of providing complex objects, dynamic charts, raw data, and data files, forming a solid foundation of valuable resources upon which end-user interfaces can be built.



Above: The ***Orbiter Federation*** Virtual File System combines a series of Web Services and thin- and thick-client interfaces to support experiment data at a DOE National Laboratory. As this graph indicates, VFS usage has swelled over the years to support **more than 3.7 million files** totalling **over 51 Terabytes of data**, and fielding **millions of secure service requests**.

Federation SOA in Action

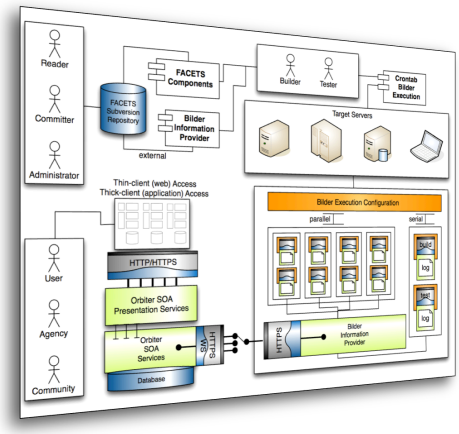
Orbiter Federation Web Services are already being used to support a number of initiatives at the Tech-X Corporation, enabling thin- and thick-client interfaces that support a number of users. Highlights include:



Orbiter Virtual File System

The Orbiter Virtual File System (VFS) supports facility and experiment repository data access for the DOE National Laboratory. The **Orbiter VFS** provides services that enable (a) summary dashboards and Quality of Service (QoS) metrics, (b) data repository NeXus file full text search capabilities, (c) fully functional role-based file browser, (d) user/group defined metadata for data repository files, (e) user, group, repository, and web 2.0-based global positioning with additional service capabilities.

Learn more...



Orbiter Bilder Dashboard

Orbiter Federation services support **Bilder**, an internal LCF-applicable unix-oriented package management system. These services enable the Orbiter Pilot **Bilder Dashboard** and the Orbiter Commander **BilderBench**, which use current and historical cross-platform build information to provide a number of different capabilities that support Tech-X users.

For project details, please contact Orbiter PI, Mark L. Green.

Orbiter Pilot

Orbiter Pilot is a thin-client web interface, leveraging the Orbiter Federation Service Oriented Architecture to deliver a wide range of capabilities. Any modern browser can access Orbiter Pilot, giving users and developers a low barrier of entry with a simple point of access to a diverse set of applications.

Versatile Capabilities

Orbiter Pilot directly leverages the Orbiter Federation Service Oriented Architecture, allowing it to implement a presentation layer over a wide range of capabilities. Using dynamic web technologies such as AJAX, Orbiter Pilot pages are able to call any of the Orbiter Federation SOA services to deliver automatically-updating information to the end-user. Advanced JavaScript and jQuery components allow Orbiter Pilot interfaces to be dynamic and customizable, providing a rich look and feel.

Pilot in Action

Orbiter Pilot is already being used to support a number of initiatives at the Tech-X Corporation, providing interfaces that support a number of users. Highlights include:

Orbiter VFS Collaboratory

The Orbiter VFS Collaboratory presents a thin web client for interacting with the **Orbiter Virtual File System**, a Tech-X system at a DOE National Laboratory using Role-Based Access Control for data repository file access and computational job workflow management. The Orbiter VFS Collaboratory presents bird's eye dashboard views, interactive charts showing both current and historical information, and user management capabilities.

Learn more...

Orbiter Bilder Dashboard

The Bilder Dashboard supports **Bilder**, an internal LCF-applicable unix-oriented package management system. Orbiter Federation services enable this Orbiter Pilot Bilder Dashboard, which collects, manages, and delivers both historical and current build results for a number of Tech-X products through an interactive web application. Interactive charts and dynamic content create a rich interface for managing Bilder results.

Learn more...

Orbiter ESS

The **Experiment Scheduling System**, developed by Tech-X for a DOE National Laboratory, manages scheduling experiments and equipment for instruments at these facilities. This role-based multi-user system allows the ORNL user office and instrument scientists to manage scheduling hundreds of experiments, optimizing the utilization of laboratory resources.

For project details, please contact Orbiter PI, Mark L. Green.

Orbiter Commander

ORBITER
About the Project
Orbiter Federation Services
Orbiter Pilot
Orbiter Commander
Orbiter Collective
Release Schedule

Orbiter Commander is a thick-client Java desktop application, leveraging the Orbiter Federation SOA to deliver versatile capabilities to end-users. This desktop application allows users to seamlessly stay connected to Orbiter Federation web services while harnessing the resources of their local machines.

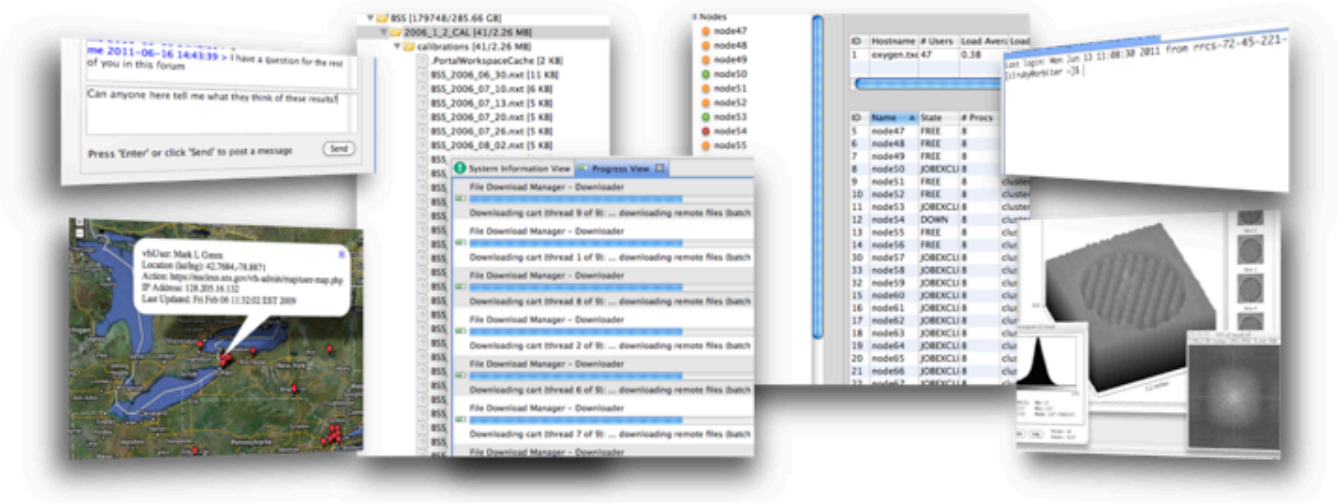
An Application Framework

Orbiter Commander is built on the Eclipse Rich Client Platform (RCP), an architecture that allows a robust and powerful Graphical User Interface (GUI) with a sophisticated interface and a rich overall user experience. Eclipse RCP utilizes the Eclipse Equinox runtime, an implementation of the OSGi specification for modular Java applications.

In leveraging the Eclipse Rich Client Platform Orbiter Commander is able to itself serve as a robust **application development framework**. The core Commander architecture takes responsibility for the basic capabilities of the application, providing the foundation for the development of highly customizable end-user products. Commander defines a set of plug-in extension points for defining Commander-compliant suites and modules. The core framework implementation performs much of the basic module functionality enhance these extension points, and a growing set of utilities and visual widgets provide a solid base for rapidly developing sophisticated Commander modules. This allows Commander applications to be **rapidly tailorable** to the needs of an individual user base.

Modular Design

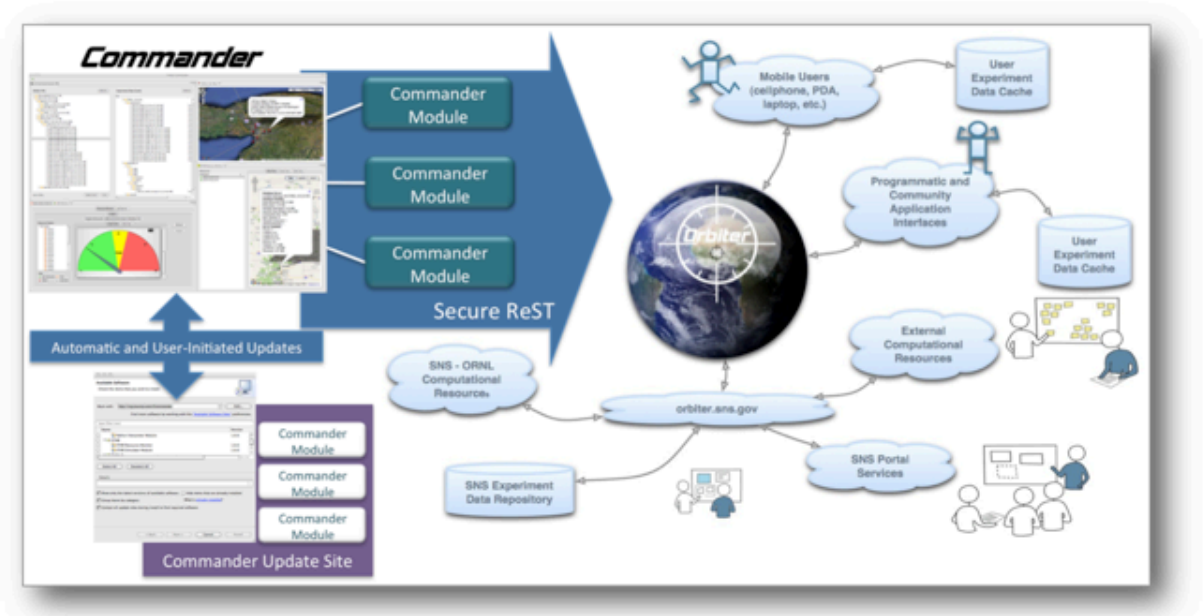
Commander is composed of suites of individual **modules**, or functional components that provide valuable capabilities to the end-user. These independent and individually provisionable modules provide a wide range of capabilities, from **interactive chat** to **remote file browsing**, **SSH integration**, live **HPC monitoring**, **Software-as-a-Service (SaaS)** interaction, and more. Commander modules interact with the Orbiter Federation Service Oriented Architecture, providing seamless access to the power of its available web services. The Orbiter Federation SOA coupled with the solid Commander **Application Framework** enables the rapid development of an **endless variety** of new modules.



Above: The ***Orbiter Commander*** Application Framework enables a wide range of capabilities, from interactive chat to remote file browsing, multithreaded file downloading, integration with SSH, live HPC and job monitoring data, Software-as-a-Service (SAAS) integration, and more.

Seamless Service Interaction

Commander's implementation as a desktop application allows it achieve the best of both worlds in terms of online and offline computing. The Orbiter Federation SOA on its Logic Tier ensures secure web service interaction and allows users to fully leverage the resources available via the Orbiter infrastructure. However, because Commander is accessed locally, offline capabilities can be provided when networks are poor or unavailable, such as while users are traveling or working remotely on slow connections. That is, service calls can be cached for when the network becomes available again, and Commander may also poll commonly used services to built a usable system state that can be referred to in the event that connectivity is lost. In addition, computationally- or data-intensive capabilities can be provided locally such that users can continue their work offline.



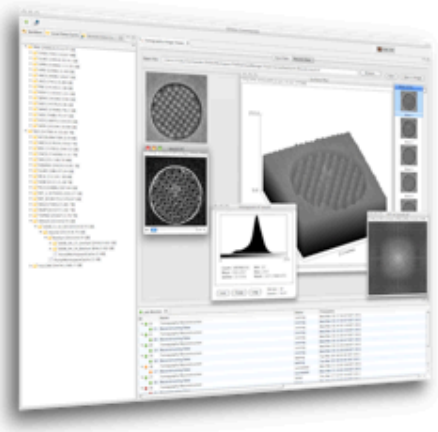
Above: The ***Orbiter Commander*** thick-client application seamlessly integrates with the Orbiter Federation SOA to deliver a wide variety of application to end-users. Its individual functional components, or modules, are individually installable and updatable through an integrated web-enabled wizard, permitting maximum flexibility and seamless capability provisioning.

Commander in Action

Commander is already being used to support several projects at Tech-X Corporation, providing a wide range of capabilities for end-users on a number of initiatives. Highlights include:

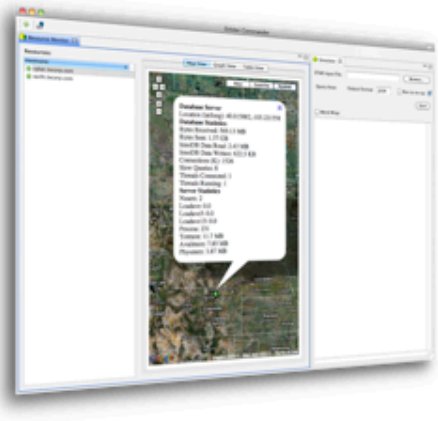
Commander NExTHUB

Commander offers a wide range of capabilities. Users can browse their remote Virtual File System (VFS) data and download it to their local files through a **multi-threaded streaming download**. VFS usage information is displayed as an interactive map, while users can also get direct access to the Orbiter Pilot portal at the SNS for live and historical usage and performance statistics.



Commander CDAS-RIP

This suite supports the Computational and Data Analysis System for Multi-Technique Rapid Tomography Reconstruction and Quantification Processing. Users can browse remote data files on-the-fly and invoke a Software-as-a-Service enabled tomography reconstruction job given raw FITS files. Live job monitoring shows the user the progress on their reconstruction jobs, while an ImageJ-enabled module allows them to review their results.



Commander STAR

This suites supports our collaboration with the Solenoidal Tracker at the Relativistic Heavy Ion Collider (STAR) experiment located at a DOE National Laboratory. Commander is capable of simulating Nuclear Physics database queries on distributed database resources, and provides a resource monitoring interface to support our contribution to the project.

Commander BilderBench

BilderBench supports **Bilder**, an internal LCF-applicable unix-oriented package management system. Orbiter Federation services support the Orbiter Pilot **Bilder Dashboard**, which collects, manages, and delivers both historical and current build results for a number of Tech-X products through an interactive web application.

Currently under alpha-testing, BilderBench aims to add additional capabilities to enhance what is offered by the Bilder Dashboard, including integration with SSH and an interactive chat interface.

Commander Tech-X

Demonstrated at the Supercomputing 2010 conference in New Orleans, this suite features live Tech-X product information and an interactive chat interface, inviting all users to communicate about Tech-X Products and Services. This suite was also used to give a live talk on the Orbiter infrastructure, using Orbiter Federation services to deliver content to the podium computer in real-time.

Future development on this suite will enable other functionality to support our company in its collaboration and marketing activities.

For project details, please contact Orbiter PI, Mark L. Green.

Orbiter Collective

The Orbiter Collective is the fourth tier of the Orbiter infrastructure, pulling together the Federation SOA, thin-client Pilot and thick-client Commander Application Framework to provide a set of tools for customizing a wide variety of end-user interfaces. The Orbiter Collective **allows anyone to be the developer**, providing a platform exposing Orbiter capabilities as plug-and-play components.

Secure Services at Your Fingertips

The experienced developer has been able to leverage the **Orbiter Federation SOA** from the beginning, using the standards-based implementation of its web services. **WSDL 2.0** support defines Orbiter web service parameters, allowing service capabilities to be automatically consumed and used within other applications. Client-configurable response formats are enabled through the simple RESTful web service interface, allowing responses to be consumed as **XML**, **JSON**, and other custom formats.

Using secure Web Service signatures is as easy as signing up for an Orbiter account. Bindings for creating signed service requests are already available in **Java**, **BeanShell**, **Perl**, **Python**, **C++**, **JavaScript**, and **PHP**, and this set continues to grow. Contact us to find out how we can get you started using secure Orbiter Federation services and applications today.

Pilot Your Dashboard

The thin-client **Orbiter Pilot** is already using Orbiter Federation web services to seamlessly deliver its underlying capabilities to the end-user. Web-based forms, the traditional means of providing interactive and configurable web interfaces, have provided a wide range of powerful interfaces for Pilot uers.

Our next step moves **beyond the form**, delivering infinite possibilities through a fully configurable web application platform. Exposing Orbiter Federation services as pluggable web components, we can allow on-click customization of service calls that can be saved as drag-and-drop interface widgets. Users will be able to create their fully customized dashboard interface in minutes, permitting even the most inexperienced users to consume Orbiter services as soon as they are created.

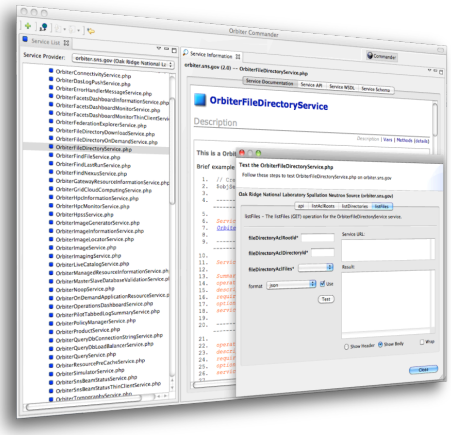
Command Your Application

The thick-client **Orbiter Commander** is already using Orbiter Federation web services to deliver its wide range of modules to the end-user. By leveraging the Eclipse Rich Client Platform (RCP) and Eclipse's p2 provisioning system Commander is able to deliver seamlessly updateable modules through a point-and-click interface. Users can add new capabilities to their Commander instance as they are published to the designated Commander update site.

Our next step **puts the user in the driver's seat**, allowing users to create their own modules directly using Orbiter Federation web services. The flexible Commander Application Framework will allow users to browse available services and incorporate them directly into their installation, allowing them to generate new interfaces and modules on-the-fly with a simple step-by-step configuration process.

The Collective in Action

We have already begun laying the groundwork for the Orbiter Collective, with several examples of this

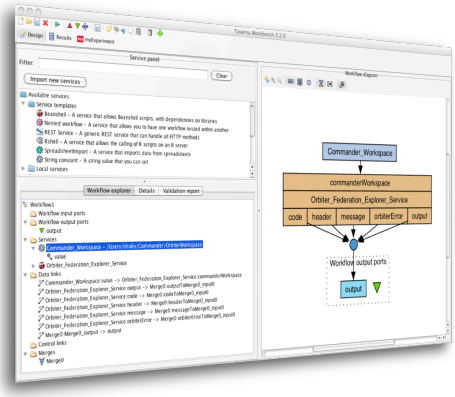


Orbiter Federation Explorer

Commander provides the **Orbiter Federation Explorer**, a module which will display all available Orbiter Federation SOA providers and services. Selecting a service will automatically retrieve its service documentation, API, WSDL 2.0 and parameter schema. Users can interact with these services directly using a dynamically-generated testing interface. We have begun laying the groundwork for an **export** feature, allowing the user to export a dynamically-created service stub in the language binding of their choosing.

Pilot Service Explorer

Current initiatives on the Orbiter project are enhancing the capabilities of the Orbiter Pilot interface to allow Orbiter Federation service browsing through a web interface, similar to what is provided through Commander. This represents the first steps in developing a fully user-customizable **mashup** interface, where users will be able to save configured service calls and add them to their dashboards. Future work will enable rich drag-and-drop functionality, allowing users to create new dashboard widgets in minutes.



Workflow Management Integration

Scientific Workflow Management Systems have become popular as point-and-click tools for designing complex workflows and **in silico** experiments. Systems like the open source Taverna suite provide simple ways of hooking together various remote and local processes to create repeatable workflows. We have already proven that we can integrate Orbiter Federation **secure web services** into a Taverna scientific workflow. Future work will involve streamlining this process and making the Orbiter Federation SOA even more accessible to other popular workflow managers.

For project details, please contact Orbiter PI, Mark L. Green.

TECH-X CORPORATION

PRODUCTS | SERVICES | CORPORATE | RESEARCH | DOWNLOADS | Google™ Custom Search | SEARCH

Orbiter Project Releases

Orbiter components are released in families, with each major version marking a significant milestone in the development of the project. Below we have listed each major version release, as well as future releases scheduled for Orbiter.

ORBITER
About the Project
Orbiter Federation Services
Orbiter Pilot
Orbiter Commander
Orbiter Collective
Release Schedule

Apollo: January 15, 2011

Our debut, this version marked the first synchronized release of the Orbiter Federation Services, Orbiter Pilot thin-client and Orbiter Commander thick-client.

Taurus: June 7, 2011

This version refined the Orbiter Federation service API to include a WSDL 2.0, service schema, and defined service operations. Orbiter Commander's capabilities grew and several interfaces were refactored for cleaner and more intuitive interactions.

Orion: July 25, 2011

New support for Orbiter Commander modules and suites, enhanced service operation and attribute validation and checking, more complete and comprehensive error messages for failed service requests, and better API and documentation text.

Delphinus: November 2011

We are working toward improved Software-as-a-Service (SaaS) multi-core and GPU-accelerated application support. In addition we are streamlining integration between the Orbiter Pilot and Commander tiers, as well as expanding the services provided by the Orbiter Federation SOA. Service security will be enhanced using *FlexTrust* security protocols.

For project details, please contact Orbiter PI, Mark L. Green.

3.3 Product Sheet

We have produced a product sheet detailing the capabilities of the Orbiter project which was developed in conjunction with this effort. We have attached the product sheet in the following pages.

TECH-X CORPORATION

ORBITER

Federation - Pilot - Commander - Collective

Integrated Solutions: Orbiter provides a powerful and flexible application framework, from versatile user interfaces to a sophisticated and secure Service Oriented Architecture.

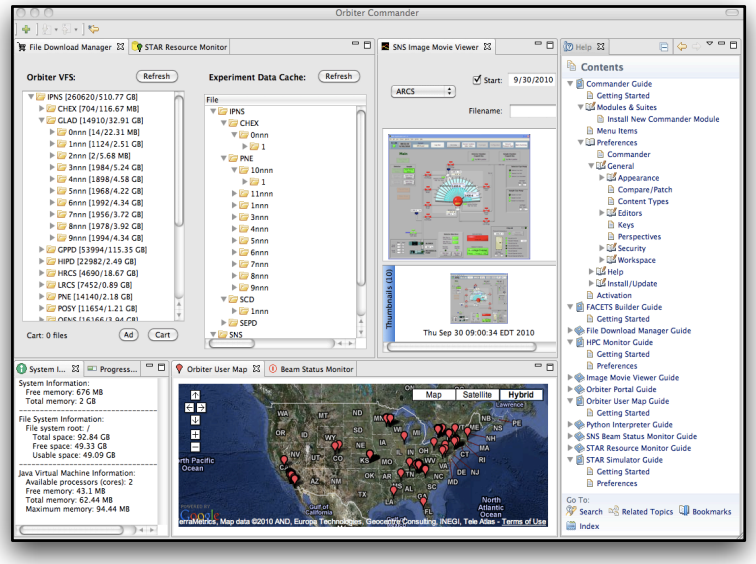
Federation **provides a robust and scalable Service Oriented Architecture of RESTful web services, delivering powerful capabilities via lightweight service calls.**

Pilot **delivers Federation services through a thin web client. This portal, accessible from any web browser, offers Orbiter capabilities in an easy-to-use web interface.**

Commander **is a desktop application that provides access to Federation services while allowing Orbiter capabilities to integrate directly with local resources.**

Collective **opens the door to Orbiter integration with other third party capabilities, laying the groundwork for advanced collaboration across a wide range of technologies.**

ORBITER is an end-to-end framework delivering fast and secure solutions as both thin-client web and thick-client desktop applications. Combining sophisticated end-user applications with a robust Service Oriented Architecture, ORBITER provides a versatile set of solutions to a variety of problems. Through its four layers, Federation, Pilot, Commander, and Collective, ORBITER represents a flexible and highly customizable framework for rapidly developing robust solutions.

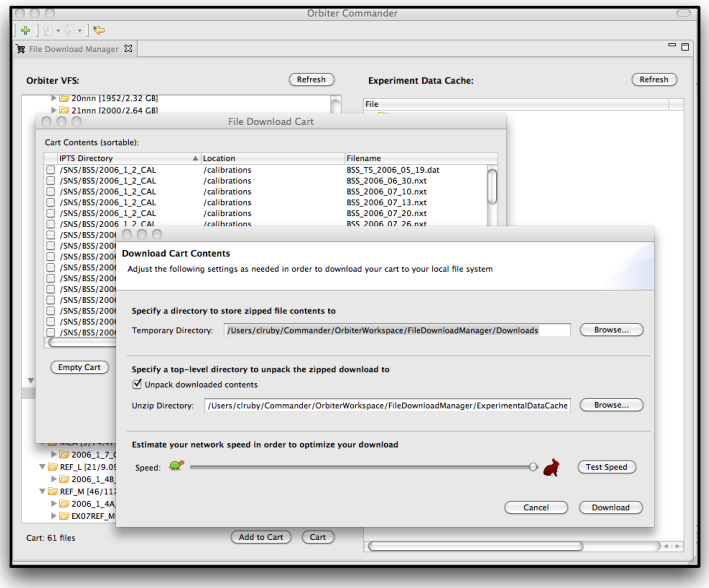


Federation services build on industry standards to deliver fast and secure capabilities to end-user applications. These easy-to-use services encapsulate the business logic of complex applications, serving as a powerful engine for end-user applications. Orbiter's Pilot and Commander offer a variety of rich end-user interfaces, leveraging the capabilities of the underlying Federation layer to build sophisticated and highly tailorable applications. Collective?

TECH X

Scalability

Orbiter solutions are inherently scalable, where Federation, Pilot, Commander, and Collective each build modular capabilities that are focused on particular needs. Orbiter solutions have been applied in a number of situations that require the management and retrieval of large amounts of information.

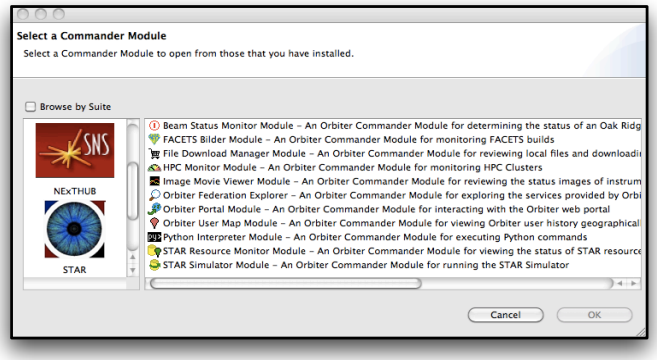


Versatile Solutions

Orbiter is capable of providing a wide variety of solutions, from file management applications to monitoring and information gathering services. Sophisticated dashboards and end-user interfaces can easily be customized for a wide variety of solutions.

Consulting Services

Tech-X Corporation offers product training and application consulting to help you leverage your investment in Orbiter as quickly as possible. We offer **full application development** to tailor Orbiter's Pilot and Commander clients to your specific needs.

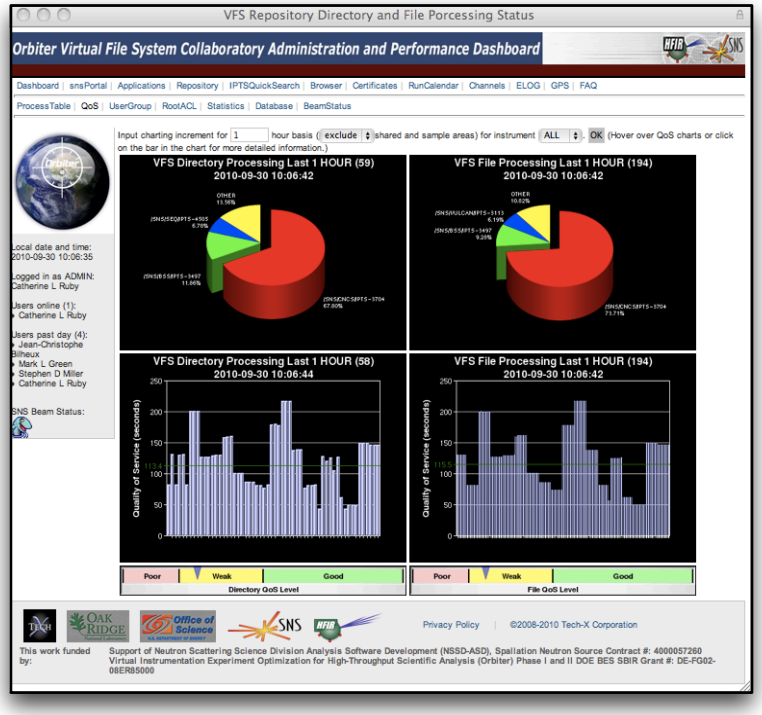


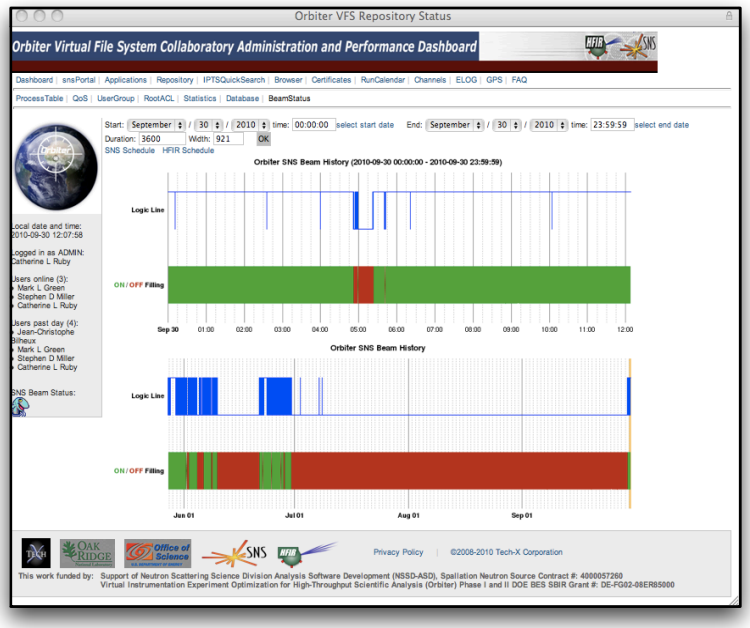
Supports Multiple Platforms and Standard Hardware

Orbiter's Service Oriented Architecture and Orbiter Pilot run on ???. Orbiter's Pilot web portal is accessible via any web browser, and Orbiter's Commander desktop application is supported on Linux, Mac OS X, Windows, Solaris, AIX, and HP-UX.

About Tech-X Corporation

At Tech-X, we address specific research questions in science and engineering by applying our expertise in high-performance computing, modeling and data analysis, physics, grid computing, and HPC infrastructure (visualization, remote data access). Software solutions developed as part of our research efforts also support industries such as aerospace and semiconductor manufacturing.





TECH-X CORPORATION
5621 ARAPAHOE AVE, SUITE A | BOULDER, CO 80303
TEL: +1 303 448 0727 | FAX: +1 303 448 7756
SALES@TXCORP.COM | HTTPS://ORBITER.TXCORP.COM/FACETS/ORBITER

ORBITER

3.4 Project Whitepaper

We have produced an internal company whitepaper detailing the capabilities of the Orbiter project which was developed in conjunction with this effort. We have attached this whitepaper in the following pages.

CWS4DB: A CUSTOMIZABLE WEB SERVICE FOR EFFICIENT ACCESS TO DISTRIBUTED NUCLEAR PHYSICS RELATIONAL DATABASES

May 26, 2011

Contents

This project is funded by Phase I and II DOE BES SBIR Grant : DE-FG02-07ER84757

Keywords: Customizable Web Service, Distributed Relational Database, Nuclear Physics, Code Generation, OGSA-DAI, Globus Grid Middleware, On-Demand Resource, Application Framework, Scalability, Reusability, Client-server Architectures, SOA

1 Executive Summary

An increasing fraction of the data generated in Nuclear Physics (NP) experiments are managed in distributed and relational databases. As the size of this data grows and the collaborative nature of NP experiments increases, the ability to access differently organized relational databases remotely, efficiently and yet in a user-friendly and interoperable manner is becoming very important. Tech-X has developed a customizable Web service, CWS4DB (Customizable Web Service for Efficient Access to Distributed Nuclear Physics Relational Databases), for efficient access to distributed NP databases and NP analysis jobs in ROOT framework by implementing a high-level client emulating the STAR C++ API. This system comprises a generic Web service for accessing arbitrary distributed relational databases, a reference client for the Solenoidal Tracker at RHIC (STAR) at Brookhaven National Lab (BNL) and a tool for the creation of high-level and domain-specific clients. Web services are open standards-based, modular, distributed, dynamics web applications that are self described, published, located, or invoked over the internet. The STAR experiment studies quark-gluon plasma (QGP)[1], state of matter believed to exist at sufficiently high energy densities[2]. This experiment, at BNL, conducts simultaneous studies using several types of specialized detectors. These Web services are able to address the distributed and heterogeneous nature of the databases by building the Web services on top of OGSA-DAI which provided mechanisms for the coordination of various data resources. The problem of providing high-level quires for many different NP applications has been overcome by generating a customized interface on top of the vendor relational database connector APIs for the Web service client.

Each component of the CWS4DB design involved a separate piece of technical functionality that is implemented in way that can be exercised in the STAR computing environment, yet developed in a general way for application to other NP projects. CWS4DB has provided a set of software tools and services that can be easily adapted by the NP application developer. The abstraction of the details of the database query languages is invaluable to making collaborative efforts possible for US physicist in remote experiments in Europe which in turn will allow NP scientist to concentrate more on science. By providing efficient and fast service for application programmers where they do not need to know where the actual data resides is a great benefit to computational scientist.

Relational databases are widely used in many scientific and commercial applications, therefore, by building a generic bridge between distributed relational databases and high-end users CWS4DB will allow for more efficient and productive work. Many scientific applications running on the Grid need to access data from distributed and heterogeneous relational databases. These applications can directly benefit various elements of CWS4DB. As distributed computing based on Service Oriented Architecture becomes more mainstream, the techniques and advances made by CWS4DB will be applicable directly to business and engineering fields where a commercial market for ”computing on demand” and where users wish to access their databases more efficiently already exists.

2 CWS4DB Architecture

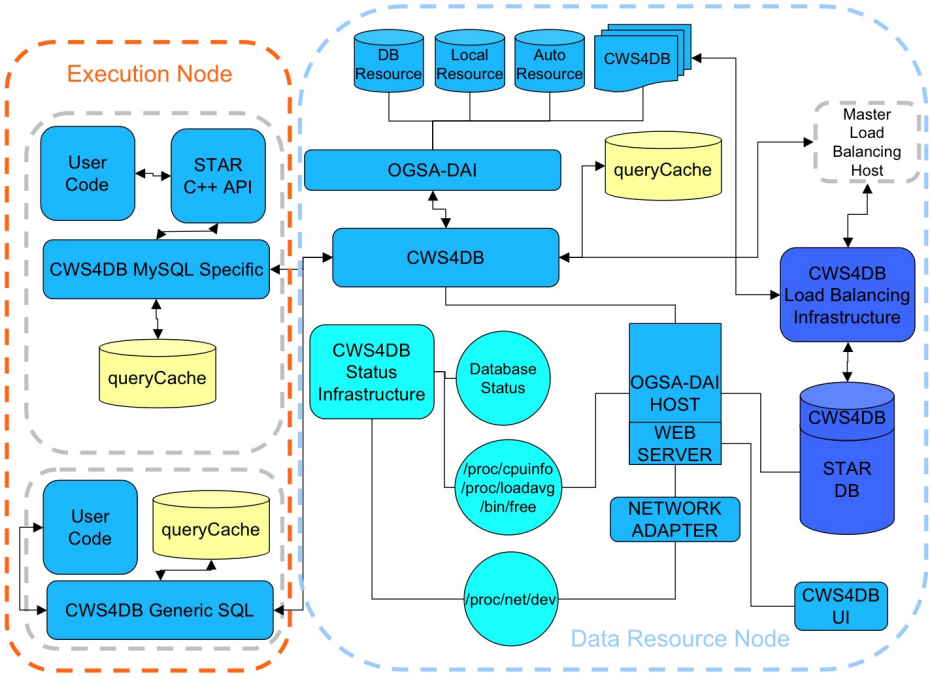


Figure 1: CWS4DB Architecture Diagram

The CWS4DB system architecture, shown in Figure 1 was developed from the successful completion of our previous investigations and input from our Nuclear Physics collaborators who provided co-ordination of the discussions between the sites and the experiments. The CWS4DB system is now a production-quality, load-balanced, auto-caching, grid-enabled, fault-tolerant, on-demand system. It will use the Globus Grid middleware and implement the Web service in Java. As a reference implementation, Tech-X has developed a prototype high-level client emulating the STAR C++ API to use in the NP analysis jobs in ROOT framework.

The CWS4DB web services are built top of OGSA-DAI to take advantage of its support for accessing the distributed and heterogeneous databases as well as the community development. The OGSA-DAI data resources shown at the top of this figure are configured via the CWS4DB User Interface (UI). The CWS4DB MySQL specific or generic SQL provide user code bindings and have the capability of generating Web Service query requests to the Data Resource Node CWS4DB service. The proposed query auto-caching capability described later is denoted in the above Figure 1 as the queryCache, this is where the cached query results are stored on the Execution Node while the Data Resource Node are where the main CWS4DB Web Services reside. The CWS4DB Load Balancing Infrastructure stores the data resource node statistics in a local database or file and also transmits them to the centralized Master Load Balancing Host.

3 Orbiter Integrated Solution

Orbiter is a modular and extensible end-to-end application framework capable of delivering fast and secure solutions through both thin-client web access and thick-client desktop application suites and modules. Application frameworks organize their capabilities in well-defined abstraction layers and provide points for extending functionality for future development. Orbiter solutions are scalable and they include Federation, Pilot, Commander and Collective where each can build modular capabilities that are focused on individual requirements. Orbiter Commander provides a robust and highly customizable user interface built on the Orbiter Federation Service Oriented Architecture. Orbiter is built upon many industry standers and frameworks such as: Representational State Transfer (REST), Web Services Description Language (WSDL), Extensible Markup Language (XML), OSGi R4 core framework specification, and Eclipse Rich Client Platform (RCP).

3.1 Orbiter Federation

Orbiter Federation provides a Service Oriented Architecture (SOA)[3][4] of web services which delivers powerful, lightweight, secure and scalable capabilities. Federation is the data and logic tiers that provide remote functionality. Federation is used to encapsulate much of the lifecycle and infrastructure associated with an Orbiter service implementation, enabling the more streamlined development of all of the Orbiter services. This component allows services to define their properties and business logic in a more straightforward manner, allowing the Orbiter services to be more maintainable and stable as the infrastructure continues to grow. An overview of the Orbiter Federation is shown below in Figure 2

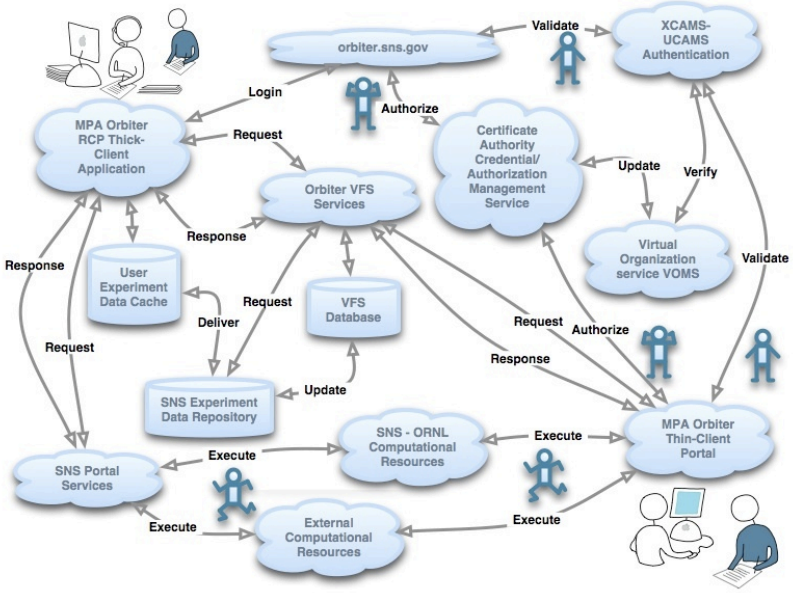


Figure 2: Orbiter Federation SOA Overview

The Orbiter Commander Federation Explorer module provides an interactive interface for browsing and testing Orbiter web services. An Orbiter web service interface was created that returns the base stations, network nodes, and available services for an Orbiter host endpoint. This tool permits Orbiter administrators to interact directly with the Orbiter web services for development and testing purposes. After an Orbiter base station or network node is selected, a list of all Orbiter SOA web services is provided, in groups by package names. Selecting a service will display its service Application Programming Interface (API), Web Service Description Language (WSDL) definition, and a schema, generated by the service itself to describe its capabilities and parameters. Orbiter services are implemented as RESTful web services[5] that deliver this functionality through a well defined API.

3.2 Orbiter Multitier Portal Architecture

The Multitier Portal Architecture (MPA)[6] implements the presentation tier of Orbiter's three-tier client-server architecture. The MPA provides scalable and sophisticated user interfaces by using the Orbiter Federation SOA to build increasingly complex and customizable applications. This layered approach maximizes the reuse of the developed infrastructure and services in successive tiers. Tier I is implemented by the Orbiter Federation SOA and thin-client applets that utilize the portal server for file transfers, visualization caching, and file meta-data transfers. Orbiter Pilot implements tier II of this architecture, providing a thin-client and portlet layer of the MPA for accessing Orbiter services. Orbiter Commander implements tier III of this architecture by providing a desktop application to end-users. Commander utilizes the Orbiter SOA to provide services to end-users, while providing the capability for offline computing and integration with other local desktop applications. Tier IV, the Orbiter Collective, is enabled by the flexible and modular framework provided by Orbiter Commander and the underlying Orbiter SOA. Orbiter Collective is a collaboratory for exchanging information and data and for applying Orbiter capabilities in new ways. An overview of the MPA architecture is shown below in Figure 3

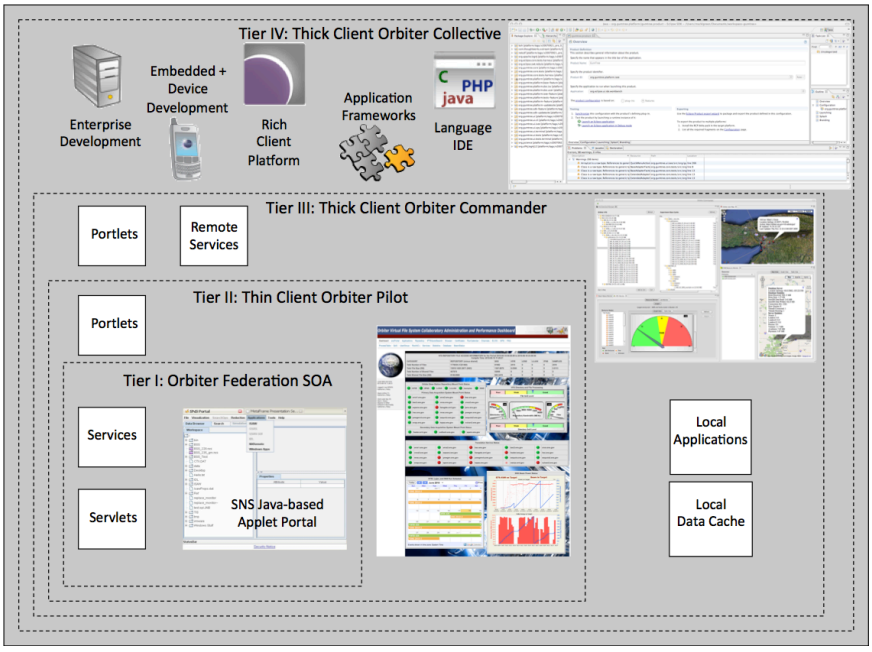


Figure 3: Multitier Protal Architecture

3.3 Orbiter Security Model

The flexible and secure infrastructure of the Orbiter Federation SOA allows it to provide robust Software-as-a-Service (SaaS) capabilities to its clients. The security measures taken have allowed the Orbiter Federation SOA to pass the Oak Ridge National Lab security audit for usage and deployment to their site. These measures address 8 of the 9 top security threats facing web services as identified by the Web Services Interoperability Organization.[7][8] These include:

Message alteration: Attackers cannot alter an Orbiter request without breaking the RSA SHA hash signature. Orbiter will reject a request that does not match canonical string signed resource identifier for the specified Orbiter access key ID.

Loss of confidentiality: The SSL protocol ensures that Orbiter service transactions are handled privately and provides transport-level encryption.

Falsified messages: Secure Orbiter services cannot be reached without a signed canonical string resource identifier that matches the signature for the specified Orbiter Federation SOA resource address.

Man in the middle: The SSL protocol prevents an attacker from reviewing requests and responses send securely between the Orbiter Federation SOA web services and their clients.

Principal spoofing: The Orbiter infrastructure is the only provider of valid Orbiter access key identifiers and RSA private keys that are authorized to use Orbiter Federation SOA secure web

services.

Forging claims: Attackers cannot create valid Orbiter Federation SOA service requests without obtaining an Orbiter access key identifier and valid RSA private key from the Orbiter Federation SOA authentication/authorization infrastructure.

Replay of message: Attackers cannot repeat a RESTful request to secure Orbiter Federation services, as subsequent identical requests will be rejected. Attackers cannot alter the user-provided expiration time without breaking the RSA signature.

Replay of message parts: An Orbiter RESTful service request is not complete without a valid signature that is applied to all other message parts. Attackers cannot construct a new request from any part of a previous request without altering the service request canonical string resource identifier and generating a valid signature.

Denial of service: The denial of service propensity is greatly reduced by the listed security measures in place at the current time within the Orbiter Federation SOA. Furthermore, more specific measures are planned which will ban specific offending IP addresses to further reduce the threat.

4 Key Challenges & Technical Capabilities

By working closely with our STAR BNL collaborators in understanding the database access patterns generated for the typical STAR root4star user and production jobs, Tech-X has determined that a query aggregation algorithm would not be appropriate. This is due to the fact that an average root4star job will generate ˜500K database queries, with the resulting data size of a typical query being ˜120 bytes.

The OGSA-DAI infrastructure has some significant limitations in utilizing SOAP messaging exclusively. We have identified that a RESTful interface for the CWS4DB infrastructure and Data Services will provide faster accesses by a factor of 2. Furthermore, by optimizing the interface object definitions with JSON instead of XML will provide a significant boost in performance with a significant reduction in the required network bandwidth.

A tiered deployment based protocol has also been developed for CWS4DB system due to the the possibility of remote servers that may have limited internet bandwidth and/or high latency issues. This was facilitated by implementing RESTful web services that use an easy and user-friendly configuration files to provide a high quality of service to support the tiered deployment based on the remote server configurations, internet band width, user needs and system capabilities. The RESTful service architecture allowed for more the streamlined development of Orbiter Services, and Orbiter infrastructure versioning was adopted to promote a flexible stable system of releases.

The STAR resource Monitor module helps users track live status of STAR databases. Using Orbiter Federation RESTful web services, this module lists available resources and displays an interactive

Google map displaying detailed resource information. This module also provides real-time charts and tables presenting this information, allowing users to choose the best presentation in order to understand the states of available STAR NP database resources. Once this data is available via an Orbiter web service, this tool will be able to give users up-to-the-minute information on the performance of STAR resources. An example of this interactive map is shown in Figure 4

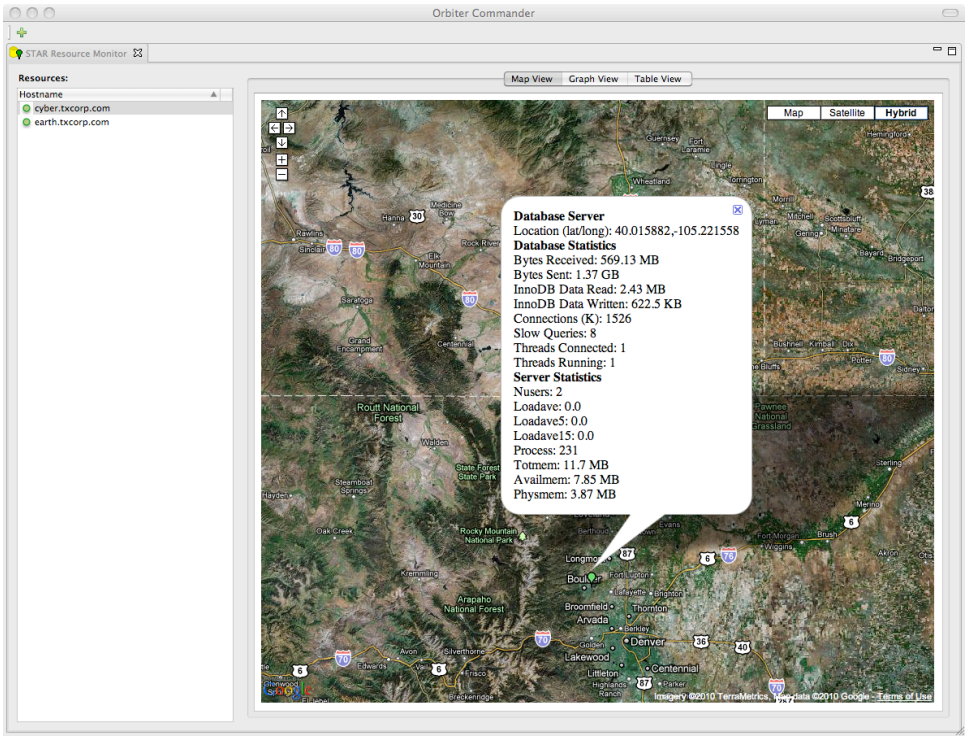


Figure 4: Real-Time STAR Resource Statistics

Commander also provides a STAR Simulator module for running queries against available NP database resources. Using this module users may run a STAR input file against any host, specifying the output format, whether or not to perform the query (or a no-op operation), use the query cache, or to use validation, allowing this service to be tested wherever it is deployed. Service round-trip timings can be displayed to benchmark the services between resources as well, providing a powerful tool for interacting with the STAR query service. Together these modules add value to the STAR web service development initiative. Using STAR resources Monitor, resources can be chosen from the monitor and used in the Simulator module, allowing users to systematically test the performance of the available STAR NP databases.

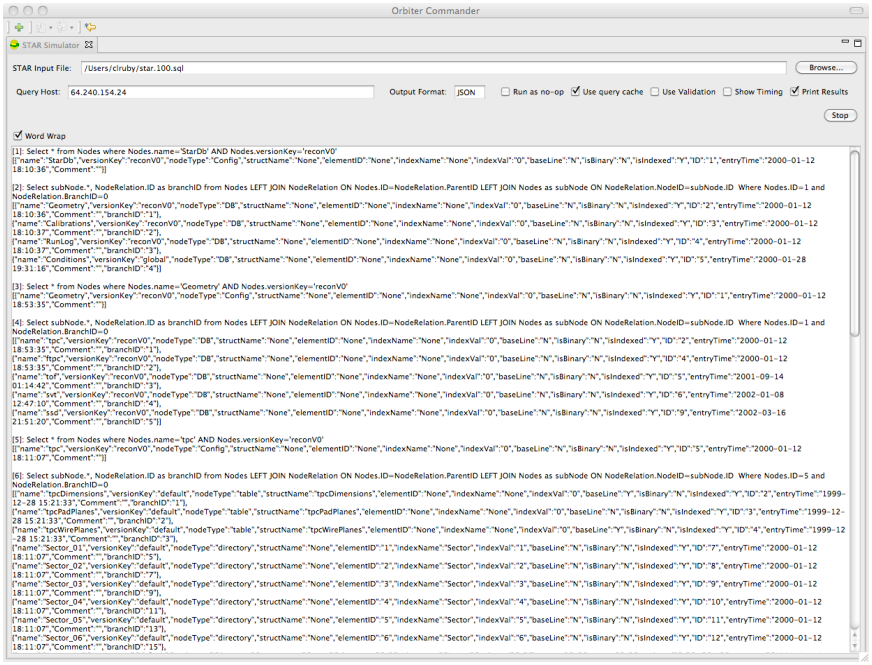


Figure 5: Orbiter Federation STAR Simulator

The Orbiter Commander Python Interpreter module allows users to interact with their command-line python installation on their local machines. By configuring Commander to use the path to an existing python installation, this module is capable of executing single or multi-line Python scripts entered via a command prompt, or can execute existing Python scripts that are loaded from an external file. The Commander Python Interpreter module can be used to execute a Python script that tests an Orbiter web service with a service URI specified as a command-line argument.

We have developed a developed a Query service to provide a two level auto-caching algorithm to increase the effective system performance. This was done by caching the query results on proxy servers at different levels of network and then making them available for other servers in the network. When a request comes in and if the query result was not found in cached files list on this requested server and is available in one of the proxy servers in the network, the cache file will be retrieved rather than making a call to the database server. The function of a proxy server that caches query result on the servers hard disk so that the result can be quickly retrieved by the same or a different user the next time that query is requested. The proxy cache eases bandwidth requirements and reduces delays that are inherent in a heavily trafficked, Internet-connected network.

The authentication and authorization for multiple Virtual Organizations (VOs) is essential for delivering a high quality of service with user-friendly management capabilities. We employed the standard GSI security, authentication, and authorization methods available within the Globus infrastructure for CWS4DB. Only an authenticated and authorized user can set vo (virtual organization name) and role (his role in the organization) for Multi-VO Role-Based caching. When a query request is made with vo and role, first the service will look for any cached files in the cache folder created for the specific Virtual Organization and the cache file will be retrieved rather than

making a call to the database server. An authenticated and authorized user has the ability to get, list, delete and recreate the Multi-VO Role-Based cache by using OrbiterCacheFileService and by passing virtual organization name and role.

Dynamic On-Demand Data Resource Access will provide a STAR MySQL database instance using the Virtual Workspaces infrastructure, Sun Grid Utility Computing resources for use with Grid deployments. This on-demand service will implement Virtual Workspace infrastructure (Nimbus) developed at Argonne National Laboratory. This database can then be integrated with CWS4DB data resource nodes as a data resource for providing additional load balancing capabilities. An authorized user can add a new db resource to the list by using the Query DB Connection String Service and the load balancer sets the ranking based on weighted score and make it available to use by the query service and other services. That way every time a new database resource node is added to the virtual work stations on demand it will be made available for all the services to use.

We have implemented 3 levels of fault resilient mechanism in our services. Our query service uses Load Balancer to get the top three high ranked active database resources available to make a connection. If the first resource fails to connect for some reason it will try connect second and third resources and keeps them as an array of available data resources for querying. When the database query fails at one of the resources it will try the other resources available in the database resources array. The service will trigger an error only if all the fault resilient mechanisms are failed. The number of database resources can be changed and can be set in the requesting query service to make it more fault resilient system when more db resources are available for use.

5 Conclusion

CWS4DB was created and developed with the intention of aiding in the management and analysis of massive, distributed and heterogeneous Neutron Physics databases created by the work done at the STAR experiment. Using the robust, secure and customizable web services offered by the Orbiter Federation SOA to access distributed relational databases and utilize the STAR C++ API which has been integrated with root4star, CWS4DB is able to allow NP and HEP users to focus on science and other critical tasks rather than on database maintenance. Orbiter and Orbiter Commander also provides a sophisticated desktop application while enabling offline computing capabilities and integration with other local applications. Future work for CWS4DB could include extending this service to different scientific and commercial applications which will take advantage of the generic bridge between distributed and heterogeneous relational databases and high-end users.

6 About Tech-X Corporation

At Tech-X, we address specific research questions in science and engineering by applying our expertise in high-performance computing, modeling and data analysis, physics, grid computing,

and HPC infrastructure (visualization, remote data access). Software solutions developed as part of our research efforts also support industries such as aerospace and semiconductor manufacturing.

The Systems Integration Group at Tech-X Corporation has a wealth of experience in tailoring integrated solutions to individual needs. Orbiter Virtualization has been in use for several years with tremendous success, and Orbiter Federation services, Pilot browser client and Commander desktop application are designed to be easily customized to fit your needs. We offer full application development to tailor Orbiter solutions to your specific needs.

Sean Burley (sburley@txcorp.com) is a software developer and an Orbiter developer at Tech-X Corporation.

Tech-X Corporation
5621 Arapahoe Ave. Suite A
Boulder, CO 80303

http://orbiter.txcorp.com
orbiter@txcorp.com

References

[1] "The star experiment at the relativistic heavy ion collider, brookhaven national lab, [online]." http://www.star.bnl.gov/, 2011.

[2] "The star experiment at the relativistic heavy ion collider, brookhave national lab: The physics of a star, [online]." http://www.star.bnl.gov/central/physics/, 2011.

[3] "A methodology for service architectures oasis draft." http://www.oasis-open.org/committees/download.php/15071, 2005.

[4] "Orbiter virtual file system service oriented architecure interfaces." https://orbiter.sns.gov.

[5] U. o. C. I. R.T. Fielding, Ph.D. Dissertation, "Architectural styles and design of network-based software architectures." http://www.ics.uci.edu/fielding/pubs/dissertation/top.htm, 2000.

[6] M. Green and S. Miller, "Multitier portal architecture for thin and thick-client neutron scattering experiment support." M. Pierce, Ed. Internatiopnal Workshop on Grid Computing Enviroments (GCE), November 2007.

[7] A. S. T. Winograd and N. U. D. o. C. S. P. .-. K. Scarfone, "Guide to secure web servies: Recommendations of the national institute of standards and technology." http://csrc.nist.gov/publications/nistpubs/800-95/SP800-95.pdf, Augest 2007.

[8] "Web services interoperabilty organization." http://www.ws-i.org/.

4 Documentation

4.1 RESTful Service Documentation

The following pages include the full service documentation for every RESTful web service developed for this effort.

OrbiterConnectivityService

Description

This is the Orbiter Connectivity service class.

Brief example of use:

```
Create an instance of OrbiterConnectivityService
$objService = new OrbiterConnectivityService;

-----------------------------------------------------------------------------------------

Service name
OrbiterConnectivityService.php

-----------------------------------------------------------------------------------------

Service Operation Definitions

Summary Format
operation:
description:
required attributes:
optional attributes:
service method:

-----------------------------------------------------------------------------------------

operation:'api'
description: 'List the service API.'
required attributes: ''
optional attributes: 'format'
service method: 'GET'

operation:'noop'
description: 'Perform a NOOP service request to determine Orbiter connectivity'
             with the service provider.'
required attributes: ''
optional attributes: ''
service method: 'GET'

-----------------------------------------------------------------------------------------

Service Attribute Definitions

Summary Format
attribute:
description:
defaultValue:
allowableValues:

-----------------------------------------------------------------------------------------

attribute: 'operation'
description: 'Specify the type of operation service request required.'
defaultValue: ''
allowableValues: 'api', 'noop'

attribute: 'html'
description: 'Generate HTML compliant output byt replacing escaped character and use
              appropriate tagging and line returns.'
```

```
defaultValue: 'off'
allowableValues: 'on', 'off'

attribute: 'format'
description: 'Specify the service response format.'
defaultValue: 'json'
allowableValues: 'json', 'api', 'schema'


View the service schema at ServiceSchema

View the service wsdl 2.0 at ServiceWSDL
```

- **author:** Mark L. Green <mlgreen@txcorp.com>
- **version:** Release: @package_version@
- **copyright:** 2006-2010 Tech-X Corporation. All rights reserved.
- **link:** Tech-X Corporation
- **license:** BSD License

*Located in **/OrbiterFederation/classes/OrbiterConnectivityService.class.php** (line **132**)*

```
OrbiterService
   |
   --OrbiterConnectivityService
```

Method Summary

Description | Vars | *Methods* (details)

- *OrbiterConnectivityService* **__construct** ()
- *void* **__destruct** ()
- *void* **processAttributes** ()
- *void* **processOperations** ()
- *void* **processServiceRequest** ()

Variables

Description | Vars (*details*) | Methods (details)

Inherited Variables

Inherited from OrbiterService

- **OrbiterService::$_auth**
- **OrbiterService::$_cnt**
- **OrbiterService::$_contents**
- **OrbiterService::$_count**
- **OrbiterService::$_dbMaster**
- **OrbiterService::$_dbSlave**
- **OrbiterService::$_definedAttr**
- **OrbiterService::$_errorHandler**
- **OrbiterService::$_logger**
- **OrbiterService::$_phpVariables**
- **OrbiterService::$_queryString**
- **OrbiterService::$_requestMethod**
- **OrbiterService::$_requestUri**
- **OrbiterService::$_scriptName**
- **OrbiterService::$_serviceAddress**

- OrbiterService::$_serviceSchema
- OrbiterService::$_userId
- OrbiterService::$_userRole

Methods

Description | Vars Methods (*details*)

Constructor __construct (line 136)

Constructs a new OrbiterConnectivityService.class instance

OrbiterConnectivityService **__construct** ()

Redefinition of:

OrbiterService::__construct()
Constructs a new OrbiterService abstract class instance

Destructor __destruct (line 153)

Destructs the OrbiterConnectivityService.class instance

void **__destruct** ()

Redefinition of:

OrbiterService::__destruct()
Destructs a OrbiterService abstract class instance

processAttributes (line 159)

Process the service attributes

- **access:** protected

void **processAttributes** ()

Redefinition of:

OrbiterService::processAttributes()
Abstract function for setting the default attributes the service

processOperations (line 168)

Process the service operations

- **access:** protected

void **processOperations** ()

Redefinition of:

OrbiterService::processOperations()
Abstract function for setting the operations for the service

processServiceRequest (line 244)

```
-------------------------------------------------------------------------------------------
Usage: No required attributes

operation: 'api'
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterConnectivityService.php/operation/api

Response: Returns the api of the service in 'json' format

[{"service":"{Service Name}","attributes":{"operation":{"default":"","restrictions":
[List of Operations]},..List of attributes}, "operations":{"0":[],"operation1":{"GET":
{"description":"{Operation Description}","restrictions":{"required":[List of required attributes],
"optional":[List of optional attributes]}}},..List of Operations}}]


Usage: with optional attribute 'format'

operation: 'api'
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterConnectivityService.php/operation/api/format/api

Response: Returns the api of the service in 'api' format as

array(
     'service'=>{ServiceName},
     'attributes'=>array(
         'operation'=>array(
             'default'=>{Default Value},
             'restrictions'=>array(
                List of Allowable Values
             )
          ),.. List of attributes
     ),
     'operations'=>array(
         'operation1'=>array(
             'method type'=>array(
                 'description'=>{operation description}
                 'restrictions'=>array(
                     'required'=>array(
                         List of required attributes,
                     )
                     'optional'=>array(
                         List of optional attributes
                     )
                  )
              )
         ), .. List of operations
     )
)

-------------------------------------------------------------------------------------------

Usage: No required attributes

operation: 'noop'
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterConnectivityService.php/operation/noop

Response: Returns the response time of the service without any operation in seconds

-------------------------------------------------------------------------------------------
```

- **access:** protected

void **processServiceRequest** ()

Redefinition of:

OrbiterService::processServiceRequest()

Abstract function for the business logic for the service request

Inherited Methods

Inherited From OrbiterService

- **OrbiterService::__construct()**
- **OrbiterService::authenticateServiceRequest()**
- **OrbiterService::checkAttributeValidity()**
- **OrbiterService::checkInterfaceRequest()**
- **OrbiterService::checkOperationValidity()**
- **OrbiterService::defineServiceAttributes()**
- **OrbiterService::defineServiceOperations()**
- **OrbiterService::doServiceRequest()**
- **OrbiterService::endLogServiceRequest()**
- **OrbiterService::getServiceAddress()**
- **OrbiterService::logServiceRequest()**
- **OrbiterService::processAttributes()**
- **OrbiterService::processOperations()**
- **OrbiterService::processServiceAttributes()**
- **OrbiterService::processServiceRequest()**
- **OrbiterService::processServiceResponse()**
- **OrbiterService::setDatabaseConnections()**
- **OrbiterService::setServiceAddress()**
- **OrbiterService::triggerError()**
- **OrbiterService::__destruct()**

OrbiterErrorHandlerMessageService

Description

Description | Vars | Methods (details)

This is an error handler message service class which adds a class or an error message, lists an error message,lists all the error messages of a class lists message template and delete an error message.It also lists the service API.

Brief example of use:

```
// Create an instance of OrbiterErrorHandlerMessageService
$objService = new OrbiterErrorHandlerMessageService();

---------------------------------------------------------------------------------------------

Service name
OrbiterErrorHandlerMessageService.php

---------------------------------------------------------------------------------------------

Service Operation Definitions

Summary Format
operation:
description:
required attributes:
optional attributes:
service method:

---------------------------------------------------------------------------------------------

operation:'api'
description: 'List the service API.'
required attributes: ''
optional attributes: 'format'
service method: 'GET'

operation:'template'
description: 'List message template.'
required attributes: ''
optional attributes: 'format'
service method: 'GET'

operation:'deleteError'
description: 'Delete an error message.'
required attributes: 'errorNumber'
optional attributes: 'format'
service method: 'GET'

operation:'listErrors'
description: 'List error messages of the class'
required attributes: 'className'
optional attributes: 'format'
service method: 'GET'

operation:'listError'
description: 'List error message'
required attributes: 'errorNumber'
optional attributes: 'format'
service method: 'GET'

operation:'addClass'
description: 'Add a class'
required attributes: 'className'
optional attributes: 'format'
service method: 'GET'

operation:'addError'
description: 'Add an error message'
required attributes: 'className', 'name', 'message'
optional attributes: 'format'
service method: 'GET'
```

```
operation:'updateClass'
description: 'Update a Class'
required attributes: 'className', 'newClassName'
optional attributes: 'format'
service method: 'GET'

------------------------------------------------------------------------------------------

Service Attribute Definitions

Summary Format
attribute:
description:
defaultValue:
allowableValues:

------------------------------------------------------------------------------------------

attribute: 'operation'
description: 'Specify the type of operation service request required.'
defaultValue: ''
allowableValues: 'api', 'template', 'deleteError', 'listErrors', 'listError', 'addClass', 'addError', 'update

attribute: 'className'
description: 'Specify the class name.'
defaultValue: ''
allowableValues: ''

attribute: 'newClassName'
description: 'Specify the new class name.'
defaultValue: ''
allowableValues: ''

attribute: 'name'
description: 'Specify the name.'
defaultValue: ''
allowableValues: ''

attribute: 'message'
description: 'Specify the message.'
defaultValue: ''
allowableValues: ''

attribute: 'errorNumber'
description: 'Specify the error number.'
defaultValue: ''
allowableValues: ''

attribute: 'format'
description: 'Specify the service response format.'
defaultValue: 'text'
allowableValues: 'json', 'api', 'xml', 'text'


View the service schema at ServiceSchema

View the service wsdl 2.0 at ServiceWSDL
```

- **author:** Mark L. Green <mlgreen@txcorp.com>
- **version:** Release: @package_version@
- **copyright:** 2006-2010 Tech-X Corporation. All rights reserved.
- **link:** http://www.txcorp.com/
- **license:** BSD License

*Located in **/OrbiterFederation/classes/OrbiterErrorHandlerMessageService.class.php** (line **189**)*

```
OrbiterService
   |
   --OrbiterErrorHandlerMessageService
```

Method Summary

Description | Vars | *Methods* (details)

OrbiterErrorHandlerMessageService **__construct** ()

void **__destruct** ()

- *void* **processAttributes** ()
- *void* **processOperations** ()
- *void* **processServiceRequest** ()

Variables

Description | Vars (*details*) | Methods (details)

Inherited Variables

Inherited from OrbiterService

- **OrbiterService::$_auth**
- **OrbiterService::$_cnt**
- **OrbiterService::$_contents**
- **OrbiterService::$_count**
- **OrbiterService::$_dbMaster**
- **OrbiterService::$_dbSlave**
- **OrbiterService::$_definedAttr**
- **OrbiterService::$_errorHandler**
- **OrbiterService::$_logger**
- **OrbiterService::$_phpVariables**
- **OrbiterService::$_queryString**
- **OrbiterService::$_requestMethod**
- **OrbiterService::$_requestUri**
- **OrbiterService::$_scriptName**
- **OrbiterService::$_serviceAddress**
- **OrbiterService::$_serviceSchema**
- **OrbiterService::$_userId**
- **OrbiterService::$_userRole**

Methods

Description | Vars Methods (*details*)

Constructor __construct (line 194)

Constructs a new OrbiterErrorHandlerMessageService class instance

OrbiterErrorHandlerMessageService **__construct** ()

Redefinition of:

OrbiterService::__construct()
Constructs a new OrbiterService abstract class instance

Destructor __destruct (line 213)

Destructs a OrbiterErrorHandlerMessageService class instance

void **__destruct** ()

Redefinition of:

OrbiterService::__destruct()
Destructs a OrbiterService abstract class instance

processAttributes (line 220)

Process the service attributes

- **access:** protected

void **processAttributes** ()

Redefinition of:

OrbiterService::processAttributes()
Abstract function for setting the default attributes the service

processOperations (line 232)

Process the service operations

- **access:** protected

void **processOperations** ()

Redefinition of:

OrbiterService::processOperations()
Abstract function for setting the operations for the service

processServiceRequest (line 494)

Process the service request based on the operation the user specified

- **exception:** E_USER_ERROR Orbiter Error 100:1017 E_USER_ERROR Database query error
- **exception:** E_USER_ERROR Orbiter Error 100:1043 E_USER_ERROR Database query error
- **exception:** E_USER_ERROR Orbiter Error 100:1002 E_USER_ERROR Error finding last inserted error message
- **exception:** E_USER_ERROR Orbiter Error 100:1044 E_USER_ERROR Database query error
- **exception:** E_USER_ERROR Orbiter Error 100:1042 E_USER_ERROR Database query error
- **exception:** E_USER_ERROR Orbiter Error 100:1041 E_USER_ERROR Database query error
- **exception:** E_USER_ERROR Orbiter Error 100:1046 E_USER_ERROR Database query error
- **exception:** E_USER_ERROR Orbiter Error 100:1047 E_USER_ERROR Database query error
- **exception:** E_USER_ERROR Orbiter Error 100:1048 E_USER_ERROR Database query error
- **exception:** E_USER_ERROR Orbiter Error 100:1045 E_USER_ERROR Database query error
- **exception:** E_USER_ERROR Orbiter Error 100:1198 E_USER_ERROR The error message must be defined in the form of {class ID:error ID}
- **exception:** E_USER_ERROR Orbiter Error 100:1826 E_USER_ERROR The specified error name is invalid
- **exception:** E_USER_ERROR Orbiter Error 100:1827 E_USER_ERROR The specified class name must be added before adding error
- **exception:** E_USER_ERROR Orbiter Error 100:1871 E_USER_ERROR Database query error
- **exception:** E_USER_ERROR Orbiter Error 100:1872 E_USER_ERROR Database query error
- **exception:** E_USER_ERROR Orbiter Error 100:1824 E_USER_ERROR Error finding last inserted class name
- **exception:** E_USER_ERROR Orbiter Error 100:1822 E_USER_ERROR The specified error number is invalid
- **exception:** E_USER_ERROR Orbiter Error 100:1200 E_USER_ERROR Database query error
- **exception:** E_USER_ERROR Orbiter Error 100:1199 E_USER_ERROR The error message must be defined in the form of {class ID:error ID}
- **exception:** E_USER_ERROR Orbiter Error 100:1820 E_USER_ERROR The specified class name is invalid
- **exception:** E_USER_ERROR Orbiter Error 100:1874 E_USER_ERROR The specified className doesn't exist

void **processServiceRequest** ()

Redefinition of:

OrbiterService::processServiceRequest()
Abstract function for the business logic for the service request

Inherited Methods

Inherited From OrbiterService

- **OrbiterService::__construct()**
- **OrbiterService::authenticateServiceRequest()**
- **OrbiterService::checkAttributeValidity()**
- **OrbiterService::checkInterfaceRequest()**
- **OrbiterService::checkOperationValidity()**
- **OrbiterService::defineServiceAttributes()**
- **OrbiterService::defineServiceOperations()**
- **OrbiterService::doServiceRequest()**
- **OrbiterService::endLogServiceRequest()**
- **OrbiterService::getServiceAddress()**
- **OrbiterService::logServiceRequest()**

- **OrbiterService::processAttributes()**
- **OrbiterService::processOperations()**
- **OrbiterService::processServiceAttributes()**
- **OrbiterService::processServiceRequest()**
- **OrbiterService::processServiceResponse()**
- **OrbiterService::setDatabaseConnections()**
- **OrbiterService::setServiceAddress()**
- **OrbiterService::triggerError()**
- **OrbiterService::__destruct()**

OrbiterFederationExplorerService

Description

Description | Vars (details) | Methods (details)

This is the Orbiter Federation Explorer service class.

```
Brief example of use:

Create an instance of OrbiterFederationExplorerService
$objService = new OrbiterFederationExplorerService;

-------------------------------------------------------------------------------------------

Service name
OrbiterFederationExplorerService.php

-------------------------------------------------------------------------------------------

Service Operation Definitions

Summary Format
operation:
description:
required attributes:
optional attributes:
service method:

-------------------------------------------------------------------------------------------

operation:'api'
description: 'List the service API.'
required attributes: ''
optional attributes: 'format'
service method: 'GET'

operation:'baseStations'
description: 'List the Orbiter Federation SOA baseStations.'
required attributes: ''
optional attributes: 'list'
service method: 'GET'

operation: 'updateServices'
description: 'Update the Orbiter Federation SOA services from the file system.'
required attributes: ''
optional attributes: ''
service method: 'GET'

-------------------------------------------------------------------------------------------

Service Attribute Definitions

Summary Format
attribute:
description:
defaultValue:
allowableValues:

-------------------------------------------------------------------------------------------

attribute: 'operation'
description: 'Specify the type of operation service request required.'
defaultValue: ''
```

```
allowableValues: 'api', 'baseStations', 'updateServices'

attribute: 'html'
description: 'Generate HTML compliant output byt replacing escaped character and use
              appropriate tagging and line returns.'
defaultValue: 'off'
allowableValues: 'on', 'off'

attribute: 'list'
description: 'List the Orbiter packages and Services along with Orbiter baseStations.'
defaultValue: ''
allowableValues: 'services'

attribute: 'format'
description: 'Specify the service response format.'
defaultValue: 'json'
allowableValues: 'json', 'api', 'schema'


View the service schema at ServiceSchema

View the service wsdl 2.0 at ServiceWSDL
```

- **author:** Mark L. Green <mlgreen@txcorp.com>
- **version:** Release: @package_version@
- **copyright:** 2006-2010 Tech-X Corporation. All rights reserved.
- **link:** Tech-X Corporation
- **license:** BSD License

*Located in **/OrbiterFederation/classes/OrbiterFederationExplorerService.class.php** (line **144**)*

```
OrbiterService
   |
   --OrbiterFederationExplorerService
```

Variable Summary

Description | *Vars* (details) | Methods (details)

array **$_serviceArray**

Method Summary

Description | Vars (details) | *Methods* (details)

OrbiterFederationExplorerService **__construct** ()
void **__destruct** ()
void **flushDatabase** ()
void **processAttributes** ()
void **processOperations** ()
void **processServiceRequest** ()
void **updateDatabase** ()
void **updateServices** (*string* **$directory**)

Variables

Description | Vars (*details*) | Methods (details)

array **$_serviceArray** (line 156)

- **var:** package/service names.

- **access:** protected

Inherited Variables

Inherited from OrbiterService

- **OrbiterService::$_auth**
- **OrbiterService::$_cnt**
- **OrbiterService::$_contents**
- **OrbiterService::$_count**
- **OrbiterService::$_dbMaster**
- **OrbiterService::$_dbSlave**
- **OrbiterService::$_definedAttr**
- **OrbiterService::$_errorHandler**
- **OrbiterService::$_logger**
- **OrbiterService::$_phpVariables**
- **OrbiterService::$_queryString**
- **OrbiterService::$_requestMethod**
- **OrbiterService::$_requestUri**
- **OrbiterService::$_scriptName**
- **OrbiterService::$_serviceAddress**
- **OrbiterService::$_serviceSchema**
- **OrbiterService::$_userId**
- **OrbiterService::$_userRole**

Methods

Description | Vars (details) Methods (*details*)

Constructor __construct (line 160)

Constructs a new OrbiterFederationExplorerService.class instance

OrbiterFederationExplorerService **__construct** ()

Redefinition of:

OrbiterService::__construct()
Constructs a new OrbiterService abstract class instance

Destructor __destruct (line 177)

Destructs a OrbiterFederationExplorerService.class instance

void **__destruct** ()

Redefinition of:

OrbiterService::__destruct()
Destructs a OrbiterService abstract class instance

flushDatabase (line 534)

Flush the service database tables.The database is first flushed before updating the service database tables

- **exception:** E_USER_ERROR Orbiter Error 161:1914 E_USER_ERROR Database query error
- **exception:** E_USER_ERROR Orbiter Error 161:1916 E_USER_ERROR Orbiter service access must be defined
- **exception:** E_USER_ERROR Orbiter Error 161:1917 E_USER_ERROR Database query error
- **exception:** E_USER_ERROR Orbiter Error 161:1918 E_USER_ERROR Orbiter version must be defined
- **exception:** E_USER_ERROR Orbiter Error 161:1919 E_USER_ERROR Database query error

void **flushDatabase** ()

processAttributes (line 183)

Process the service attributes

- **access:** protected

void **processAttributes** ()

Redefinition of:

OrbiterService::processAttributes()
Abstract function for setting the default attributes the service

processOperations (line 192)

Process the service operations

- **access:** protected

void **processOperations** ()

Redefinition of:

OrbiterService::processOperations()
Abstract function for setting the operations for the service

processServiceRequest (line 329)

Process the service request based on the operation the user specified

```
 ---------------------------------------------------------------------------------------

 Usage: No required attributes

 operation: 'api'
 URI:/orbiter/ORBITERVERSION/service/webservice/
 OrbiterFederationExplorerService.php/operation/api

 Response: Returns the api of the service in 'json' format

 [{"service":"{Service Name}","attributes":{"operation":{"default":"","restrictions":
 [List of Operations]},..List of attributes}, "operations":{"0":[],"operation1":{"GET":
 {"description":"{Operation Description}","restrictions":{"required":[List of required attributes],
 "optional":[List of optional attributes]}}},..List of Operations}}]


 Usage: with optional attribute 'format'

 operation: 'api'
 URI:/orbiter/ORBITERVERSION/service/webservice/
 OrbiterFederationExplorerService.php/operation/api/format/api

 Response: Returns the api of the service in 'api' format as
```

```
array(
     'service'=>{ServiceName},
     'attributes'=>array(
         'operation'=>array(
             'default'=>{Default Value},
             'restrictions'=>array(
                List of Allowable Values
             )
          ),.. List of attributes
     ),
     'operations'=>array(
         'operation1'=>array(
             'method type'=>array(
                 'description'=>{operation description}
                 'restrictions'=>array(
                     'required'=>array(
                         List of required attributes,
                     )
                     'optional'=>array(
                         List of optional attributes
                     )
                  )
              )
         ), .. List of operations
     )
)
```

Usage: No required attributes

operation: 'baseStations'
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterFederationExplorerService.php/operation/baseStations

Response: Lists the basestations of the Orbiter Federation SOA in the format as

{"bid":"1","name":"Oak Ridge National Laboratory Spallation Neutron Source ","FQHN":
"orbit.sns.gov", "database":"MASTER","database_id":"1","use_ssl":"1",
"stamp":"2010-09-22 09:41:40"}

where
'bid' denotes basestation id
'name' denotes basestation name
'FQHN' denotes Fully Qualified Host Name of the service provider
'database' denotes if it is master or slave
'database_id' denotes the database id
'use_ssl' denotes if ssl should be used to communicate with the basestation
'stamp' denotes the timestamp including date and time

Usage: with optional attribute 'format'

operation: 'baseStations'
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterFederationExplorerService.php/operation/baseStations/list/services

Response: Lists the basestations of the Orbiter Federation SOA in the format as

{"bid":"2","package":"webservice","service":"OrbiterPolicyManagerService.php",
"version":"trunk","use_ssl":"1"}

where
'bid' denotes basestation id
'package' denotes web service package
'service' denotes the name of the service
'version' denotes the version of the service

```
'use_ssl' denotes if ssl should be used to communicate with the basestation

---------------------------------------------------------------------------------------------

Usage: No required attributes

operation: 'updateServices'
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterFederationExplorerService.php/operation/updateServices

Response: Updates the Orbiter Federation SOA services in the database from the file system


Usage: with optional attribute 'format'

operation: 'updateServices'
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterFederationExplorerService.php/operation/updateServices/format/json

Response: Updates the Orbiter Federation SOA services in the database from the file system

---------------------------------------------------------------------------------------------
```

- **exception:** E_USER_ERROR Orbiter Error 161:1656 E_USER_ERROR Database query error
- **exception:** E_USER_ERROR Orbiter Error 161:1909 E_USER_ERROR Not a valid directory
- **exception:** E_USER_ERROR Orbiter Error 161:1657 E_USER_ERROR Database query error
- **exception:** E_USER_ERROR Orbiter Error 161:1908 E_USER_ERROR Orbiter service directory location must be defined
- **access:** protected

void **processServiceRequest** ()

Redefinition of:

OrbiterService::processServiceRequest()
Abstract function for the business logic for the service request

updateDatabase (line 444)

Updates the service database tables

- **exception:** E_USER_ERROR Orbiter Error 161:1911 E_USER_ERROR Database query error
- **exception:** E_USER_ERROR Orbiter Error 161:1912 E_USER_ERROR Database query error
- **exception:** E_USER_ERROR Orbiter Error 161:1913 E_USER_ERROR Database query error

void **updateDatabase** ()

updateServices (line 395)

Updates the services in the database after reading from the file system

- **exception:** E_USER_ERROR Orbiter Error 161:1910 E_USER_ERROR Directory is not readable

void **updateServices** (*string* **$directory**)

- *string* **$directory**: absolute services directory path

Inherited Methods

Inherited From OrbiterService

OrbiterService::__construct()

- OrbiterService::authenticateServiceRequest()
- OrbiterService::checkAttributeValidity()
- OrbiterService::checkInterfaceRequest()
- OrbiterService::checkOperationValidity()
- OrbiterService::defineServiceAttributes()
- OrbiterService::defineServiceOperations()
- OrbiterService::doServiceRequest()
- OrbiterService::endLogServiceRequest()
- OrbiterService::getServiceAddress()
- OrbiterService::logServiceRequest()
- OrbiterService::processAttributes()
- OrbiterService::processOperations()
- OrbiterService::processServiceAttributes()
- OrbiterService::processServiceRequest()
- OrbiterService::processServiceResponse()
- OrbiterService::setDatabaseConnections()
- OrbiterService::setServiceAddress()
- OrbiterService::triggerError()
- OrbiterService::__destruct()

Documentation generated on Thu, 01 Dec 2011 13:50:27 -0500 by phpDocumentor 1.4.3

OrbiterMasterSlaveDatabaseValidationService

Description

This class compares the Orbiter Master database tables with the Slave database tables.

Brief example of use:

```
// Create an instance of OrbiterMasterSlaveDatabaseValidationService
$objService = new OrbiterMasterSlaveDatabaseValidationService();

---------------------------------------------------------------------------------------------

Service name
OrbiterMasterSlaveDatabaseValidationService.php

---------------------------------------------------------------------------------------------

Service Operation Definitions

Summary Format
operation:
description:
required attributes:
optional attributes:
service method:

---------------------------------------------------------------------------------------------

operation:'api'
description: 'List the service API.'
required attributes: ''
optional attributes: 'format'
service method: 'GET'

operation:'replaceTable'
description: 'Replace table.'
required attributes: 'table', 'key'
optional attributes: 'repair', 'checkError', 'format'
service method: 'GET'

operation:'validateTable'
description: 'Validate table.'
required attributes: 'table', 'key'
optional attributes: 'repair', 'checkError', 'format'
service method: 'GET'

---------------------------------------------------------------------------------------------

Service Attribute Definitions

Summary Format
attribute:
description:
defaultValue:
allowableValues:

---------------------------------------------------------------------------------------------

attribute: 'operation'
description: 'Specify the type of operation service request required.'
defaultValue: ''
allowableValues: 'api', 'replaceTable', 'validateTable'
```

```
attribute: 'table'
description: 'table name that needs to be replaced/repaired.'
defaultValue: ''
allowableValues: ''

attribute: 'key'
description: 'table column key ID.'
defaultValue: ''
allowableValues: ''

attribute: 'repair'
description: 'when true existing table will be droped and '_new' table will replace the droped table.'
defaultValue: 'false'
allowableValues: 'true', 'false'

attribute: 'checkError'
description: 'when true checks for the query errors when  table is replaced/repaired.'
defaultValue: 'false'
allowableValues: 'true', 'false'

attribute: 'format'
description: 'Specify the service response format.'
defaultValue: 'json'
allowableValues: 'json', 'api', 'xml', 'text'


View the service schema at ServiceSchema

View the service wsdl 2.0 at ServiceWSDL
```

- **author:** Mark L. Green <mlgreen@txcorp.com>
- **version:** Release: @package_version@
- **copyright:** 2006-2010 Tech-X Corporation. All rights reserved.
- **link:** http://www.txcorp.com/
- **license:** BSD License

*Located in **/OrbiterFederation/classes/OrbiterMasterSlaveDatabaseValidationService.class.php** (line **153**)*

```
OrbiterService
   |
   --OrbiterMasterSlaveDatabaseValidationService
```

Method Summary

Description | Vars | *Methods* (details)

- *OrbiterMasterSlaveDatabaseValidationService* **__construct** ()
- *void* **__destruct** ()
- *void* **processAttributes** ()
- *void* **processOperations** ()
- *void* **processServiceRequest** ()

Variables

Description | Vars (*details*) | Methods (details)

Inherited Variables

Inherited from OrbiterService

- **OrbiterService::$_auth**
- **OrbiterService::$_cnt**
- **OrbiterService::$_contents**

- **OrbiterService::$_count**
- **OrbiterService::$_dbMaster**
- **OrbiterService::$_dbSlave**
- **OrbiterService::$_definedAttr**
- **OrbiterService::$_errorHandler**
- **OrbiterService::$_logger**
- **OrbiterService::$_phpVariables**
- **OrbiterService::$_queryString**
- **OrbiterService::$_requestMethod**
- **OrbiterService::$_requestUri**
- **OrbiterService::$_scriptName**
- **OrbiterService::$_serviceAddress**
- **OrbiterService::$_serviceSchema**
- **OrbiterService::$_userId**
- **OrbiterService::$_userRole**

Methods

Description | Vars Methods (*details*)

Constructor __construct (line 158)

Constructs a new OrbiterMasterSlaveDatabaseValidationService class instance

OrbiterMasterSlaveDatabaseValidationService **__construct** ()

Redefinition of:

OrbiterService::__construct()
Constructs a new OrbiterService abstract class instance

Destructor __destruct (line 177)

Destructs a OrbiterMasterSlaveDatabaseValidationService class instance

void **__destruct** ()

Redefinition of:

OrbiterService::__destruct()
Destructs a OrbiterService abstract class instance

processAttributes (line 184)

Process the service attributes

- **access:** protected

void **processAttributes** ()

Redefinition of:

OrbiterService::processAttributes()
Abstract function for setting the default attributes the service

processOperations (line 196)

Process the service operations

- **access:** protected

void **processOperations** ()

Redefinition of:

OrbiterService::processOperations()
Abstract function for setting the operations for the service

processServiceRequest (line 321)

Processes the service request based on the operation the user specified

- **exception:** E_USER_ERROR Orbiter Error 124:1249 E_USER_ERROR Cannot stop slave database replication
- **exception:** E_USER_ERROR Orbiter Error 124:1250 E_USER_ERROR Cannot lock master database table
- **exception:** E_USER_ERROR Orbiter Error 124:1209 E_USER_ERROR Database query error
- **exception:** E_USER_ERROR Orbiter Error 124:1208 E_USER_ERROR Database query error
- **exception:** E_USER_ERROR Orbiter Error 124:1207 E_USER_ERROR Database query error
- **exception:** E_USER_ERROR Orbiter Error 124:1251 E_USER_ERROR Cannot select from master database table
- **exception:** E_USER_ERROR Orbiter Error 124:1252 E_USER_ERROR Cannot unlock master database table
- **exception:** E_USER_ERROR Orbiter Error 124:1256 E_USER_ERROR Cannot rename _new slave database table
- **exception:** E_USER_ERROR Orbiter Error 124:1255 E_USER_ERROR Cannot drop slave database table
- **exception:** E_USER_ERROR Orbiter Error 124:1254 E_USER_ERROR Cannot insert record into _new slave database table
- **exception:** E_USER_ERROR Orbiter Error 124:1253 E_USER_ERROR Cannot create _new slave database table
- **exception:** E_USER_ERROR Orbiter Error 124:1257 E_USER_ERROR Cannot start slave database replication

void **processServiceRequest** ()

Redefinition of:

OrbiterService::processServiceRequest()
Abstract function for the business logic for the service request

Inherited Methods

Inherited From OrbiterService

- **OrbiterService::__construct()**
- **OrbiterService::authenticateServiceRequest()**
- **OrbiterService::checkAttributeValidity()**
- **OrbiterService::checkInterfaceRequest()**
- **OrbiterService::checkOperationValidity()**
- **OrbiterService::defineServiceAttributes()**
- **OrbiterService::defineServiceOperations()**
- **OrbiterService::doServiceRequest()**
- **OrbiterService::endLogServiceRequest()**
- **OrbiterService::getServiceAddress()**
- **OrbiterService::logServiceRequest()**
- **OrbiterService::processAttributes()**
- **OrbiterService::processOperations()**
- **OrbiterService::processServiceAttributes()**
- **OrbiterService::processServiceRequest()**
- **OrbiterService::processServiceResponse()**
- **OrbiterService::setDatabaseConnections()**
- **OrbiterService::setServiceAddress()**
- **OrbiterService::triggerError()**
- **OrbiterService::__destruct()**

OrbiterNoopService

Description

Description | Methods (details)

This is authenticated RESTful service class for testing.

Brief example of use:

```
// Create an instance of OrbiterNoopService
$objService = new OrbiterNoopService();

// Process the service request
$objService->processRequest();
```

- **author:** Krishna R. Kantam <krishna@txcorp.com>
- **author:** Mark L. Green <mlgreen@txcorp.com>
- **version:** Release: @package_version@
- **copyright:** 2006-2010 Tech-X Corporation. All rights reserved.
- **link:** http://www.txcorp.com/
- **license:** BSD License

*Located in **/OrbiterFederation/classes/OrbiterNoopService.class.php** (line **75**)*

Method Summary

Description | *Methods* (details)

- *OrbiterNoopService* **__construct** ()
- *void* **authenticateServiceRequest** ()
- *void* **defineServiceAttributes** ()
- *void* **endLogServiceRequest** ()
- *void* **logServiceRequest** ()
- *void* **processRequest** ()
- *void* **processServiceAttributes** ()
- *void* **processServiceRequest** ()
- *void* **processServiceResponse** ()
- *void* **setDatabaseConnections** ()

Methods

Description | Methods (*details*)

Constructor __construct (line 125)

Constructs a new OrbiterDasLogPushService.class instance

OrbiterNoopService **__construct** ()

authenticateServiceRequest (line 174)

This authenticateServiceRequest function authenticates the service request and returns the user id.

void **authenticateServiceRequest** ()

defineServiceAttributes (line 197)

Defines service attributes default values.

void **defineServiceAttributes** ()

endLogServiceRequest (line 287)

Ends logging the service request.

void **endLogServiceRequest** ()

logServiceRequest (line 189)

Logs the service request.

void **logServiceRequest** ()

processRequest (line 135)

Processes the web service request

void **processRequest** ()

processServiceAttributes (line 212)

Get the service attributes from the authenticated service request and process the service attributes.

void **processServiceAttributes** ()

processServiceRequest (line 231)

Process the service request.

void **processServiceRequest** ()

processServiceResponse (line 250)

Output service response.

- **exception:** E_USER_ERROR Orbiter Error 101:1009 E_USER_ERROR 'Service Attribute Format' unknown format request

void **processServiceResponse** ()

setDatabaseConnections (line 160)

Initializes database conections.

void **setDatabaseConnections** ()

OrbiterQueryDbConnectionStringService

Description

Description | Vars | Methods (details)

This service provides the user an ability to add, update, and delete the database connection string information; so that the user can manage database resources on demand.

Brief example of use:

```
Create an instance of OrbiterQueryDbConnectionStringService
$objService = new OrbiterQueryDbConnectionStringService();

-----------------------------------------------------------------------------------------

Service name
OrbiterQueryDbConnectionStringService.php

-----------------------------------------------------------------------------------------

Service Operation Definitions

Summary Format
operation:
description:
required attributes:
optional attributes:
service method:

-----------------------------------------------------------------------------------------

operation:'api'
description: 'List the service API.'
required attributes: ''
optional attributes: 'format'
service method: 'GET'

operation:'insert'
description: 'Insert the connection string'
required attributes: 'conStr', 'type'
optional attributes: 'rank', 'hostType'
service method: 'GET'

operation:'update'
description: 'Update the connection string'
required attributes: 'conStr', 'type'
optional attributes: 'rank', 'status', 'hostType'
service method: 'GET'

operation:'delete'
description: 'Delete the connection string'
required attributes: 'conStr'
optional attributes: ''
service method: 'GET'

-----------------------------------------------------------------------------------------

Service Attribute Definitions

Summary Format
attribute:
description:
defaultValue:
allowableValues:

-----------------------------------------------------------------------------------------

attribute: 'operation'
description: 'Specify the type of operation service request required.'
defaultValue: ''
allowableValues: 'api', 'insert', 'update', 'delete'

attribute: 'html'
description: 'Generate HTML compliant output by replacing escaped character and use
              appropriate tagging and line returns.'
defaultValue: 'off'
allowableValues: 'on', 'off'

attribute: 'conStr'
description: 'Database Connection String'
defaultValue: ''
allowableValues: ''
```

```
attribute: 'type'
description: 'Type of the database'
defaultValue: 'star'
allowableValues: ''

attribute: 'hostType'
description: 'Type of Host'
defaultValue: 'local'
allowableValues: 'local', 'remote'

attribute: 'status'
description: 'Status of the Connection String'
defaultValue: '1'
allowableValues: '1', '0'

attribute: 'rank'
description: 'Database Rank'
defaultValue: '0'
allowableValues: ''

attribute: 'format'
description: 'Specify the service response format.'
defaultValue: 'json'
allowableValues: 'json', 'api', 'xml', 'text', 'custom'


View the service schema at ServiceSchema

View the service wsdl 2.0 at ServiceWSDL
```

- **author:** Krishna R. Kantam <krishna@txcorp.com>
- **author:** Mark L. Green <mlgreen@txcorp.com>
- **version:** Release: @package_version@
- **copyright:** 2006-2010 Tech-X Corporation. All rights reserved.
- **link:** http://www.txcorp.com/
- **license:** BSD License

*Located in **/OrbiterFederation/classes/OrbiterQueryDbConnectionStringService.class.php** (line **171**)*

```
OrbiterService
   |
   --OrbiterQueryDbConnectionStringService
```

Method Summary

Description | Vars | *Methods* (details)

- *OrbiterQueryDbConnectionStringService* **__construct** ()
- *void* **__destruct** ()
- *bool* **deleteConstr** ()
- *bool* **insertConstr** ()
- *void* **processAttributes** ()
- *void* **processOperations** ()
- *void* **processServiceRequest** ()
- *bool* **updateConstr** ()

Variables

Description | Vars (*details*) | Methods (details)

Inherited Variables

Inherited from OrbiterService

- **OrbiterService::$_auth**
- **OrbiterService::$_cnt**
- **OrbiterService::$_contents**
- **OrbiterService::$_count**
- **OrbiterService::$_dbMaster**
- **OrbiterService::$_dbSlave**
- **OrbiterService::$_definedAttr**
- **OrbiterService::$_errorHandler**
- **OrbiterService::$_logger**
- **OrbiterService::$_phpVariables**
- **OrbiterService::$_queryString**
- **OrbiterService::$_requestMethod**

- OrbiterService::$requestUri
- OrbiterService::$_scriptName
- OrbiterService::$_serviceAddress
- OrbiterService::$_serviceSchema
- OrbiterService::$_userId
- OrbiterService::$_userRole

Methods

Description | Vars Methods (*details*)

Constructor __construct (line 176)

Constructs a new OrbiterQueryDbConnectionStringService.class instance

OrbiterQueryDbConnectionStringService **__construct** ()

Redefinition of:

OrbiterService::__construct()
Constructs a new OrbiterService abstract class instance

Destructor __destruct (line 193)

Destructs a OrbiterQueryDbConnectionStringService class instance

void **__destruct** ()

Redefinition of:

OrbiterService::__destruct()
Destructs a OrbiterService abstract class instance

deleteConstr (line 469)

Deletes the connection string from db_constr table.

- **return:** returns TRUE on success and FALSE on failure
- **exception:** E_USER_ERROR Orbiter Error 150:1523 E_USER_ERROR Database query error

bool **deleteConstr** ()

insertConstr (line 422)

Inserts the connection string into db_conStr table.

- **return:** returns TRUE on success and FALSE on failure
- **exception:** E_USER_ERROR Orbiter Error 150:1520 E_USER_ERROR Database query error
- **exception:** E_USER_ERROR Orbiter Error 150:1524 E_USER_ERROR Database Connection string already exists in the table
- **exception:** E_USER_ERROR Orbiter Error 150:1521 E_USER_ERROR Database query error

bool **insertConstr** ()

processAttributes (line 200)

Process the service attributes

- **access:** protected

void **processAttributes** ()

Redefinition of:

OrbiterService::processAttributes()
Abstract function for setting the default attributes the service

processOperations (line 213)

Process the service operations

- **access:** protected

void **processOperations** ()

Redefinition of:

OrbiterService::processOperations()
Abstract function for setting the operations for the service

processServiceRequest (line 384)

Processes the service request based on the operation the user specified

```
--------------------------------------------------------------------------------------------

Usage: No required attributes

operation: 'api'
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterQueryDbConnectionStringService.php/operation/api

Response: Returns the api of the service in 'json' format

[{"service":"{Service Name}","attributes":{"operation":{"default":"","restrictions":
[List of Operations]},..List of attributes}, "operations":{"0":[],"operation1":{"GET":
{"description":"{Operation Description}","restrictions":{"required":[List of required attributes],
"optional":[List of optional attributes]}}},..List of Operations}}]


Usage: with optional attribute 'format'

operation: 'api'
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterQueryDbConnectionStringService.php/operation/api/format/api

Response: Returns the api of the service in 'api' format as

array(
     'service'=>{ServiceName},
     'attributes'=>array(
         'operation'=>array(
             'default'=>{Default Value},
             'restrictions'=>array(
                List of Allowable Values
             )
          ),.. List of attributes
     ),
     'operations'=>array(
         'operation1'=>array(
             'method type'=>array(
                 'description'=>{operation description}
                 'restrictions'=>array(
                     'required'=>array(
                         List of required attributes,
                     )
                     'optional'=>array(
                         List of optional attributes
                     )
                  )
              )
         ), .. List of operations
     )
)

--------------------------------------------------------------------------------------------

Usage: With required attributes 'type' and 'conStr'

operation: 'insert'
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterQueryDbConnectionStringService.php/operation/insert/type/star/
conStr/ORBITERDBQ1

Response: Returns success message with connection string and database type added to the database

Connection string inserted successfully


Usage: With optional attribute 'rank'

operation: 'insert'
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterQueryDbConnectionStringService.php/operation/insert/type/star/
conStr/ORBITERDBQ1/rank/1

Response: Returns success message with connection string, database and of the user specified DB rank added to the

Connection string inserted successfully
```

```
Usage: With optional attributes 'hostType'

operation: 'insert'
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterQueryDbConnectionStringService.php/operation/insert/type/star/
conStr/ORBITERDBQ1/hostType/remotehost

Response: Returns success message with connection string, database and of the user specified DB host type added to

Connection string inserted successfully

-------------------------------------------------------------------------------------------

Usage: With required attributes 'type' and 'conStr'

operation: 'update'
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterQueryDbConnectionStringService.php/operation/update/type/star/
conStr/ORBITERDBQ1

Response: Returns the success message indicating that the specified database was updated

Connection string inserted successfully


Usage: With optional attributes 'status'

operation: 'update'
URI:/orbiter/ORBITERVERSION/service/webservice/
OOrbiterQueryDbConnectionStringService.php/operation/insert/type/star/
conStr/ORBITERDBQ1/status/1

Response: Returns the success message indicating that the specified database was updated

Connection string inserted successfully


Usage: With optional attributes 'hostType'

operation: 'update'
URI:/orbiter/ORBITERVERSION/service/webservice/
OOrbiterQueryDbConnectionStringService.php/operation/insert/type/star/
conStr/ORBITERDBQ1/hostType/remotehost

Response: Returns the success message indicating that the specified database was updated

Connection string inserted successfully


Usage: With optional attributes 'rank'

operation: 'update'
URI:/orbiter/ORBITERVERSION/service/webservice/
OOrbiterQueryDbConnectionStringService.php/operation/insert/type/star/
conStr/ORBITERDBQ1/rank/1

Response: Returns the success message indicating that the specified database was updated

Connection string inserted successfully

-------------------------------------------------------------------------------------------

Usage: With required attributes 'conStr'

operation: 'delete'
URI:/orbiter/ORBITERVERSION/service/webservice/
OOrbiterQueryDbConnectionStringService.php/operation/delete/
conStr/ORBITERDBQ1

Response: Returns the success message indicating that the specified database was deleted

Connection string deleted successfully

-------------------------------------------------------------------------------------------
```

void **processServiceRequest** ()

Redefinition of:

OrbiterService::processServiceRequest()
Abstract function for the business logic for the service request

updateConstr (line 451)

Updates the connnection string in db_constr table by setting the rank and status.

- **return:** returns TRUE on success and FALSE on failure
- **exception:** E_USER_ERROR Orbiter Error 1501:1522 E_USER_ERROR Database query error

bool **updateConstr** ()

Inherited Methods

Inherited From OrbiterService

- **OrbiterService::__construct()**
- **OrbiterService::authenticateServiceRequest()**
- **OrbiterService::checkAttributeValidity()**
- **OrbiterService::checkInterfaceRequest()**
- **OrbiterService::checkOperationValidity()**
- **OrbiterService::defineServiceAttributes()**
- **OrbiterService::defineServiceOperations()**
- **OrbiterService::doServiceRequest()**
- **OrbiterService::endLogServiceRequest()**
- **OrbiterService::getServiceAddress()**
- **OrbiterService::logServiceRequest()**
- **OrbiterService::processAttributes()**
- **OrbiterService::processOperations()**
- **OrbiterService::processServiceAttributes()**
- **OrbiterService::processServiceRequest()**
- **OrbiterService::processServiceResponse()**
- **OrbiterService::setDatabaseConnections()**
- **OrbiterService::setServiceAddress()**
- **OrbiterService::triggerError()**
- **OrbiterService::__destruct()**

OrbiterQueryDbLoadBalancerService

Description

Description | Vars | Methods (details)

This service is responsible for load balancing the query and other databases. Also used for updating the database rank and status. The Service response will return connection strings of the type specified by the user based on the rank and status.

Brief example of use:

```
Create an instance of OrbiterQueryDbLoadBalancerService
$objService = new OrbiterQueryDbLoadBalancerService();

---------------------------------------------------------------------------------------------

Service name
OrbiterQueryDbLoadBalancerService.php

---------------------------------------------------------------------------------------------

Service Operation Definitions

Summary Format
operation:
description:
required attributes:
optional attributes:
service method:

---------------------------------------------------------------------------------------------

operation:'api'
description: 'List the service API.'
required attributes: ''
optional attributes: 'format'
service method: 'GET'

operation:'get'
description: 'Get the connection strings'
required attributes: 'endpoints', 'type'
optional attributes: 'format'
service method: 'GET'

operation:'update'
description: 'Update the connection string'
required attributes: 'rank', 'conStr'
optional attributes: 'status'
service method: 'GET'

---------------------------------------------------------------------------------------------

Service Attribute Definitions

Summary Format
attribute:
description:
defaultValue:
allowableValues:

---------------------------------------------------------------------------------------------

attribute: 'operation'
description: 'Specify the type of operation service request required.'
defaultValue: ''
allowableValues: 'api', 'get', 'update'

attribute: 'html'
description: 'Generate HTML compliant output by replacing escaped character and use
```

```
            appropriate tagging and line returns.|
defaultValue: 'off'
allowableValues: 'on', 'off'

attribute: 'conStr'
description: 'Database Connection String'
defaultValue: ''
allowableValues: ''

attribute: 'type'
description: 'Type of the database'
defaultValue: ''
allowableValues: ''

attribute: 'endpoints'
description: 'Number of connection strings to be retrieved'
defaultValue: ''
allowableValues: ''

attribute: 'status'
description: 'Status of the Connection String'
defaultValue: '1'
allowableValues: '1', '0'

attribute: 'rank'
description: 'Database Rank'
defaultValue: ''
allowableValues: ''

attribute: 'format'
description: 'Specify the service response format.'
defaultValue: 'json'
allowableValues: 'json', 'api', 'xml', 'text', 'custom'


View the service schema at ServiceSchema

View the service wsdl 2.0 at ServiceWSDL
```

- **author:** Mark L. Green <mlgreen@txcorp.com>
- **version:** Release: @package_version@
- **copyright:** 2006-2010 Tech-X Corporation. All rights reserved.
- **link:** http://www.txcorp.com/
- **license:** BSD License

*Located in **/OrbiterFederation/classes/OrbiterQueryDbLoadBalancerService.class.php** (line **168**)*

```
OrbiterService
   |
   --OrbiterQueryDbLoadBalancerService
```

Method Summary

Description | Vars | *Methods* (details)

- *OrbiterQueryDbLoadBalancerService* **__construct** ()
- *void* **__destruct** ()
- *array* **getConStrings** ([*integer* **$endPoints** = null], [*string* **$type** = null])
- *void* **processAttributes** ()
- *void* **processOperations** ()
- *void* **processServiceRequest** ()
- *bool* **updateConStr** ()

Variables

Description | Vars (*details*) | Methods (details)

Inherited Variables

Inherited from OrbiterService

- **OrbiterService::$_auth**
- **OrbiterService::$_cnt**
- **OrbiterService::$_contents**
- **OrbiterService::$_count**
- **OrbiterService::$_dbMaster**
- **OrbiterService::$_dbSlave**
- **OrbiterService::$_definedAttr**
- **OrbiterService::$_errorHandler**
- **OrbiterService::$_logger**
- **OrbiterService::$_phpVariables**
- **OrbiterService::$_queryString**
- **OrbiterService::$_requestMethod**
- **OrbiterService::$_requestUri**
- **OrbiterService::$_scriptName**
- **OrbiterService::$_serviceAddress**
- **OrbiterService::$_serviceSchema**
- **OrbiterService::$_userId**
- **OrbiterService::$_userRole**

Methods

Description | Vars Methods (*details*)

Constructor __construct (line 173)

Constructs a new OrbiterQueryDbLoadBalancerService.class instance

OrbiterQueryDbLoadBalancerService **__construct** ()

Redefinition of:

OrbiterService::__construct()
Constructs a new OrbiterService abstract class instance

Destructor __destruct (line 190)

Destructs a OrbiterQueryDbLoadBalancerService class instance

void **__destruct** ()

Redefinition of:

OrbiterService::__destruct()
Destructs a OrbiterService abstract class instance

getConStrings (line 376)

Gets the id and connection string information from db_constr table

- **return:** Array of Connection Strings
- **exception:** E_USER_ERROR Orbiter Error 151:1530 E_USER_ERROR Database query error
- **exception:** E_USER_ERROR Orbiter Error 151:1531 E_USER_ERROR Missing the endpoints or the type

array **getConStrings** ([*integer* **$endPoints** = null], [*string* **$type** = null])

- *integer* **$endPoints**: number of connection string to be returned
- *string* **$type**: database type

processAttributes (line 197)

Process the service attributes

- **access:** protected

void **processAttributes** ()

Redefinition of:

OrbiterService::processAttributes()
Abstract function for setting the default attributes the service

processOperations (line 210)

Process the service operations

- **access:** protected

void **processOperations** ()

Redefinition of:

OrbiterService::processOperations()
Abstract function for setting the operations for the service

processServiceRequest (line 325)

Processes the service request based on the operation the user specified

```
-------------------------------------------------------------------------------------------

Usage: No required attributes

operation: 'api'
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterQueryDbLoadBalancerService.php/operation/api

Response: Returns the api of the service in 'json' format

[{"service":"{Service Name}","attributes":{"operation":{"default":"","restrictions":
[List of Operations]},..List of attributes}, "operations":{"0":[],"operation1":{"GET":
{"description":"{Operation Description}","restrictions":{"required":[List of required attributes],
"optional":[List of optional attributes]}}},..List of Operations}}]


Usage: with optional attribute 'format'

operation: 'api'
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterQueryDbLoadBalancerService.php/operation/api/format/api

Response: Returns the api of the service in 'api' format as

array(
     'service'=>{ServiceName},
     'attributes'=>array(
         'operation'=>array(
             'default'=>{Default Value},
             'restrictions'=>array(
                List of Allowable Values
             )
          ),.. List of attributes
     ),
     'operations'=>array(
         'operation1'=>array(
             'method type'=>array(
                 'description'=>{operation description}
                 'restrictions'=>array(
                     'required'=>array(
                         List of required attributes,
                     )
```

```
                    'optional'=>array(
                        List of optional attributes
                    )
                )
            )
        ), .. List of operations
    )
)
```

```
Usage: With required attributes 'endpoints' and 'type'

operation: 'get'
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterQueryDbLoadBalancerService.php/operation/get/endpoints/2/type/star

Response: Returns the connection strings for the database type and number of endpoints

{"1":"localhost|DB_USERNAME|DB_USERNAME_PASSWORD|DB_NAME"}


Usage: With optional attribute 'format'

operation: 'get'
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterQueryDbLoadBalancerService.php/operation/get/endpoints/2/type/star/format/text

Response: Returns the connection strings for the database type and number of endpoints in plain text for

[1] => localhost|DB_USERNAME|DB_USERNAME_PASSWORD|DB_NAME

---

Usage: With required attributes 'rank' and 'conStr'

operation: 'update'
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterQueryDbLoadBalancerService.php/operation/update/rank/2/
conStr/ORBITERDBQLOCAL

Response: Returns the success message indicating that the databse was updated with the rank set by the u

"Update was successful"


Usage: With optional attributes 'status'
operation: 'update'
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterQueryDbLoadBalancerService.php/operation/update/rank/2/
conStr/ORBITERDBQLOCAL/status/1

Response: Returns the success message indicating that the databse was updated with the rank set by the u

---
```

void **processServiceRequest** ()

Redefinition of:

OrbiterService::processServiceRequest()
Abstract function for the business logic for the service request

updateConStr (line 354)

Updates the db_constr table and sets the rank and status.

- **return:** Return true on success and false on failure
- **exception:** E_USER_ERROR Orbiter Error 151:1529 E_USER_ERROR Database query error

bool **updateConStr** ()

Inherited Methods

Inherited From OrbiterService

- **OrbiterService::__construct()**
- **OrbiterService::authenticateServiceRequest()**
- **OrbiterService::checkAttributeValidity()**
- **OrbiterService::checkInterfaceRequest()**
- **OrbiterService::checkOperationValidity()**
- **OrbiterService::defineServiceAttributes()**
- **OrbiterService::defineServiceOperations()**
- **OrbiterService::doServiceRequest()**
- **OrbiterService::endLogServiceRequest()**
- **OrbiterService::getServiceAddress()**
- **OrbiterService::logServiceRequest()**
- **OrbiterService::processAttributes()**
- **OrbiterService::processOperations()**
- **OrbiterService::processServiceAttributes()**
- **OrbiterService::processServiceRequest()**
- **OrbiterService::processServiceResponse()**
- **OrbiterService::setDatabaseConnections()**
- **OrbiterService::setServiceAddress()**
- **OrbiterService::triggerError()**
- **OrbiterService::__destruct()**

Description

Description | Vars (details) | Methods (details)

This is the STAR query service that is useful to connect and run queries against a query database. This query service can cache the query result and validate against query result and return it when same sql query comes in, instead of going to the database to get the result set.

Brief example of use:

```
Create an instance of OrbiterQueryService
$objService = new OrbiterQueryService();

-----------------------------------------------------------------------------------------------

Service name
OrbiterQueryService.php

-----------------------------------------------------------------------------------------------

Service Operation Definitions

Summary Format
operation:
description:
required attributes:
optional attributes:
service method:

-----------------------------------------------------------------------------------------------

operation:'api'
description: 'List the service API.'
required attributes: ''
optional attributes: 'format'
service method: 'GET'

operation:'get'
description: 'Get the query result'
required attributes: 'query', 'database'
optional attributes: 'format', 'cache', 'vo', 'role', 'useEndPoints', 'validation'
service method: 'GET'

operation:'noop'
description: 'Get the timing'
required attributes: ''
optional attributes: 'timing'
service method: 'GET'

-----------------------------------------------------------------------------------------------

Service Attribute Definitions

Summary Format
attribute:
description:
defaultValue:
allowableValues:

-----------------------------------------------------------------------------------------------

attribute: 'operation'
description: 'Specify the type of operation service request required.'
defaultValue: ''
allowableValues: 'api', 'get', 'noop'

attribute: 'html'
description: 'Generate HTML compliant output by replacing escaped character and use
              appropriate tagging and line returns.'
defaultValue: 'off'
allowableValues: 'on', 'off'

attribute: 'query'
description: 'Specify sql query with urlencode.'
defaultValue: ''
allowableValues: ''

attribute: 'database'
description: 'Specify database name on which query can be run.'
defaultValue: ''
allowableValues: ''

attribute: 'timing'
description: 'Specify whether to log response time in the service log.'
defaultValue: 'off'
allowableValues: 'on','off'

attribute: 'cache'
description: 'Specify whether to keep cache on.'
```

```
defaultValue: 'on'
allowableValues: 'on', 'off'

attribute: 'validation'
description: 'Compares the database response contents and the cache contents
when set to "on"'
defaultValue: 'off'
allowableValues: 'on', 'off'

attribute: 'vo'
description: 'Virtual Organization Name'
defaultValue: ''
allowableValues: ''

attribute: 'role'
description: 'Role in the Virtual Organization'
defaultValue: ''
allowableValues: ''

attribute: 'useEndpoints'
description: 'Gets the number of connection strings from load balancer
when set to more than 0'
defaultValue: '0'
allowableValues: ''

attribute: 'format'
description: 'Specify the service response format.'
defaultValue: 'json'
allowableValues: 'json', 'api', 'xml', 'text', 'custom'


View the service schema at ServiceSchema

View the service wsdl 2.0 at ServiceWSDL
```

- **author:** Krishna R. Kantam <krishna@txcorp.com>
- **author:** Mark L. Green <mlgreen@txcorp.com>
- **version:** Release: @package_version@
- **copyright:** 2006-2010 Tech-X Corporation. All rights reserved.
- **link:** http://www.txcorp.com/
- **license:** BSD License

*Located in **/OrbiterFederation/classes/OrbiterQueryService.class.php** (line **186**)*

```
OrbiterService
   |
   --OrbiterQueryService
```

Variable Summary

Description | *Vars* (details) | Methods (details)

OrbiterDatabaseConnection **$_dbQueryConn**

Method Summary

Description | Vars (details) | *Methods* (details)

OrbiterQueryService **__construct** ()
void **__destruct** ()
void **endLogServiceRequest** ()
string **getDirectoryStructure** ()
void **processAttributes** ()
void **processOperations** ()
void **processServiceAttributes** ()
void **processServiceRequest** ()
string **validateContents** (*array* **$contents**, *array* **$query**)
boolean **validateVoRole** ()

Variables

Description | Vars (*details*) | Methods (details)

OrbiterDatabaseConnection **$_dbQueryConn** (line 190)

- **var:** Orbiter database connection object
- **access:** protected

Inherited Variables

Inherited from OrbiterService

OrbiterService::$_auth
OrbiterService::$_cnt
OrbiterService::$_contents

- OrbiterService::$count
- OrbiterService::$_dbMaster
- OrbiterService::$_dbSlave
- OrbiterService::$_definedAttr
- OrbiterService::$_errorHandler
- OrbiterService::$_logger
- OrbiterService::$_phpVariables
- OrbiterService::$_queryString
- OrbiterService::$_requestMethod
- OrbiterService::$_requestUri
- OrbiterService::$_scriptName
- OrbiterService::$_serviceAddress
- OrbiterService::$_serviceSchema
- OrbiterService::$_userId
- OrbiterService::$_userRole

Methods

Description | Vars (details) Methods (*details*)

Constructor __construct (line 211)

Constructs a new OrbiterQueryService.class instance

OrbiterQueryService **__construct** ()

Redefinition of:

OrbiterService::__construct()
Constructs a new OrbiterService abstract class instance

Destructor __destruct (line 232)

Destructs a OrbiterQueryService class instance

void **__destruct** ()

Redefinition of:

OrbiterService::__destruct()
Destructs a OrbiterService abstract class instance

endLogServiceRequest (line 647)

Ends logging the service request after sending the response status to service log.

- **access:** protected

void **endLogServiceRequest** ()

Redefinition of:

OrbiterService::endLogServiceRequest()
Ends logging the service request.

getDirectoryStructure (line 707)

Gets the Virtual Organization Directory structure.

- **return:** Returns Virtual Organization Directory structure.
- **access:** protected

string **getDirectoryStructure** ()

processAttributes (line 239)

Process the service attributes

- **access:** protected

void **processAttributes** ()

Redefinition of:

OrbiterService::processAttributes()
Abstract function for setting the default attributes the service

processOperations (line 255)

Process the service operations

- **access:** protected

void **processOperations** ()

Redefinition of:

OrbiterService::processOperations()
Abstract function for setting the operations for the service

processServiceAttributes (line 421)

Processes the service request based on the operation the user specified

```
--------------------------------------------------------------------------------------------
Usage: No required attributes

operation: 'api'
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterQueryService.php/operation/api

Response: Returns the api of the service in 'json' format

[{"service":"{Service Name}","attributes":{"operation":{"default":"","restrictions":
[List of Operations]},..List of attributes}, "operations":{"0":[],"operation1":{"GET":
{"description":"{Operation Description}","restrictions":{"required":[List of required attributes],
"optional":[List of optional attributes]}}},..List of Operations}}]


Usage: with optional attribute 'format'

operation: 'api'
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterQueryService.php/operation/api/format/api

Response: Returns the api of the service in 'api' format as

array(
     'service'=>{ServiceName},
     'attributes'=>array(
         'operation'=>array(
             'default'=>{Default Value},
             'restrictions'=>array(
                List of Allowable Values
             )
          ),.. List of attributes
     ),
     'operations'=>array(
         'operation1'=>array(
             'method type'=>array(
                 'description'=>{operation description}
                 'restrictions'=>array(
                     'required'=>array(
                         List of required attributes,
                     )
                     'optional'=>array(
                         List of optional attributes
                     )
                  )
              )
         ), .. List of operations
     )
)

--------------------------------------------------------------------------------------------

Usage: No required attributes

operation: 'noop'
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterQueryService.php/operation/noop

Response: Returns the response time of the service without any operation in seconds as

[{"Response Time":0.0053920745849609}]


Usage: With optional attribute 'timing'

operation: 'noop'
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterQueryService.php/operation/noop/timing/on

Response: Returns the response time of the service without any operation in seconds as

[{"Response Time":0.0053920745849609}]

--------------------------------------------------------------------------------------------

Usage: With required attributes 'database' and 'query'

operation: 'get'
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterQueryService.php/operation/get/database/orbiter/
```

```
query/SELECT%20file_type_id%20FROM%20file_type%20WHERE%201

Response: Returns the query results made to the database specified

[{"file_type_id":"1"}]


Usage: With optional attribute 'format'

operation: 'get'
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterQueryService.php/operation/get/database/orbiter/
query/SELECT%20file_type_id%20FROM%20file_type%20WHERE%201/format/xml

Response: Returns the query results made to the database specified in xml format


Usage: With optional attribute 'cache'

operation: 'get'
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterQueryService.php/operation/get/database/orbiter/
query/SELECT%20file_type_id%20FROM%20file_type%20WHERE%201/cache/on

Response: Returns the query results made to the database specified and caches them in the ORBITERCACHEFILELOCATION set i


Usage: With optional attribute 'vo'

operation: 'get'
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterQueryService.php/operation/get/database/orbiter/
query/SELECT%20file_type_id%20FROM%20file_type%20WHERE%201/vo/{vo}

Response: Returns the query results made to the database specified and sets them to the Virtual Organization specified


Usage: With optional attribute 'role'

operation: 'get'
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterQueryService.php/operation/get/database/orbiter/
query/SELECT%20file_type_id%20FROM%20file_type%20WHERE%201/role/{role}

Response: Returns the query results made to the database specified


Usage: With optional attribute 'useEndpoints'

operation: 'get'
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterQueryService.php/operation/get/database/orbiter/
query/SELECT%20file_type_id%20FROM%20file_type%20WHERE%201/useEndpoints/4

Response: Returns the query results made to the database specified


Usage: With optional attribute 'validation'

operation: 'get'
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterQueryService.php/operation/get/database/orbiter/
query/SELECT%20file_type_id%20FROM%20file_type%20WHERE%201/validation/on

Response: Returns the query results made to the database specified

-------------------------------------------------------------------------------------------
```

- **exception:** E_USER_ERROR Orbiter Error 149:1534 E_USER_ERROR Post data has duplicate service attributes in the URI
- **access:** protected

void **processServiceAttributes** ()

Redefinition of:

OrbiterService::processServiceAttributes()
Get the service attributes from the authenticated service request and process the service attributes.

processServiceRequest (line 492)

Processes the service request based on the operation the user specified

- **exception:** E_USER_ERROR Orbiter Error 149:1511 E_USER_ERROR Database query error
- **exception:** E_USER_ERROR Orbiter Error 149:1510 E_USER_ERROR Database query error
- **access:** protected

void **processServiceRequest** ()

Redefinition of:

OrbiterService::processServiceRequest()
Abstract function for the business logic for the service request

validateContents (line 679)

Compares the Data base response contents and the cache contents.

- **return:** Returns a string message to the user.
- **exception:** E_USER_ERROR Orbiter Error 149:1514 E_USER_ERROR Query and Contents has to be defined.
- **access:** protected

string **validateContents** (*array* **$contents**, *array* **$query**)

- *array* **$contents**: Array of contents from Database.
- *array* **$query**: Database query which is canonical string.

validateVoRole (line 719)

Validates the Virtual Oranization name and role to be alpha numberic.

- **return:** Returns true/ triggers an error incase of an invalid input.
- **exception:** E_USER_ERROR Orbiter Error 149:1615 E_USER_ERROR Virtual Organization name and role must be valid.
- **access:** protected

boolean **validateVoRole** ()

Inherited Methods

Inherited From OrbiterService

- **OrbiterService::__construct()**
- **OrbiterService::authenticateServiceRequest()**
- **OrbiterService::checkAttributeValidity()**
- **OrbiterService::checkInterfaceRequest()**
- **OrbiterService::checkOperationValidity()**
- **OrbiterService::defineServiceAttributes()**
- **OrbiterService::defineServiceOperations()**
- **OrbiterService::doServiceRequest()**
- **OrbiterService::endLogServiceRequest()**
- **OrbiterService::getServiceAddress()**
- **OrbiterService::logServiceRequest()**
- **OrbiterService::processAttributes()**
- **OrbiterService::processOperations()**
- **OrbiterService::processServiceAttributes()**
- **OrbiterService::processServiceRequest()**
- **OrbiterService::processServiceResponse()**
- **OrbiterService::setDatabaseConnections()**
- **OrbiterService::setServiceAddress()**
- **OrbiterService::triggerError()**
- **OrbiterService::__destruct()**

OrbiterResourcePreCacheService

Description

This service provides the user an ability to pre-cache Query Db sql queries when a new database resource was added to resource table.

Brief example of use:

```
// Create an instance of OrbiterResourcePreCacheService
$objService = new OrbiterResourcePreCacheService();

---------------------------------------------------------------------------------------------

Service name
OrbiterResourcePreCacheService.php

---------------------------------------------------------------------------------------------

Service Operation Definitions

Summary Format
operation:
description:
required attributes:
optional attributes:
service method:

---------------------------------------------------------------------------------------------

operation:'api'
description: 'List the service API.'
required attributes: ''
optional attributes: 'format'
service method: 'GET'

operation:'runPreCache'
description: 'Run pre-caching for a given resource.'
required attributes: 'resource'
optional attributes: 'validatePreCache', 'format'
service method: 'GET'

operation:'flushPreCache'
description: 'Delete the cached files.'
required attributes: 'resource'
optional attributes: 'format'
service method: 'GET'

---------------------------------------------------------------------------------------------

Service Attribute Definitions

Summary Format
attribute:
description:
defaultValue:
allowableValues:

---------------------------------------------------------------------------------------------

attribute: 'operation'
description: 'Specify the type of operation service request required.'
```

```
defaultValue: ''
allowableValues: 'api', 'runPreCache', 'flushPreCache'

attribute: 'validatePreCache'
description: 'When 'on' the pre cache will be validated with query result.'
defaultValue: 'off'
allowableValues: 'on', 'off'

attribute: 'resource'
description: 'Specify the resource.'
defaultValue: ''
allowableValues: ''

attribute: 'format'
description: 'Specify the service response format.'
defaultValue: 'json'
allowableValues: 'json', 'api', 'xml', 'text'


View the service schema at ServiceSchema

View the service wsdl 2.0 at ServiceWSDL
```

- **author:** Krishna R. Kantam <krishna@txcorp.com>
- **author:** Mark L. Green <mlgreen@txcorp.com>
- **version:** Release: @package_version@
- **copyright:** 2006-2010 Tech-X Corporation. All rights reserved.
- **link:** http://www.txcorp.com/
- **todo:** Investigate appending to the contents array the messages that the service gives as a response
- **todo:** Should OrbiterResourcePreCacheService use role and vo
- **todo:** Investigate service attributes to allow for setting the configuration variables ORBITERSQLFILELOCATION or ORBITERCACHELOCATION
- **todo:** Investigate adding additional validatePreCache functionality
- **todo:** -Consider adding error triggering if the return for validatePreCache is false
- **todo:** -Consider making it its own operation
- **license:** BSD License

*Located in **/OrbiterFederation/classes/OrbiterResourcePreCacheService.class.php** (line **150**)*

```
OrbiterService
   |
   --OrbiterResourcePreCacheService
```

Method Summary

Description | Vars | *Methods* (details)

- *OrbiterResourcePreCacheService* **__construct** ()
- *void* **__destruct** ()
- *bool* **flushPreCache** ()
- *bool* **isHashFile** (*string* **$filename**, *string* **$hashtype**)
- *void* **processAttributes** ()
- *void* **processOperations** ()
- *void* **processServiceRequest** ()
- *bool* **runPreCache** ()
- *string* **validateContents** (*array* **$contents**, *array* **$query**)

Variables

Description | Vars (*details*) | Methods (details)

Inherited Variables

Inherited from OrbiterService

- **OrbiterService::$_auth**
- **OrbiterService::$_cnt**
- **OrbiterService::$_contents**
- **OrbiterService::$_count**
- **OrbiterService::$_dbMaster**
- **OrbiterService::$_dbSlave**
- **OrbiterService::$_definedAttr**
- **OrbiterService::$_errorHandler**
- **OrbiterService::$_logger**
- **OrbiterService::$_phpVariables**
- **OrbiterService::$_queryString**
- **OrbiterService::$_requestMethod**
- **OrbiterService::$_requestUri**
- **OrbiterService::$_scriptName**
- **OrbiterService::$_serviceAddress**
- **OrbiterService::$_serviceSchema**
- **OrbiterService::$_userId**
- **OrbiterService::$_userRole**

Methods

Description | Vars Methods (*details*)

Constructor __construct (line 194)

Constructs a new OrbiterResourcePreCacheService class instance

- **exception:** E_USER_ERROR Orbiter Error 159:1634 E_USER_ERROR Sql file location must be defined
- **exception:** E_USER_ERROR Orbiter Error 159:1635 E_USER_ERROR Cache file location needs to be defined
- **exception:** E_USER_ERROR Orbiter Error 159:1636 E_USER_ERROR Hash type must be defined

OrbiterResourcePreCacheService **__construct** ()

Redefinition of:

OrbiterService::__construct()
Constructs a new OrbiterService abstract class instance

Destructor __destruct (line 239)

Destructs a OrbiterResourcePreCacheService class instance

void **__destruct** ()

Redefinition of:

OrbiterService::__destruct()
Destructs a OrbiterService abstract class instance

flushPreCache (line 537)

Flush the pre-cache for the given resource.

- **return:** Returns true on success of flushing pre-cache
- **exception:** E_USER_ERROR Orbiter Error 160:1647 E_USER_ERROR Cannot open directory to delete the cache files
- **exception:** E_USER_ERROR Orbiter Error 160:1648 E_USER_ERROR Not a valid directory to delete cache files

bool **flushPreCache** ()

isHashFile (line 566)

Checks if the file is hashed.

- **return:** Returns true if the file is a hash file.
- **exception:** E_USER_ERROR Orbiter Error 160:1649 E_USER_ERROR Unknown hash type request

bool **isHashFile** (*string* **$filename**, *string* **$hashtype**)

- *string* **$filename**: Name of the cache file.
- *string* **$hashtype**: Type of hash sha1/md5

processAttributes (line 246)

Process the service attributes

- **access:** protected

void **processAttributes** ()

Redefinition of:

OrbiterService::processAttributes()
Abstract function for setting the default attributes the service

processOperations (line 256)

Process the service operations

- **access:** protected

void **processOperations** ()

Redefinition of:

OrbiterService::processOperations()
Abstract function for setting the operations for the service

processServiceRequest (line 368)

Processes the service request based on the operation the user specified

```
-------------------------------------------------------------------------------------------

Usage: No required attributes

operation: 'api'
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterResourcePreCacheService.php/operation/api

Response: Returns the api of the service in 'json' format

[{"service":"{Service Name}","attributes":{"operation":{"default":"","restrictions":
```

[List of Operations]},..List of attributes}, "operations":{"0":[],"operation1":{"GET":
{"description":"{Operation Description}","restrictions":{"required":[List of required attributes],
"optional":[List of optional attributes]}}},..List of Operations}}]

Usage: with optional attribute 'format'

operation: 'api'
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterResourcePreCacheService.php/operation/api/format/api

Response: Returns the api of the service in 'api' format as

```
array(
     'service'=>{ServiceName},
     'attributes'=>array(
         'operation'=>array(
             'default'=>{Default Value},
             'restrictions'=>array(
                List of Allowable Values
             )
          ),.. List of attributes
     ),
     'operations'=>array(
         'operation1'=>array(
             'method type'=>array(
                 'description'=>{operation description}
                 'restrictions'=>array(
                     'required'=>array(
                         List of required attributes,
                     )
                     'optional'=>array(
                         List of optional attributes
                     )
                  )
              )
         ), .. List of operations
     )
)
```

Usage: With required attributes 'resource'

operation: 'runPreCache'
URI:/orbiter/ORBITERVERSION/service/webservice/OrbiterResourcePreCacheService.php/
operation/runPreCache/resource/ORBITERDBQLOCAL

Response: Returns success pre-cache message

Sql query pre-cache was successful for precaching

Usage: With optional attribute 'validatePreCache'

operation: 'runPreCache'
URI:/orbiter/ORBITERVERSION/service/webservice/OrbiterResourcePreCacheService.php/
operation/runPreCache/resource/ORBITERDBQLOCAL/validatePreCache/on

Response: Returns success message for precaching and validation of precache

Sql query pre-cache was successful

Usage: With required attributes 'resource'

```
operation: 'flushPreCache'
URI:/orbiter/ORBITERVERSION/service/webservice/OrbiterResourcePreCacheService.php/
operation/flushPreCache/resource/ORBITERDBQLOCAL

Response: Returns success flush pre-cache message

Sql query flushing pre-cache was successful


Usage: With optional attribute 'format'

operation: 'flushPreCache'
URI:/orbiter/ORBITERVERSION/service/webservice/OrbiterResourcePreCacheService.php/
operation/flushPreCache/resource/ORBITERDBQLOCAL/format/xml

Response: Returns success flush pre-cache message in xml format similar to json format

-------------------------------------------------------------------------------------
```

void **processServiceRequest** ()

Redefinition of:

OrbiterService::processServiceRequest()
Abstract function for the business logic for the service request

runPreCache (line 404)

Get the sql queries from config location and run the pre-cache for the given resource.

- **return:** Returns true on success of running pre-cache
- **exception:** E_USER_ERROR Orbiter Error 160:1643 E_USER_ERROR Database query error
- **exception:** E_USER_ERROR Orbiter Error 160:1642 E_USER_ERROR Improper use statement 'sql query'
- **exception:** E_USER_ERROR Orbiter Error 160:1644 E_USER_ERROR Can't write to the cache file, permission denied
- **exception:** E_USER_ERROR Orbiter Error 160:1645 E_USER_ERROR The file is not readable or does not exist 'Star file location'
- **exception:** E_USER_ERROR Orbiter Error 160:1886 E_USER_ERROR Invalid resource name for connection string
- **exception:** E_USER_ERROR Orbiter Error 160:1887 E_USER_ERROR Resource location for connection string must be defined

bool **runPreCache** ()

validateContents (line 509)

Compares the Data base response contents and the cache contents.

- **return:** Returns a string message to the user.
- **exception:** E_USER_ERROR Orbiter Error 160:1646 E_USER_ERROR Query and contents have to be defined

string **validateContents** (*array* **$contents**, *array* **$query**)

- *array* **$contents**: Array of contents from Database.
- *array* **$query**: Database query which is canonical string.

Inherited Methods

Inherited From OrbiterService

OrbiterService::__construct()

- **OrbiterService::authenticateServiceRequest()**
- **OrbiterService::checkAttributeValidity()**
- **OrbiterService::checkInterfaceRequest()**
- **OrbiterService::checkOperationValidity()**
- **OrbiterService::defineServiceAttributes()**
- **OrbiterService::defineServiceOperations()**
- **OrbiterService::doServiceRequest()**
- **OrbiterService::endLogServiceRequest()**
- **OrbiterService::getServiceAddress()**
- **OrbiterService::logServiceRequest()**
- **OrbiterService::processAttributes()**
- **OrbiterService::processOperations()**
- **OrbiterService::processServiceAttributes()**
- **OrbiterService::processServiceRequest()**
- **OrbiterService::processServiceResponse()**
- **OrbiterService::setDatabaseConnections()**
- **OrbiterService::setServiceAddress()**
- **OrbiterService::triggerError()**
- **OrbiterService::__destruct()**

OrbiterSimulatorService

Description

Description | Vars (details) | Methods (details)

This is the STAR application simulator class.

Brief example of use:

```
Create an instance of OrbiterSimulatorService
$objService = new OrbiterSimulatorService();

-----------------------------------------------------------------------------------------

Service name
OrbiterSimulatorService.php

-----------------------------------------------------------------------------------------

Service Operation Definitions

Summary Format
operation:
description:
required attributes:
optional attributes:
service method:

-----------------------------------------------------------------------------------------

operation:'api'
description: 'List the service API.'
required attributes: ''
optional attributes: 'format'
service method: 'GET'

operation:'runFile'
description: 'Runs Star sql files.'
required attributes: 'file'
optional attributes: 'format', 'cache', 'noop', 'output', 'detail', 'debug', 'trails'
service method: 'GET'

-----------------------------------------------------------------------------------------

Service Attribute Definitions

Summary Format
attribute:
description:
defaultValue:
allowableValues:

-----------------------------------------------------------------------------------------

attribute: 'operation'
description: 'Specify the type of operation service request required.'
defaultValue: ''
allowableValues: 'api', 'noop', 'runFile'

attribute: 'html'
description: 'Generate HTML compliant output byt replacing escaped character and use
              appropriate tagging and line returns.'
defaultValue: 'off'
allowableValues: 'on', 'off'
```

```
attribute: 'file'
description: 'Specify the absolute file name containing the STAR application database queries.'
defaultValue: ''
allowableValues: ''

attribute: 'database'
description: 'Specify the database name for the query to run.'
defaultValue: ''
allowableValues: ''

attribute: 'debug'
description: 'Output the query and response in addition to the normal statistics when "on".'
defaultValue: 'off'
allowableValues: 'on', 'off'

attribute: 'cache'
description: 'Set to "on" to cache the queries and use cached queries.'
defaultValue: 'on'
allowableValues: 'on', 'off'

attribute: 'detail'
description: 'Set to "on" to output the cumulative query timing.'
defaultValue: 'off'
allowableValues: 'on', 'off'

attribute: 'noop'
description: 'Specify whether to log the service request.'
defaultValue: 'off'
allowableValues: 'on', 'off'

attribute: 'output'
description: 'Set to output the trial queries and not perform them.'
defaultValue: 'off'
allowableValues: 'on', 'off'

attribute: 'address'
description: 'Set to a valid Orbiter network node or basestation with a deployed service.'
defaultValue: 'ORBITERQUERYDBSERVICEADDRESS'
allowableValues: ''

attribute: 'trials'
description: 'Set the number of times to perform trial query set and report the average statistics.'
defaultValue: '1'
allowableValues: ''

attribute: 'format'
description: 'Specify the service response format.'
defaultValue: 'json'
allowableValues: 'json', 'api', 'xml', 'space'

attribute: 'timing'
description: 'Specify whether to log response time in the service log.'
defaultValue: 'off'
allowableValues: 'on', 'off'


View the service schema at ServiceSchema

View the service wsdl 2.0 at ServiceWSDL
```

- **author:** Mark L. Green <mlgreen@txcorp.com>
- **version:** Release: @package_version@
- **copyright:** 2006-2010 Tech-X Corporation. All rights reserved.
- **link:** http://www.txcorp.com/
- **license:** BSD License

*Located in **/OrbiterFederation/classes/OrbiterSimulatorService.class.php** (line **184**)*

```
OrbiterService
   |
   --OrbiterSimulatorService
```

Variable Summary

Description | *Vars* (details) | Methods (details)

- *string* **$_query**

Method Summary

Description | Vars (details) | *Methods* (details)

- *OrbiterSimulatorService* **__construct** ()
- *void* **__destruct** ()
- *void* **endLogServiceRequest** ()
- *void* **processAttributes** ()
- *void* **processOperations** ()
- *void* **processServiceRequest** ()

Variables

Description | Vars (*details*) | Methods (details)

string **$_query** = array() (line 188)

- **var:** query.
- **access:** protected

Inherited Variables

Inherited from OrbiterService

- **OrbiterService::$_auth**
- **OrbiterService::$_cnt**
- **OrbiterService::$_contents**
- **OrbiterService::$_count**
- **OrbiterService::$_dbMaster**
- **OrbiterService::$_dbSlave**
- **OrbiterService::$_definedAttr**
- **OrbiterService::$_errorHandler**
- **OrbiterService::$_logger**
- **OrbiterService::$_phpVariables**
- **OrbiterService::$_queryString**
- **OrbiterService::$_requestMethod**
- **OrbiterService::$_requestUri**
- **OrbiterService::$_scriptName**
- **OrbiterService::$_serviceAddress**
- **OrbiterService::$_serviceSchema**
- **OrbiterService::$_userId**
- **OrbiterService::$_userRole**

Constructor __construct (line 193)

Constructs a new OrbiterSimulatorService.class instance

OrbiterSimulatorService **__construct** ()

Redefinition of:

OrbiterService::__construct()
Constructs a new OrbiterService abstract class instance

Destructor __destruct (line 213)

Destructs the OrbiterSimulatorService class instance

void **__destruct** ()

Redefinition of:

OrbiterService::__destruct()
Destructs a OrbiterService abstract class instance

endLogServiceRequest (line 477)

Ends logging the service request after sending the response status to service log.

- **access:** protected

void **endLogServiceRequest** ()

Redefinition of:

OrbiterService::endLogServiceRequest()
Ends logging the service request.

processAttributes (line 220)

Process the service attributes

- **access:** protected

void **processAttributes** ()

Redefinition of:

OrbiterService::processAttributes()
Abstract function for setting the default attributes the service

processOperations (line 238)

Process the service operations

- **access:** protected

void **processOperations** ()

Redefinition of:

OrbiterService::processOperations()
Abstract function for setting the operations for the service

processServiceRequest (line 375)

Processes the service request based on the operation the user specified

```
---------------------------------------------------------------------------------------

Usage: No required attributes

operation: 'api'
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterSimulatorService.php/operation/api

Response: Returns the api of the service in 'json' format

[{"service":"{Service Name}","attributes":{"operation":{"default":"","restrictions":
[List of Operations]},..List of attributes}, "operations":{"0":[],"operation1":{"GET":
{"description":"{Operation Description}","restrictions":{"required":[List of required attributes],
"optional":[List of optional attributes]}}},..List of Operations}}]


Usage: with optional attribute 'format'

operation: 'api'
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterSimulatorService.php/operation/api/format/api

Response: Returns the api of the service in 'api' format as

array(
     'service'=>{ServiceName},
     'attributes'=>array(
         'operation'=>array(
             'default'=>{Default Value},
             'restrictions'=>array(
                List of Allowable Values
             )
          ),.. List of attributes
     ),
     'operations'=>array(
         'operation1'=>array(
             'method type'=>array(
                 'description'=>{operation description}
                 'restrictions'=>array(
                     'required'=>array(
                         List of required attributes,
                     )
                     'optional'=>array(
                         List of optional attributes
                     )
                  )
              )
         ), .. List of operations
     )
)

---------------------------------------------------------------------------------------

Usage: No required attributes

operation: 'noop' ,
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterSimulatorService.php/operation/noop
```

```
Response: Returns the time taken to process the service request in the format as

[{"Response Time":0.0040218830108643}]


Usage: with optional attribute 'format'

operation: 'noop'
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterSimulatorService.php/operation/noop/format/api

Response: Returns the time taken to process the service request in the api format as

-------------------------------------------------------------------------------------------

Usage: with required attribute 'file'

operation: 'runFile' ,
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterSimulatorService.php/operation/runFile/file//tmp/star.100.sql

Response: Returns the details of 'Number of trials averaged', 'Total number of queries',
'Total size of queries' and 'Total query time' of the given star sql file in the format as

[{"Number of trials averaged":1},{"Total number of queries":36},
{"Total size of queries":17096},{"Total query time":3.1575040817261}]


Usage: with optional attribute 'format'

operation: 'runFile'
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterSimulatorService.php/operation/runFile/file//tmp/star.100.sql/format/api

Response: Returns the details of 'Number of trials averaged', 'Total number of queries',
'Total size of queries' and 'Total query time' of the given star sql file in the api format as

Array
(
    [0] => Array
         (
            [Number of trials averaged] => 1
         )
    [1] => Array
         (
            [Total number of queries] => 36
         )
    [2] => Array
         (
            [Total size of queries] => 32769
         )
    [3] => Array
         (
            [Total query time] => 2.6990029811859
         )
)

-------------------------------------------------------------------------------------------
```

- **exception:** E_USER_ERROR Orbiter Error 155:1618 E_USER_ERROR Improper use statement '{line}'
- **exception:** E_USER_ERROR Orbiter Error 155:1619 E_USER_ERROR The file is not readable or does not exist '{file}'

void **processServiceRequest** ()

Redefinition of:

OrbiterService::processServiceRequest()

Abstract function for the business logic for the service request

Inherited Methods

Inherited From OrbiterService

- **OrbiterService::__construct()**
- **OrbiterService::authenticateServiceRequest()**
- **OrbiterService::checkAttributeValidity()**
- **OrbiterService::checkInterfaceRequest()**
- **OrbiterService::checkOperationValidity()**
- **OrbiterService::defineServiceAttributes()**
- **OrbiterService::defineServiceOperations()**
- **OrbiterService::doServiceRequest()**
- **OrbiterService::endLogServiceRequest()**
- **OrbiterService::getServiceAddress()**
- **OrbiterService::logServiceRequest()**
- **OrbiterService::processAttributes()**
- **OrbiterService::processOperations()**
- **OrbiterService::processServiceAttributes()**
- **OrbiterService::processServiceRequest()**
- **OrbiterService::processServiceResponse()**
- **OrbiterService::setDatabaseConnections()**
- **OrbiterService::setServiceAddress()**
- **OrbiterService::triggerError()**
- **OrbiterService::__destruct()**

OrbiterVersionInformationService

Description

This is the Orbiter Connectivity service class.

Brief example of use:

```
Create an instance of OrbiterVersionInformationService
$objService = new OrbiterVersionInformationService;

---------------------------------------------------------------------------------------------

Service name
OrbiterVersionInformationService.php

---------------------------------------------------------------------------------------------

Service Operation Definitions

Summary Format
operation:
description:
required attributes:
optional attributes:
service method:

---------------------------------------------------------------------------------------------

operation:'api'
description: 'List the service API.'
required attributes: ''
optional attributes: 'format'
service method: 'GET'

operation:'getVersionInfo'
description: 'Retrieve and report the Orbiter Version information for all Orbiter versions known in orbiter_versions.'
required attributes: ''
optional attributes: 'versionName', 'versionNumber','format'
service method: 'GET'

operation:'getVersionLogo'
description: 'Retrieve and report the Orbiter Version information for all Orbiter versions known in orbiter_versions.'
required attributes: 'versionLogo'
optional attributes: 'format','html','thumbs'
service method: 'GET'

---------------------------------------------------------------------------------------------

Service Attribute Definitions

Summary Format
attribute:
description:
defaultValue:
allowableValues:

---------------------------------------------------------------------------------------------

attribute: 'operation'
description: 'Specify the type of operation service request required.'
defaultValue: ''
allowableValues: 'api', 'getVersionInfo', 'getVersionLogo'

attribute: 'html'
description: 'Generate HTML compliant output byt replacing escaped character and use
              appropriate tagging and line returns.'
defaultValue: 'off'
allowableValues: 'on', 'off'

attribute: 'format'
description: 'Specify the service response format.'
defaultValue: 'json'
allowableValues: 'json', 'api', 'schema'

attribute: 'versionName'
description: 'Specify the orbiter version name to report information on.'
defaultValue: ''
allowableValues: ''

attribute: 'versionNumber'
description: 'Specify the orbiter version number to report information on.'
defaultValue: ''
allowableValues: ''

attribute: 'versionLogo'
description: 'logo attribute for the version number.'
defaultValue: ''
```

```
allowableValues: ''
attribute: 'thumbs'
description: 'when set to "on" the image thumbnail is returned'
defaultValue: 'off'
allowableValues: 'on', 'off'

attribute: 'thumbsWidth'
description: 'when set the thumbnail image widths are set to this number of pixels.'
defaultValue: '64'
allowableValues: '64'


View the service schema at ServiceSchema

View the service wsdl 2.0 at ServiceWSDL
```

- **author:** Mark L. Green <mlgreen@txcorp.com>
- **version:** Release: @package_version@
- **copyright:** 2006-2011 Tech-X Corporation. All rights reserved.
- **link:** Tech-X Corporation
- **license:** BSD License

*Located in **/OrbiterFederation/classes/OrbiterVersionInformationService.class.php** (line **163**)*

```
OrbiterService
   |
   --OrbiterVersionInformationService
```

Method Summary

Description | Vars | *Methods* (details)

- *OrbiterVersionInformationService* **__construct** ()
- *void* **__destruct** ()
- *void* **getVersionInfo** ()
- *void* **getVersionLogo** ()
- *void* **normalImage** (*string* **$image**)
- *void* **processAttributes** ()
- *void* **processOperations** ()
- *void* **processServiceRequest** ()
- *void* **queryResults** (**$query**)
- *void* **thumbnail** (*string* **$image**)

Variables

Description | Vars (*details*) | Methods (details)

Inherited Variables

Inherited from OrbiterService

- **OrbiterService::$_auth**
- **OrbiterService::$_cnt**
- **OrbiterService::$_contents**
- **OrbiterService::$_count**
- **OrbiterService::$_dbMaster**
- **OrbiterService::$_dbSlave**
- **OrbiterService::$_definedAttr**
- **OrbiterService::$_errorHandler**
- **OrbiterService::$_logger**
- **OrbiterService::$_phpVariables**
- **OrbiterService::$_queryString**
- **OrbiterService::$_requestMethod**
- **OrbiterService::$_requestUri**
- **OrbiterService::$_scriptName**
- **OrbiterService::$_serviceAddress**
- **OrbiterService::$_serviceSchema**
- **OrbiterService::$_userId**
- **OrbiterService::$_userRole**

Methods

Description | Vars Methods (*details*)

Constructor __construct (line 171)

Constructs a new OrbiterVersionInformationService.class instance

OrbiterVersionInformationService **__construct** ()

Redefinition of:

OrbiterService::__construct()
Constructs a new OrbiterService abstract class instance

Destructor __destruct (line 188)

Destructs the OrbiterVersionInformationService.class instance

void **__destruct** ()

Redefinition of:

OrbiterService::__destruct()
Destructs a OrbiterService abstract class instance

getVersionInfo (line 400)

Creates the query string for the Orbiter Version inforamtion based on attributes used by operations

- **access:** protected

void **getVersionInfo** ()

getVersionLogo (line 439)

Creates the query string for the Orbiter Version logo based on attributes used by operations

- **exception:** E_USER_ERROR Orbiter Error 185:1906 E_USER_ERROR Database query error
- **access:** protected

void **getVersionLogo** ()

normalImage (line 491)

Generate a normal sized image for the select image.

void **normalImage** (*string* **$image**)

- *string* **$image**

processAttributes (line 194)

Process the service attributes

- **access:** protected

void **processAttributes** ()

Redefinition of:

OrbiterService::processAttributes()
Abstract function for setting the default attributes the service

processOperations (line 207)

Process the service operations

- **access:** protected

void **processOperations** ()

Redefinition of:

OrbiterService::processOperations()
Abstract function for setting the operations for the service

processServiceRequest (line 381)

Processes the service request based on the operation the user specified

```
-------------------------------------------------------------------------------------------
Usage: No required attributes

operation: 'api'
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterVersionInformationService.php/operation/api

Response: Returns the api of the service in json format as

array(
     'service'=>{ServiceName},
     'attributes'=>array(
```

```
        'operation'=>array(
            'default'=>{Default value},
            'restrictions'=>array(
               List of Allowable Values
            )
         ),.. List of attributes
    ),
    'operations'=>array(
        'operation1'=>array(
            'method type'=>array(
                'description'=>{operation description}
                'restrictions'=>array(
                    'required'=>array(
                        List of required attributes,
                    )
                    'optional'=>array(
                        List of optional attributes
                    )
                 )
             )
        ), .. List of operations
    )
)
```

Usage: with optional attribute 'format'

operation: 'api'
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterVersionInformationService.php/operation/api/format/xml

Response: Returns the api of the service in xml format similar to json

---

Usage: No required attributes

operation: 'getVersionInfo'
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterVersionInformationService.php/operation/getVersionInfo

Response: Returns all of the version information for every release of Orbiter in the following format:

[{"id":"1","name":"apollo","version":"1","image_Uri":"http:\/\/txc02.ccr.buffalo.edu\/orbiter\/sbdev\/service\
/webservice\/OrbiterVersionInformationService.php\/operation\/getVersionLogo\/html\/on\/versionLogo\/1\/format\
/text","thumb_Uri":"http:\/\/txc02.ccr.buffalo.edu\/orbiter\/sbdev\/service\/webservice\/OrbiterVersionInformationServi
/operation\/getVersionLogo\/html\/off\/versionLogo\/1\/format\/text\/thumbs\/on","release":"2011-01-15 14:12:52"},

where
'id' denotes Orbiter version id
'name' denotes Orbiter version name
'version' denotes Orbiter version number
'image_Uri' denotes Orbiter version Uri of release logo
'thumb_Uri' denotes Orbiter version Uri of release logo in a smaller, thumbnail size
'release' denotes Orbiter version release timestamp

Usage: With optional attribute 'format'

operation: 'getVersionInfo'
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterVersionInformationService.php/operation/getVersionInfo/format/xml

Response: Returns all of the version information for every release of Orbiter in xml format

Usage: With optional attribute 'versionNumber'

operation: 'getVersionInfo'
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterVersionInformationService.php/operation/getVersionInfo/versionNumber/1

Response: Returns all of the version information for every release of Orbiter in the following format:

[{"id":"1","name":"apollo","version":"1","image_Uri":"http:\/\/txc02.ccr.buffalo.edu\/orbiter\/sbdev\/service\/webservi
/OrbiterVersionInformationService.php\/operation\/getVersionLogo\/html\/on\/versionLogo\/1\/format\/text","thumb_Uri":"
\/txc02.ccr.buffalo.edu\/orbiter\/sbdev\/service\/webservice\/OrbiterVersionInformationService.php\/operation\/getVersi
html\/off\/versionLogo\/1\/format\/text\/thumbs\/on","release":"2011-01-15 14:12:52"}]

where
'id' denotes Orbiter version id
'name' denotes Orbiter version name
'version' denotes Orbiter version number
'image_Uri' denotes Orbiter version Uri of release logo
'thumb_Uri' denotes Orbiter version Uri of release logo in a smaller, thumbnail size
'release' denotes Orbiter version release timestamp

Usage: With optional attribute 'versionName'

operation: 'getVersionInfo'
URI:/orbiter/ORBITERVERSION/service/webservice/

```
OrbiterVersionInformationService.php/operation/getVersionInfo/versionName/apollo
Response: Returns all of the version information for every release of Orbiter in the following format:

[{"id":"1","name":"apollo","version":"1","image_Uri":"http:\/\/txc02.ccr.buffalo.edu\/orbiter\/sbdev\/service\/webservi
/OrbiterVersionInformationService.php\/operation\/getVersionLogo\/html\/on\/versionLogo\/1\/format\/text","thumb_Uri":"
\/txc02.ccr.buffalo.edu\/orbiter\/sbdev\/service\/webservice\/OrbiterVersionInformationService.php\/operation\/getVersi
html\/off\/versionLogo\/1\/format\/text\/thumbs\/on","release":"2011-01-15 14:12:52"}]

where
'id' denotes Orbiter version id
'name' denotes Orbiter version name
'version' denotes Orbiter version number
'image_Uri' denotes Orbiter version Uri of release logo
'thumb_Uri' denotes Orbiter version Uri of release logo in a smaller, thumbnail size
'release' denotes Orbiter version release timestamp


-------------------------------------------------------------------------------------------

Usage: With required attribute 'versionLogo'

operation: 'getVersionLogo'
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterVersionInformationService.php/operation/getVersionLogo/versionLogo/1

Response: Returns the version logo binary image data for the release number specified by versionLogo.


Usage: With optional attribute 'html'

operation: 'getVersionLogo'
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterVersionInformationService.php/operation/getVersionLogo/versionLogo/1/html/on/

Response: Returns the URI of the image data for the release number specified by versionLogo in the format as

"[?]"


Usage: With optional attributes 'html' and 'format'

operation: 'getVersionLogo'
URI:/orbiter/ORBITERVERSION/service/webservice/
OrbiterVersionInformationService.php/operation/getVersionLogo/versionLogo/1/html/on/format/text

Response: Returns the URI of the image data for the release number specified by versionLogo in a plain text format simi


-------------------------------------------------------------------------------------------
```

- **access:** protected

void **processServiceRequest** ()

Redefinition of:

OrbiterService::processServiceRequest()
Abstract function for the business logic for the service request

queryResults (line 419)

Queries the database, fetches the assoc array and displays results

- **exception:** E_USER_ERROR Orbiter Error 185:1905 E_USER_ERROR Database query error
- **access:** protected

void **queryResults** (**$query**)

- **$query**

thumbnail (line 470)

Generate a thumbnail image for the select image.

void **thumbnail** (*string* **$image**)

- *string* **$image**

Inherited Methods

Inherited From OrbiterService

- **OrbiterService::__construct()**
- **OrbiterService::authenticateServiceRequest()**
- **OrbiterService::checkAttributeValidity()**
- **OrbiterService::checkInterfaceRequest()**

- OrbiterService::checkOperationValidity()
- OrbiterService::defineServiceAttributes()
- OrbiterService::defineServiceOperations()
- OrbiterService::doServiceRequest()
- OrbiterService::endLogServiceRequest()
- OrbiterService::getServiceAddress()
- OrbiterService::logServiceRequest()
- OrbiterService::processAttributes()
- OrbiterService::processOperations()
- OrbiterService::processServiceAttributes()
- OrbiterService::processServiceRequest()
- OrbiterService::processServiceResponse()
- OrbiterService::setDatabaseConnections()
- OrbiterService::setServiceAddress()
- OrbiterService::triggerError()
- OrbiterService::__destruct()

4.2 Commander Documentation

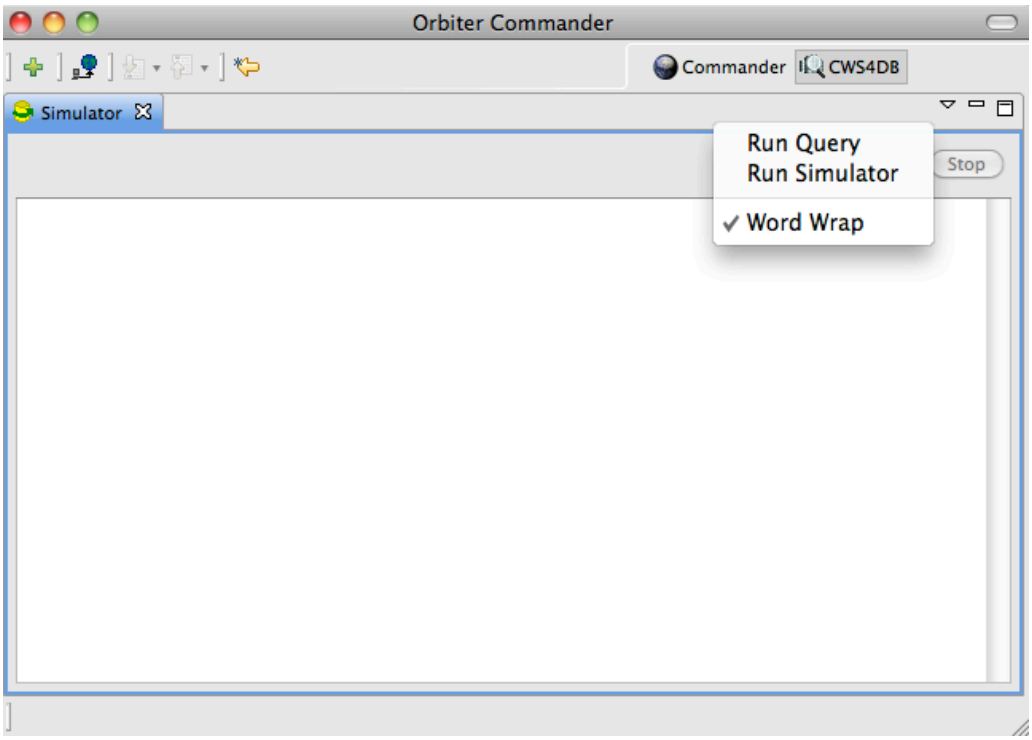
The following pages include the user help documentation for the Orbiter Commander CWS4DB Suite modules.

Query Module Guide

Query Module is an Orbiter Commander Module for querying and simulating queries to resource endpoints of remote resources. Use this module to execute prepared queries to test the performance of resources.

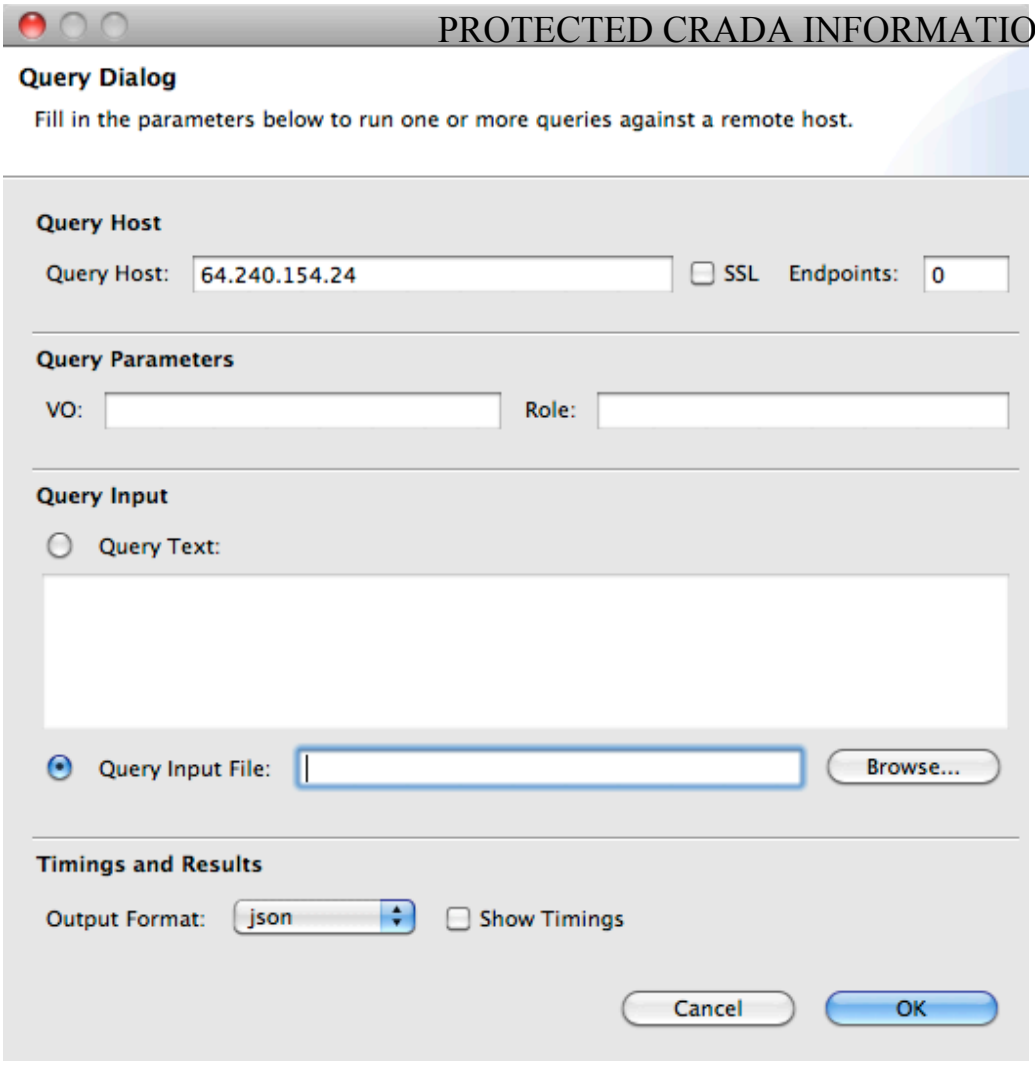
Click on the downward arrow on the right of 'Query Module' to find the options of Run Query, Run Simulator and Word Wrap.

The screen capture of 'Query Module' with its features is shown below



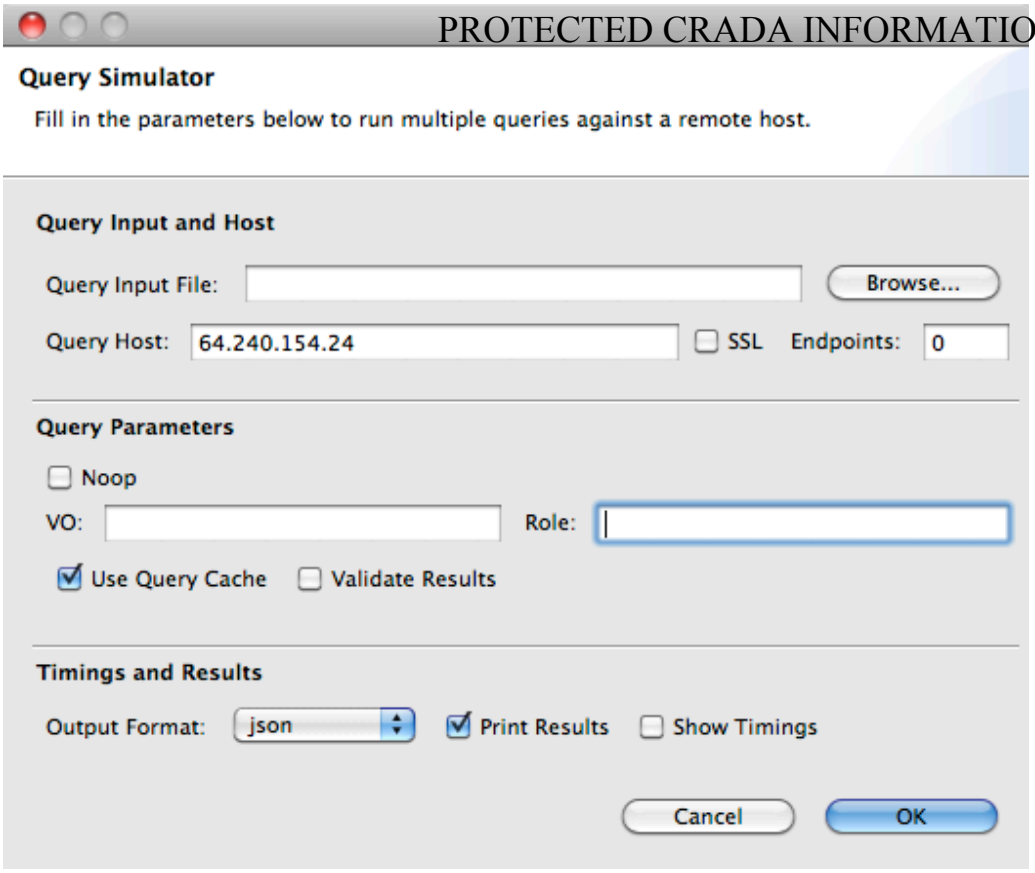
Clicking on the '***Run Query***' option opens a 'Query Dialog' prompting you to specify the Query Host in '***Query Host:***' text-box, whether SSL is used by checking the check-box '***SSL***', Endpoints in text-box '***Endpoints:***', Query Parameters VO in text-box '***VO:***' and Role in text-box '***Role:***', Query Input as Query Text by selecting the radio-button '***Query Text:***' and enter the query text in the space below the radio button or Query Input File by selecting the radio-button '***Query Input File:***' and entering the file name or browing the file by clicking on the '***Browse:***' button, choose output format for results from the drop-down '***Output Format:***' and whether to show timings of the results by checking the check-box '***Show Timings***'. Click on the '***OK***' button to run the query once all the parameters are filled.

The screen capture of 'Query Dialog' is shown below



Clicking on the '***Run Simulator***' option opens a 'Query Simulator' prompting you to specify the Query Input File in the text-box '***Query Input File:***' by entering the file name or browing the file by clicking on the '***Browse:***' button, Query Host in '***Query Host:***' text-box, whether SSL is used by checking the check-box '***SSL***', Endpoints in text-box '***Endpoints:***', Query Parameters VO in text-box '***VO:***', Role in text-box '***Role:***' , whether to run noop by checking the check-box '***Noop***' , whether to use query cache by checking the check-box '***Query Cache***', whether to validate results by checking the check-box '***Validate Results***', choose output format for results from the drop-down '***Output Format:***', whether to print results by checking the check-box '***Print Results***' and whether to show timings of the results by checking the check-box '***Show Timings***'. Click on the '***OK***' button to simulate the query once all the parameters are filled.

The screen capture of 'Query Simulator' is shown below



Clicking on the '***Word Wrap***' option wraps the text that appears in the Simulator.

Clicking on the '***Cancel***' button cancels the window.

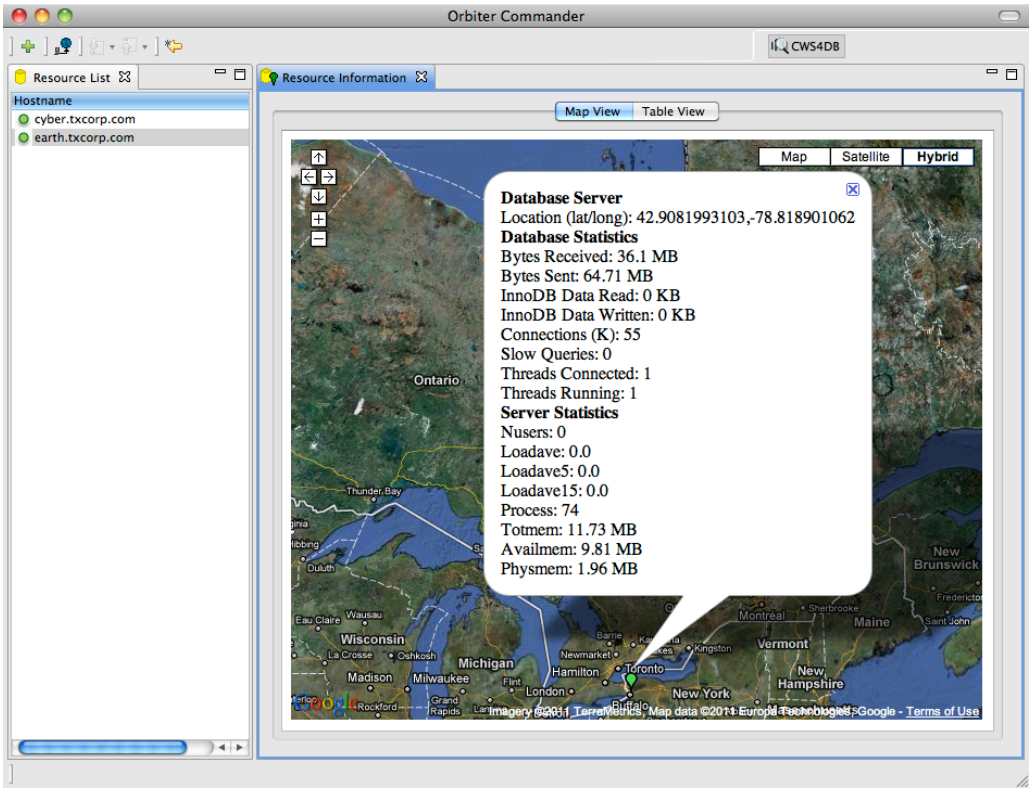
Resource Monitor Module Guide

Resource Monitor is an Orbiter Commander Module for viewing the status of resources.

It displays the resource statistics in two different views as shown below

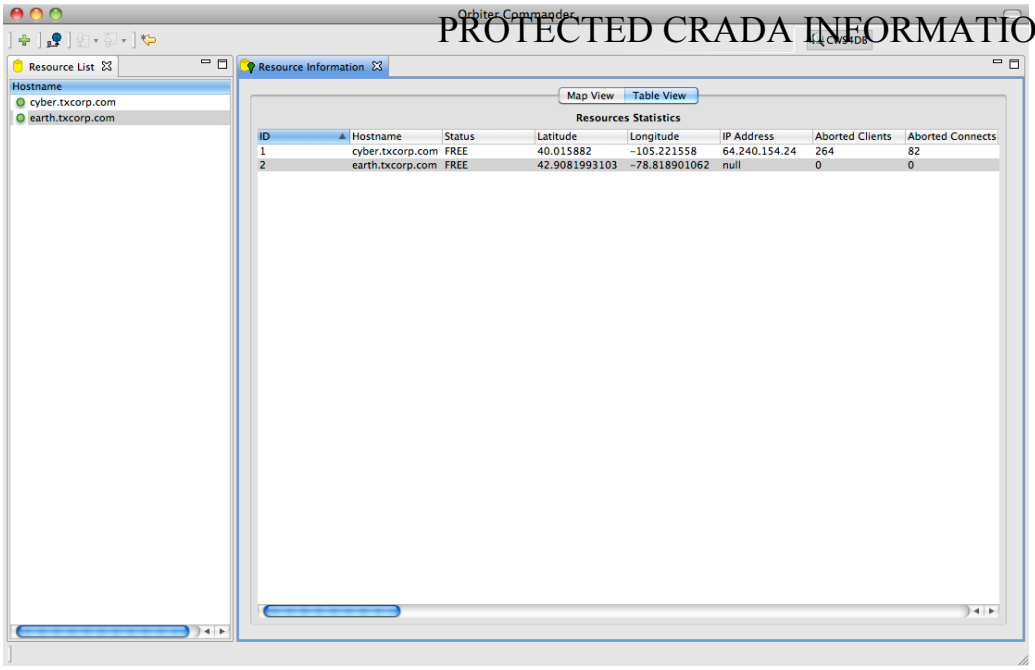
- ***Map View***: Maps the host location to the Google Map and pins it accordingly. When clicked on any of these pinned locations, it displays the location (latitude/longitude) and database statistics like bytes received, bytes sent, InnoDB Data Read, InnoDB Data Written, Connections, Slow Queries, Threads Connected, Threads running and server statistics which include Number of Users, Loadaverage, Loadaverage5, Loadaverage15, process, total memory, available memory and physical memory.

A screen capture of 'Map View' is shown below



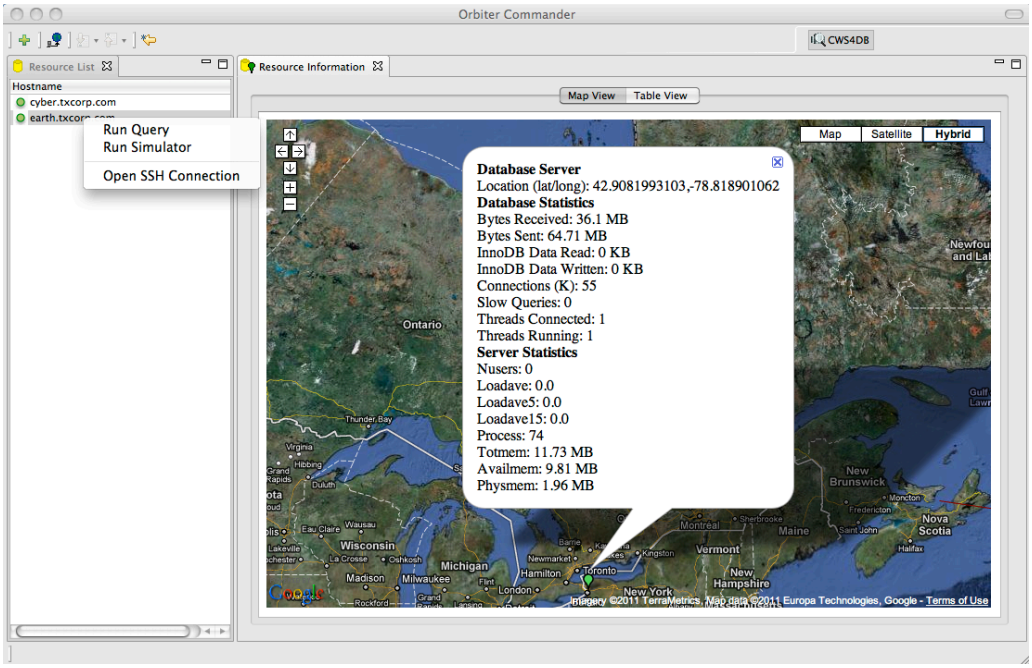
- ***Table View***: Displays the resource statistics of status, latitude, longitude, IP Address, Aborted Clients, Aborted Connects, Bytes Received, Bytes Sent, COM Insert, COM Select, COM Update, Connections, InnoDB Data Read, InnoDB Data Written, Threads Cached, Threads Connected, Threads Created, Threads Running, Slow Queries, Uptime, Number of Users, Load Average, Load Average(5), Load Average(15), Number of Processes, Total Memory, Available Memory, Physical Memory, Number of Processors, Vendor ID, CPU Family, Model Number, Model Name, CPU MHz, Cache Size, Timestamp, Timezone and Grid ID for each host.

A screen capture of 'Table View' is shown below



Interoperability:
You can perform the functions of other installed modules like 'Run Query','Run Simulator' and 'Open SSH Connection' from Resource Monitor by right-clicking on a host name if the modules are installed for you. Refer to 'Query Module' guide to know more about 'Run Query','Run Simulator' features if it is installed for you. Refer to 'Query SSH Module' guide to know more about 'Open SSH Connection' feature if it is installed for you.

A screen capture of 'Resource Monitor' with features of Query Module and Query SSH Module being displayed upon right-click on a host name is shown below

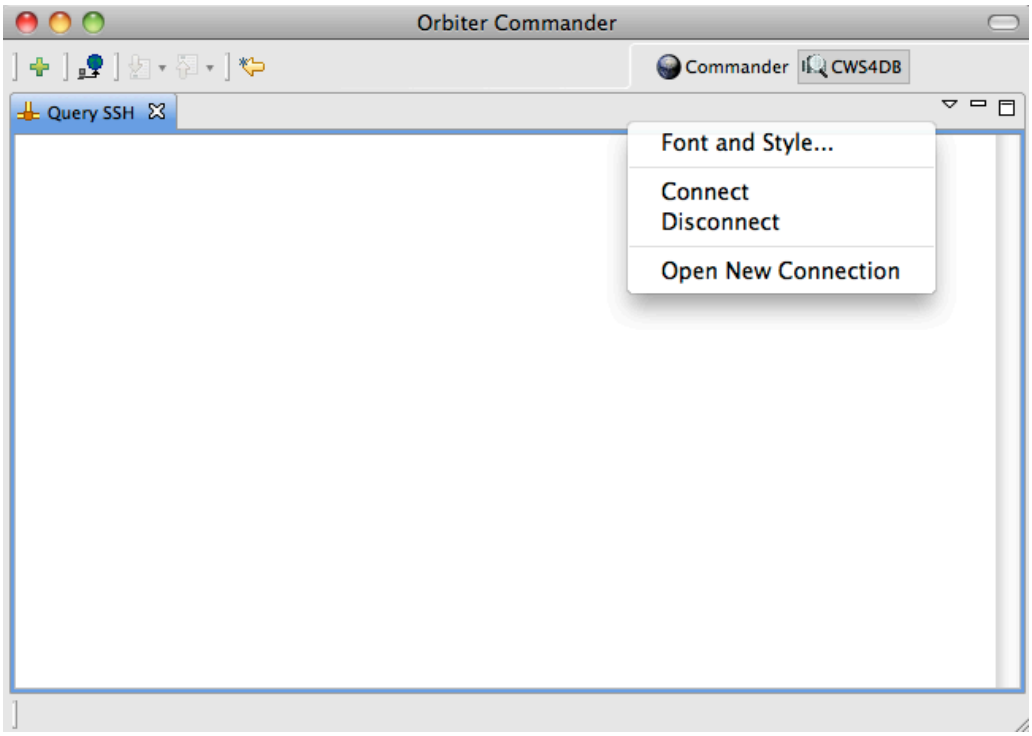


Query SSH Module Guide

Query SSH Module is an Orbiter Commander Module for initiating SSH connections to remote query resources. It is similar of initiating a SSH connection from a terminal window.

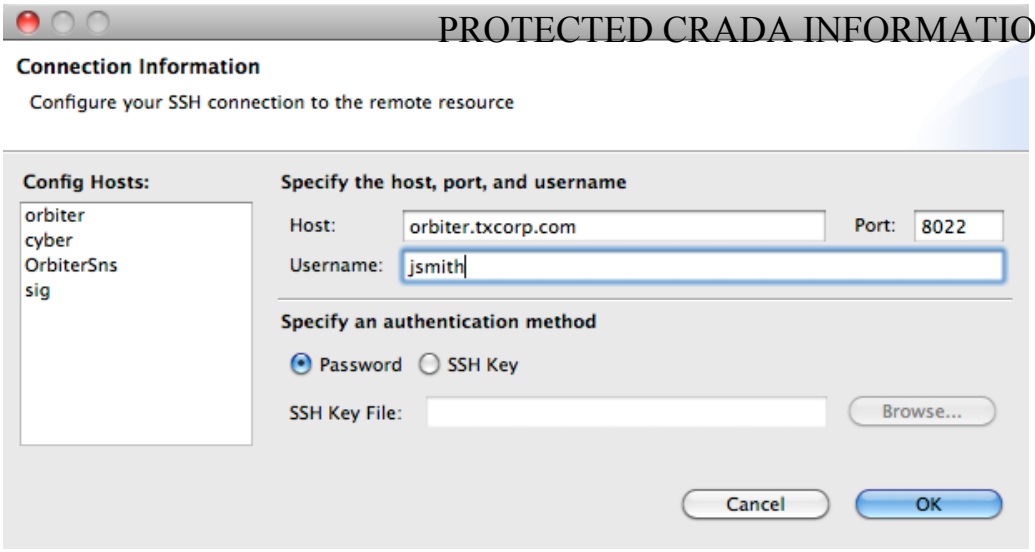
Click on the downward arrow on the right of 'Query SSH Module' to find the options of Font and Style, Connect, Disconnect, Open New Connection.

The screen capture of 'Query SSH Module' with its features is shown below



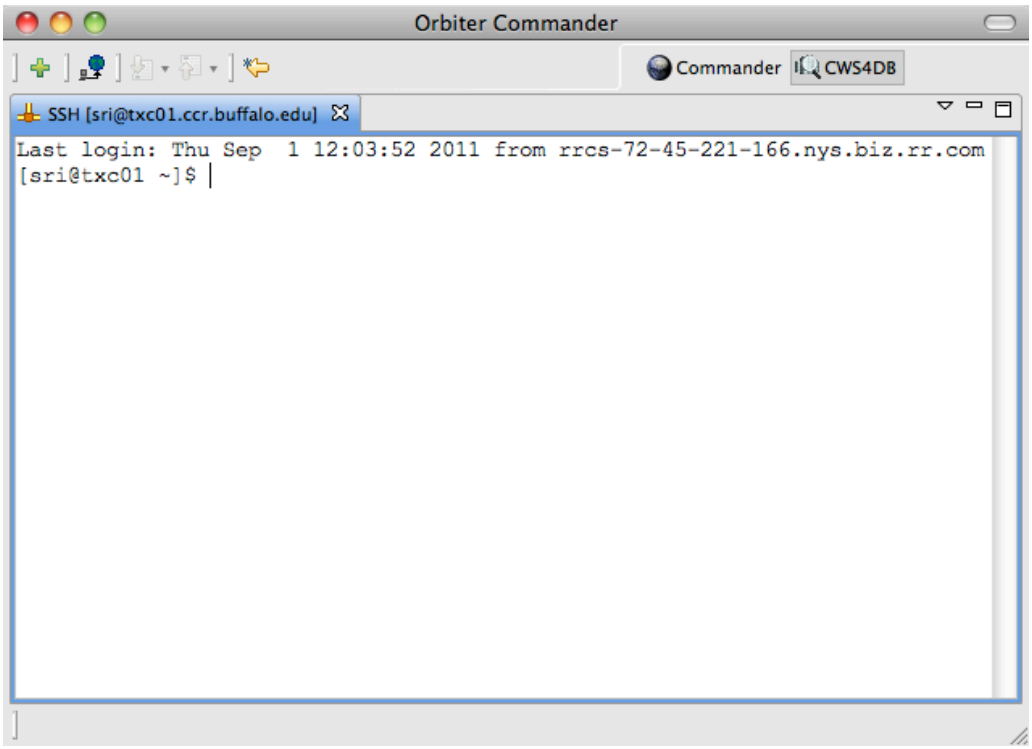
Clicking on the '***Connect***' option opens a 'Connection Information' dialog prompting you to specify the host in text-box '***Host:***', port in text-box '***Port:***', username in text-box '***Username:***' and authentication method by choosing the radio button '***Password***' or SSH Key by choosing radio button '***SSH Key***' and enter the SSH key in text-box '***SSH Key File:***' or browsing the file by clicking on the '***Browse***' button. Click on the '***OK***' button to connect to the host once all the parameters are filled.

The screen capture of 'Connection Information' dialog with its features is shown below



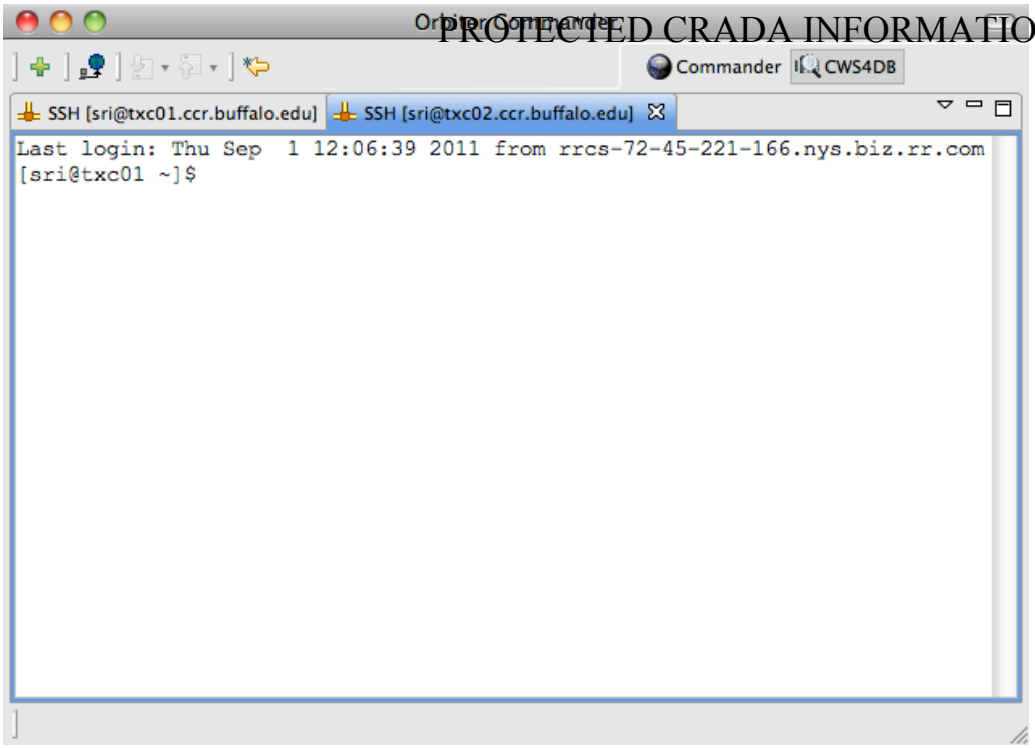
A screen capture of 'Connection Information' dialog with SSH Key chosen as authentication method is shown below

A screen capture of 'Query SSH Module' dialog when connected to the host is shown below



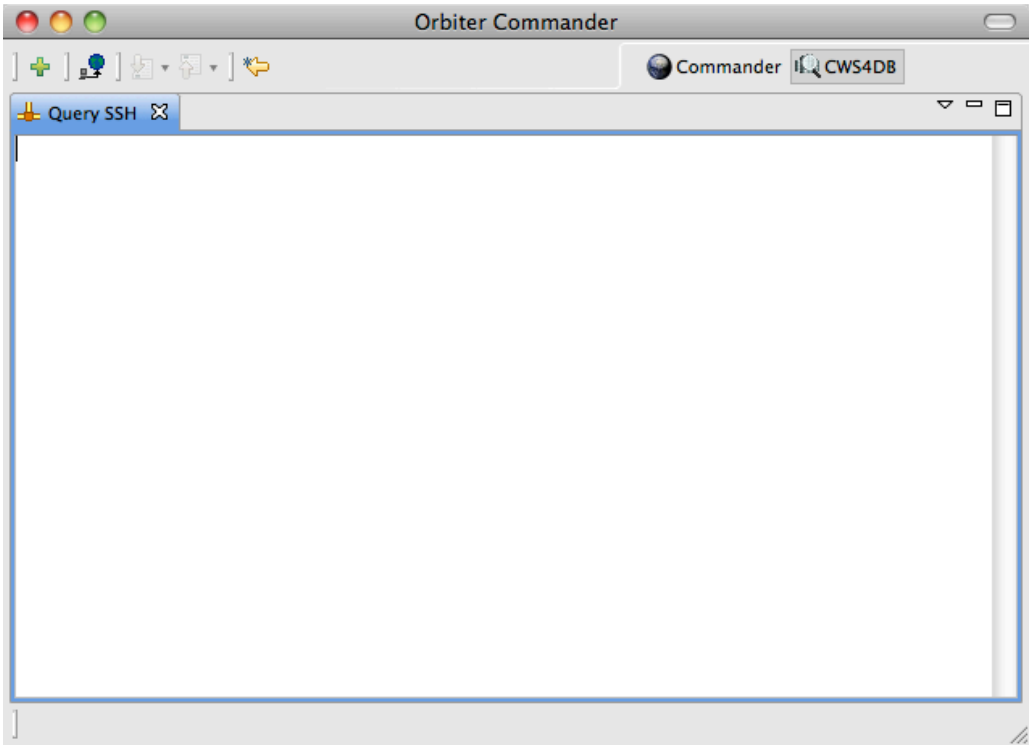
You can open a new SSH connection to a host by clicking on the '***Open New Connection***' option upon clicking the downward arrow on the right of 'Query SSH Module'.

The process is similar to connecting to a host which is explained above. A screen capture of 'Query SSH Module' dialog when a new connection is opened is shown below



You can disconnect the Query SSH Module from the host by clicking on the '***Disconnect***' option upon clicking the downward arrow on the right of 'Query SSH Module'.

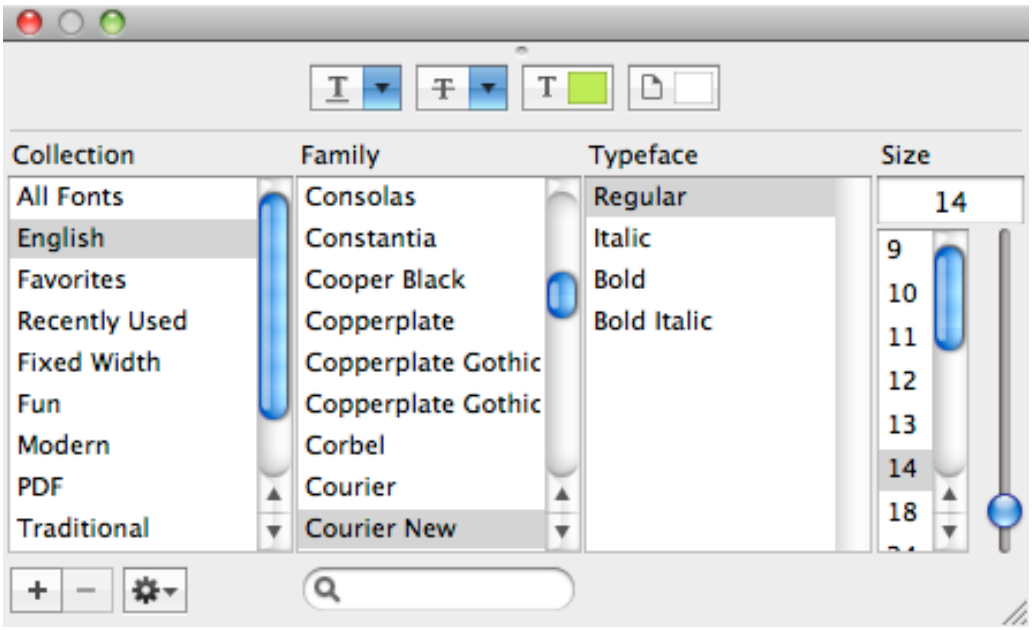
A screen capture of 'Query SSH Module' dialog when disconnected from the host is shown below



The Query SSH Module throws an error when it cannot connect to the host with the provided connection information. A screen capture of 'Query SSH Module' dialog when it could not connect to the host is shown below

Clicking on the '***Font and Style***' option opens a 'Font and Style' dialog allowing you to modify the font, style and size of the text in SSH interface.

A screen capture of 'Font and Style' dialog with its features is shown below

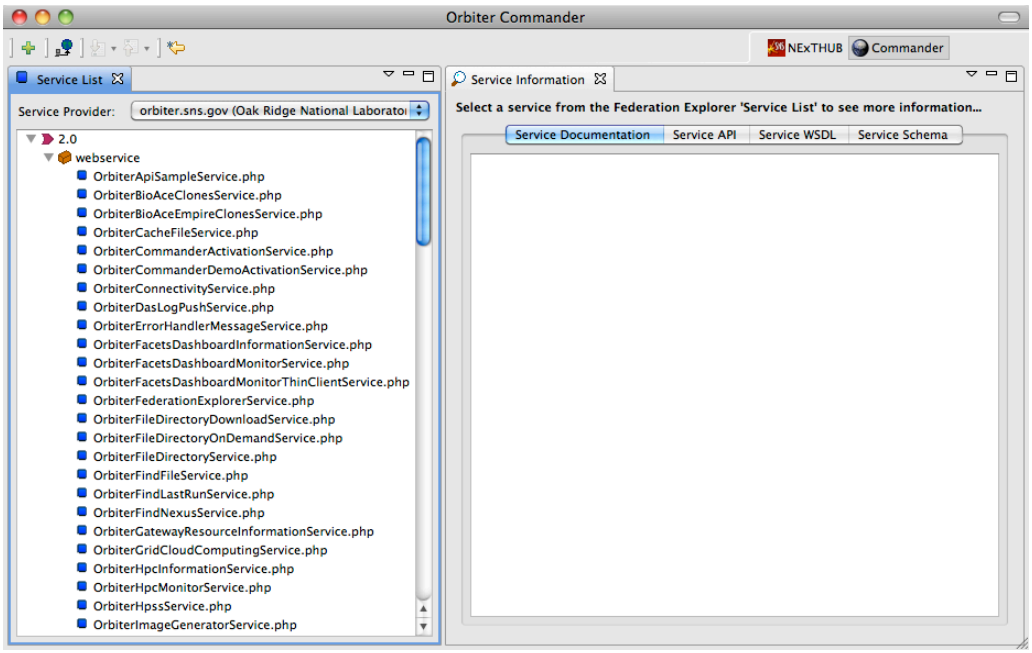


Clicking on the '***Cancel***' button cancels the window.

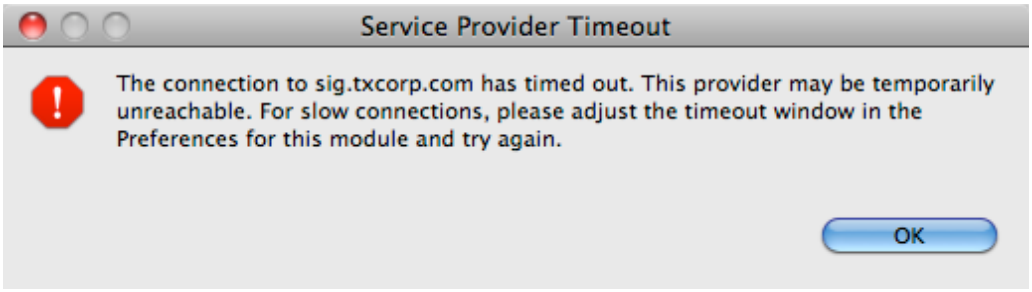
Orbiter Federation Explorer Module Guide

Orbiter Federation Explorer is an Orbiter Commander Module for exploring the services provided by Orbiter Federation.

It presents a service list for each of the chosen service provider from the 'Service Provider' drop-down list on the left hand side. The services listed are grouped by version as shown in the screen capture below.

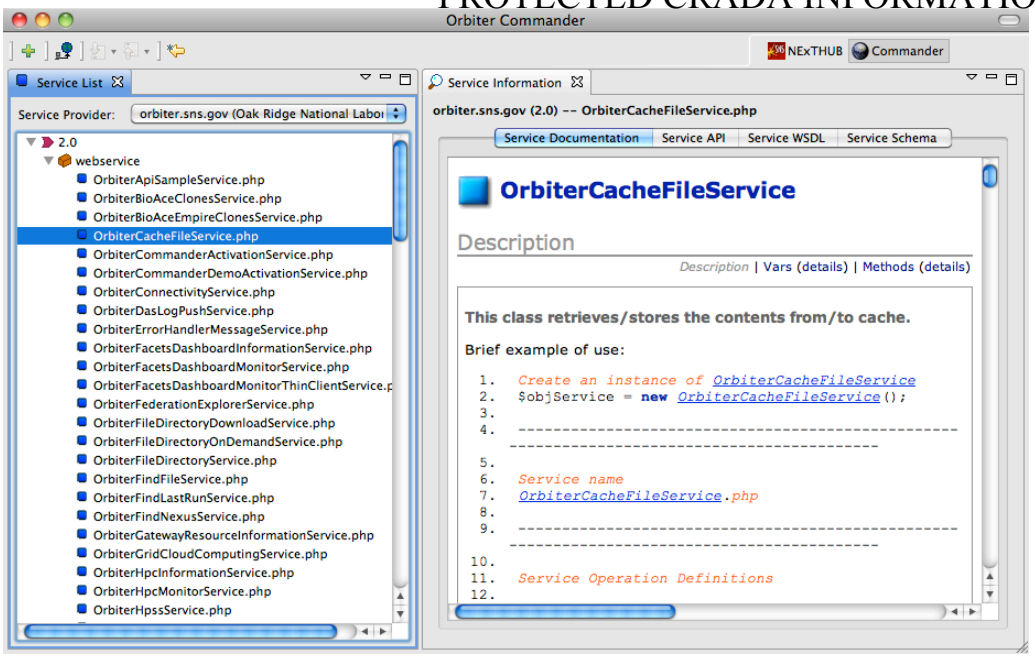


If an unreachable or temporarily unavailable provider is selected from the providers list in 'Service Provider' drop-down it throws an error as shown in the following screen capture.

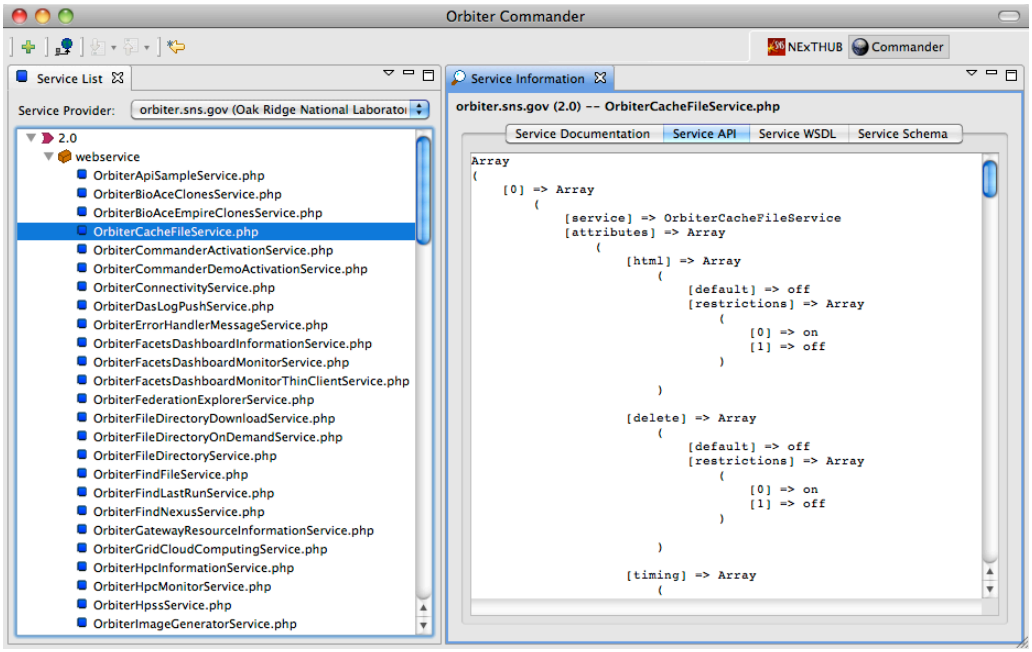


You can click on any of the services and view the Service Information consisting of Service Documentation, Service API, Service WSDL and Service Schema on the right. You can also test a service by clicking on the 'Test Service' option which is found by clicking the 'downward arrow' icon on the upper right hand side. Testing the service is explained in Testing a Service.

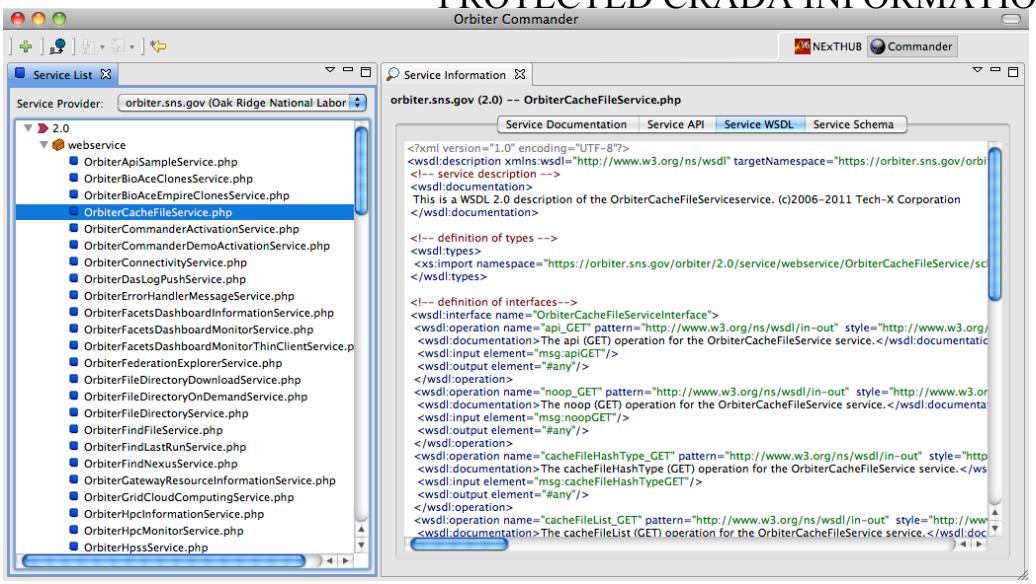
Service Documentation in Orbiter Commander displays details of service description, method summary, variables including inherited, method description, method input parameters, method returns, service attributes, service operations and triggered errors in 'Earthli' template. You can also drill down to source code, inherited methods, parent classes in the hierarchy tree. A screen capture of 'Service Documentation' is shown below



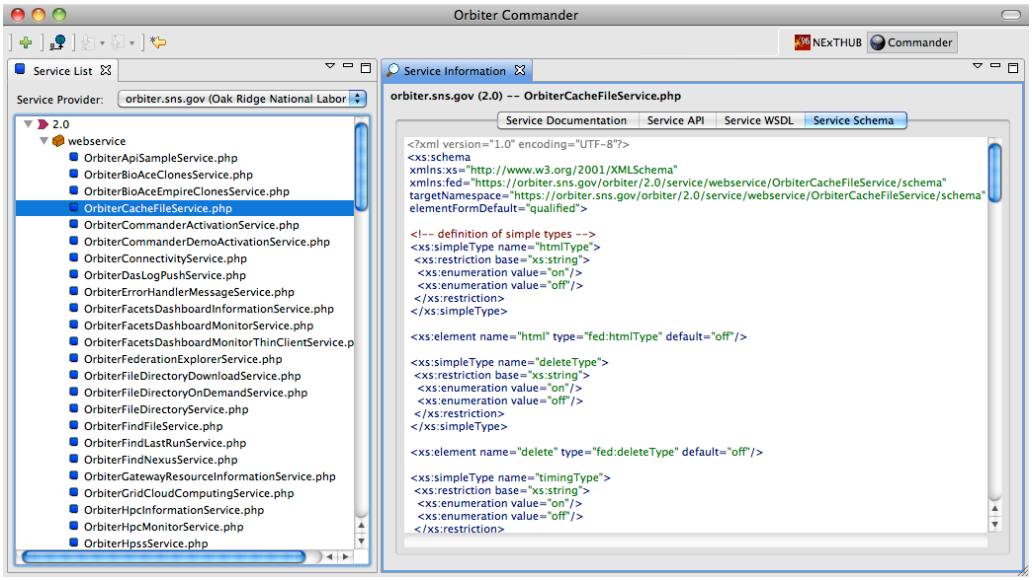
Service API are a particular set of rules and specifications that can be followed to access and make use of services and resources provided by the service. Service API in Orbiter Commander displays an array of service attribute details, service operation details of the Service. A screen capture of 'Service API' is shown below



Service WSDL stands for Web Services Description Language(WSDL). WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate. A screen capture of 'Service WSDL' is shown below



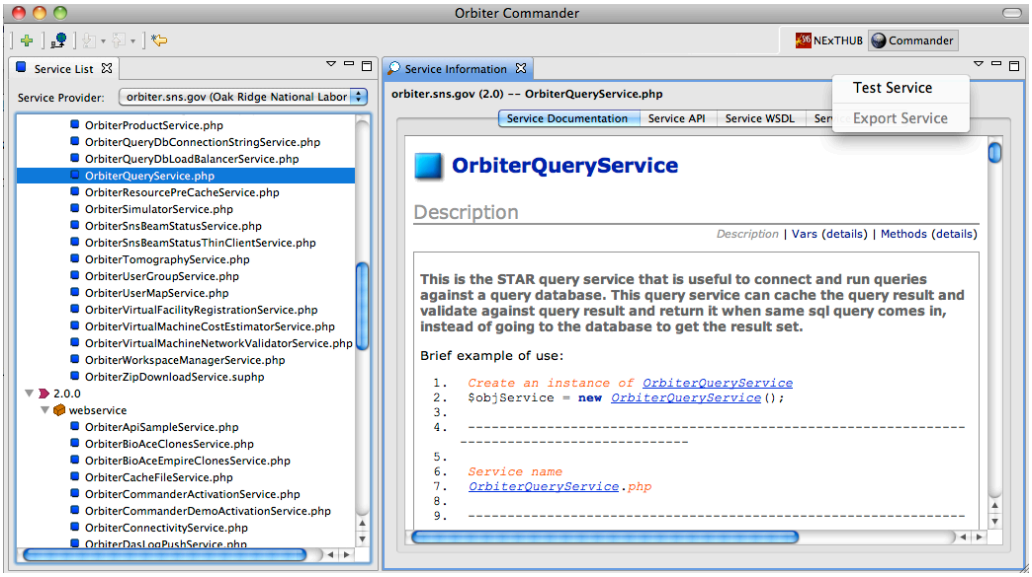
Service Schema is used to specify the structure of instance documents and the datatype of each element/attribute. Schemas provide a means of defining the structure, content and semantics of elements in the documents which can be shared between different types of computers and documents. Our service schema provides detail about input attributes accepted by service operations. A screen capture of 'Service Schema' is shown below



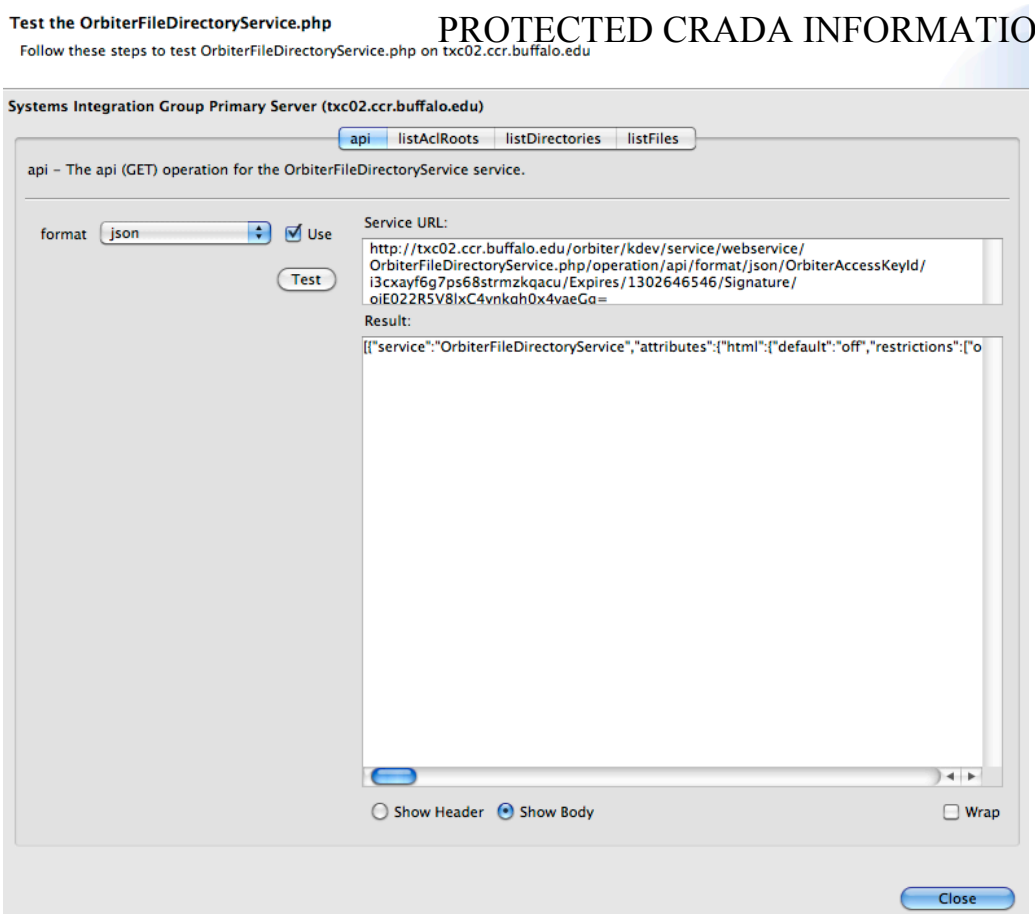
Orbiter Federation Explorer Module Guide

Orbiter Federation Explorer lets you test a service by clicking on the 'Test Service' option which is found by clicking the 'downward arrow' icon on the upper right hand side.

A screen capture of 'Test Service' option is shown below



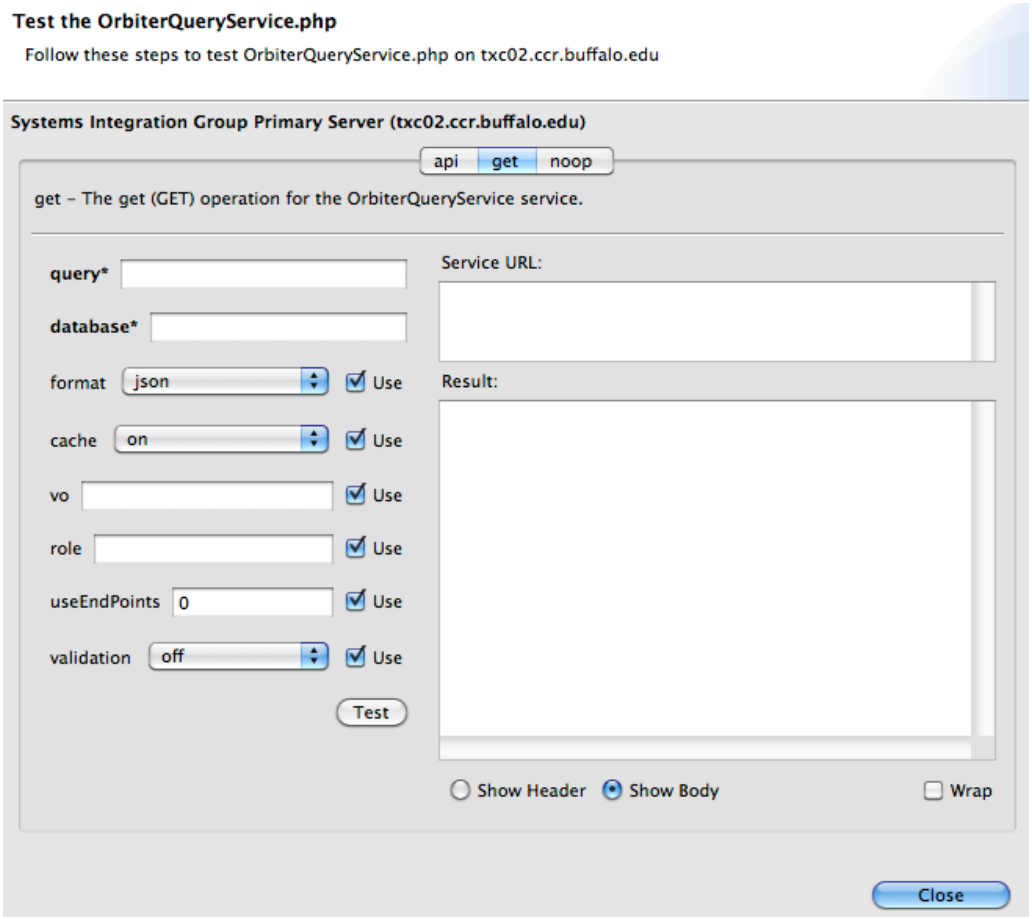
It then opens a 'Test the web service' window , a screen capture of which is shown below



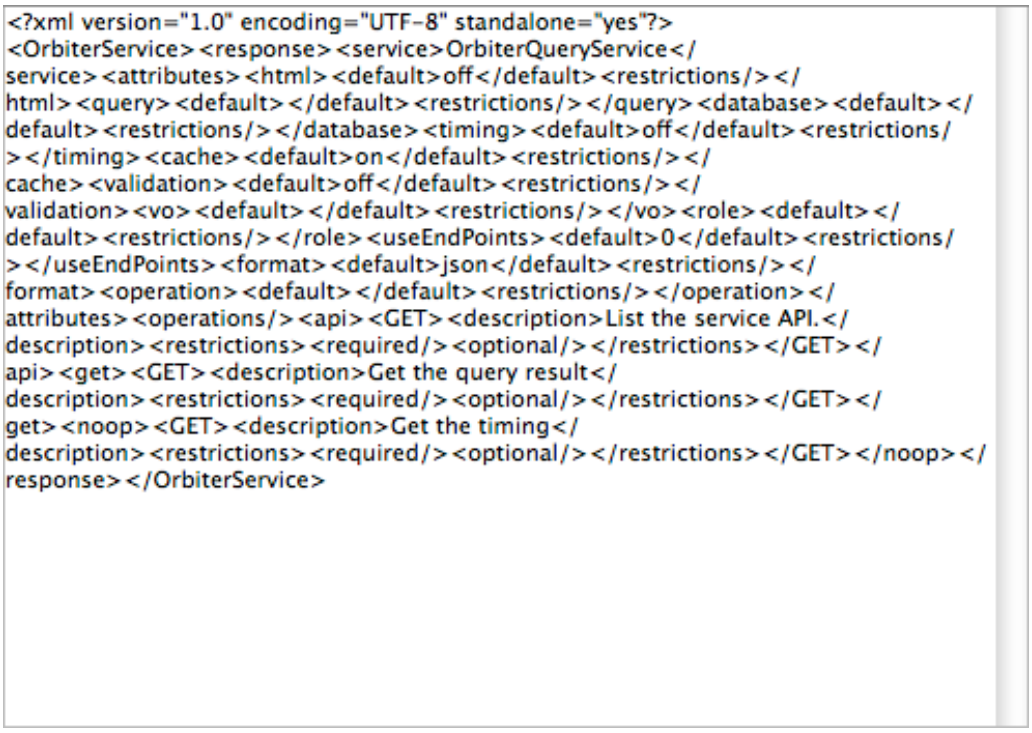
Orbiter Federation Testing dialog provides you with the following capabilities.

- The service operations of each service are provided in different tabs as shown in the above screen capture
- Required attributes are shown in Bold with a '*' suffix and optional attributes have a 'Use' check-box next to it as shown in the following screen capture
- The optional attributes uses the default value of the service attribute while testing the service if the 'Use' check-box is unchecked
- Check the 'Use' check-box and choose one of the acceptable values from the drop-down of an optional attributes to test the service with the service attribute
- Click on the 'Test' button to test the service with the chosen service attributes of an operation or default attribute values if not chosen
- It then displays the 'Service URL' and the result in the 'Result' area
- Choose 'Show Header' radio button to view the header of the service
- Choose 'Show Body' radio button to view the body of the result
- Check the check-box 'Wrap' to wrap the result displayed

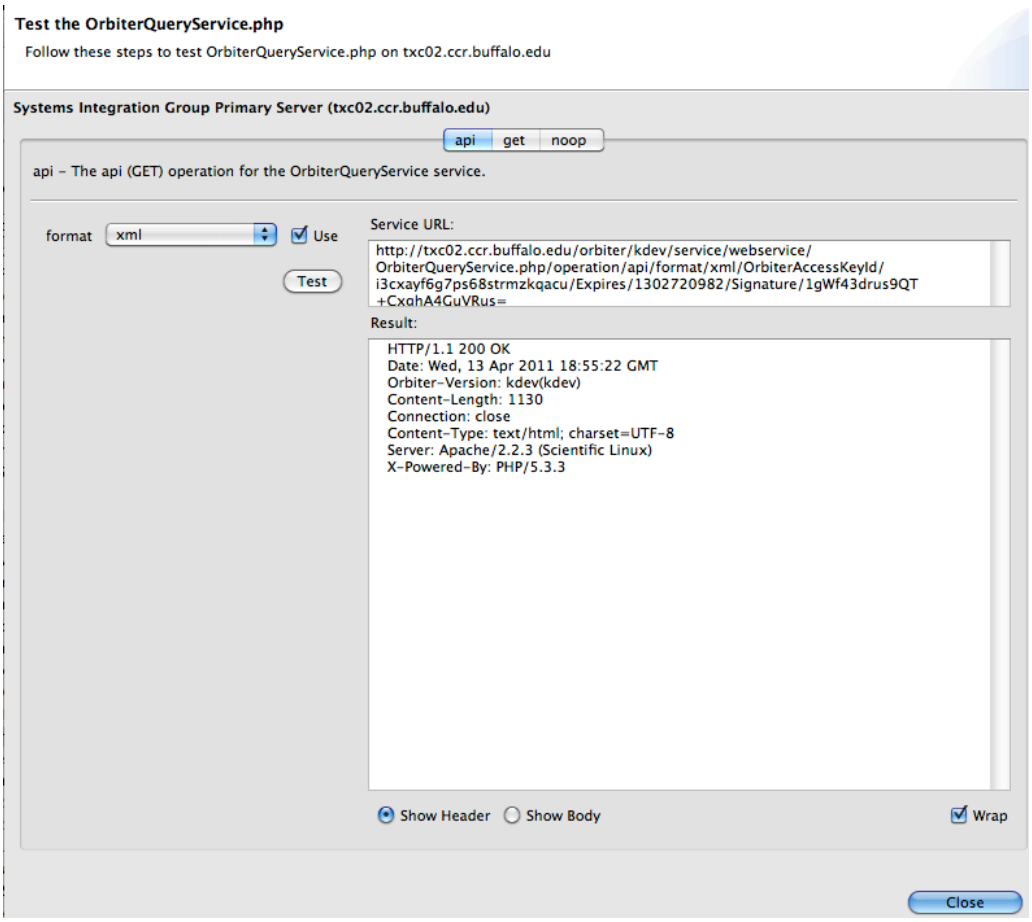
A screen capture of the 'Get' service operation for the service 'OrbiterQueryService' is tested is shown in the



A screen capture of the 'Result' wrapped when the service 'OrbiterQueryService' is tested with format as 'XML' is shown in the following screen capture



A screen capture of the header of the 'Result' wrapped when the service 'OrbiterQueryService' is tested is shown in the following screen capture

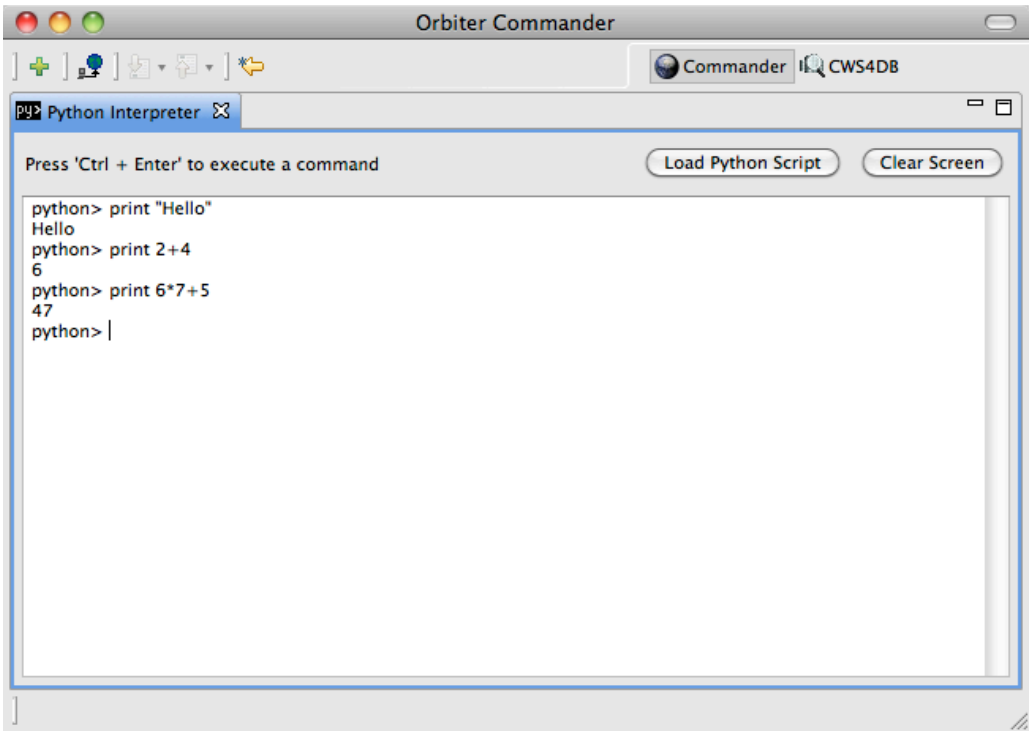


Python Interpreter Module Guide

Python Interpreter is an Orbiter Commander Module for executing Python commands.

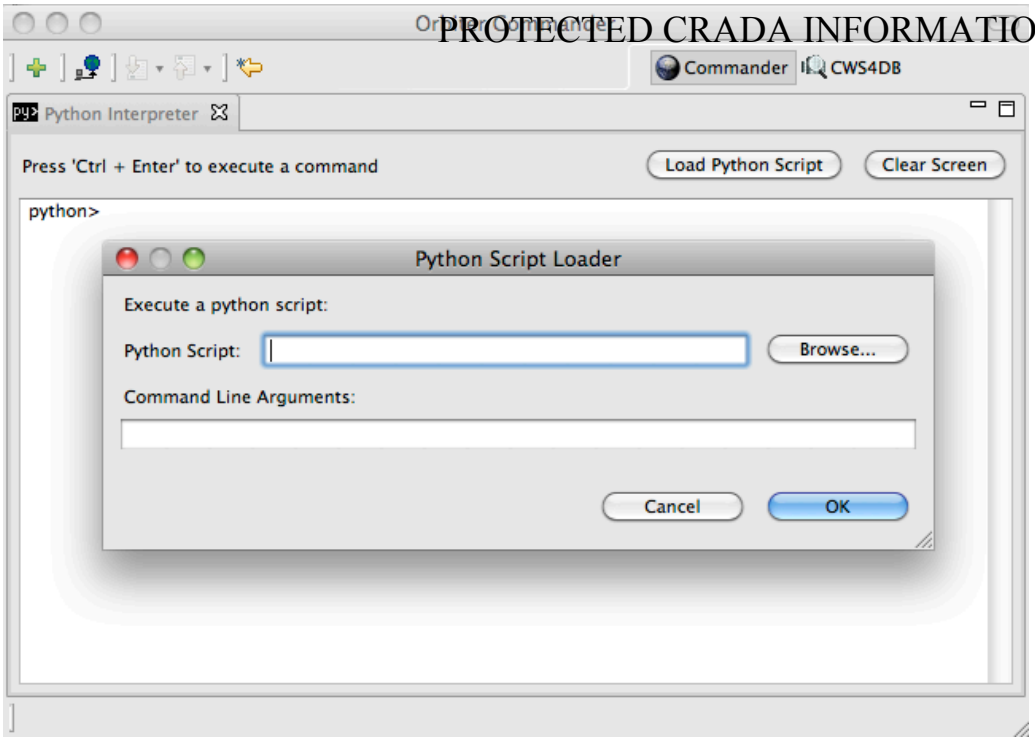
You can enter an expression and press '***Ctrl+Enter***' to execute it.

A screen capture of the 'Python Interpreter' while executing commands is shown below

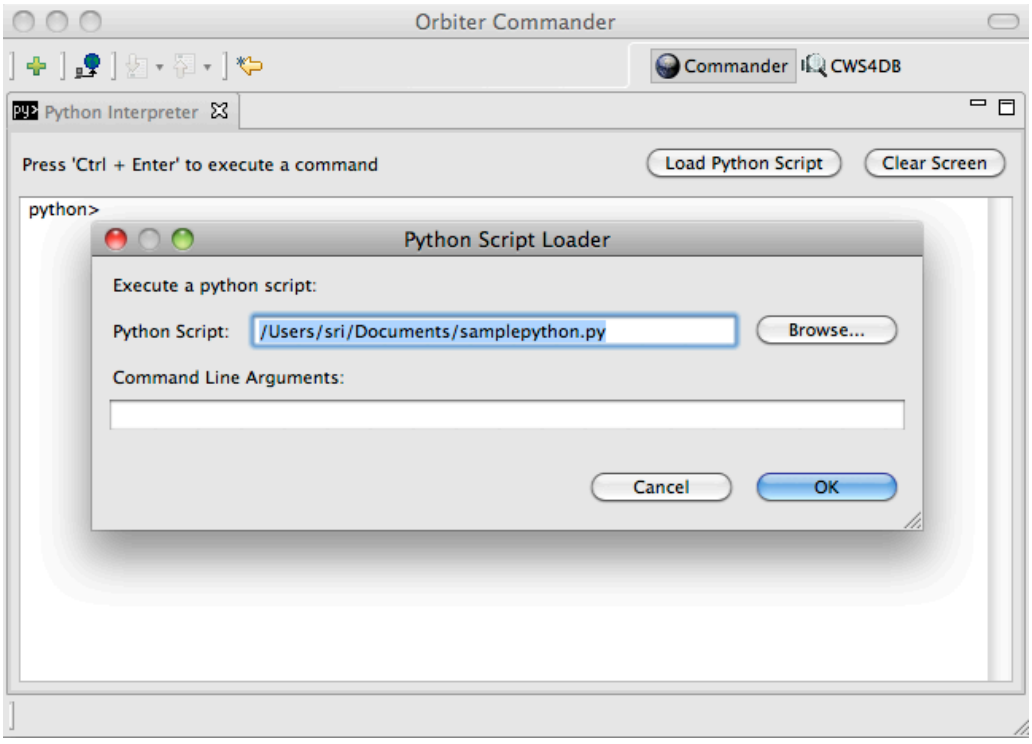


Alternatively, an existing Python Script can be executed by clicking on the '***Load Python Script***' button which opens a window to load the python script. Now you can select a python script by clicking on the '***Browse***' button and provide any Command Line Arguments needed in the '***Command Line Arguments:***' text box and click '***OK***' button to execute the script.

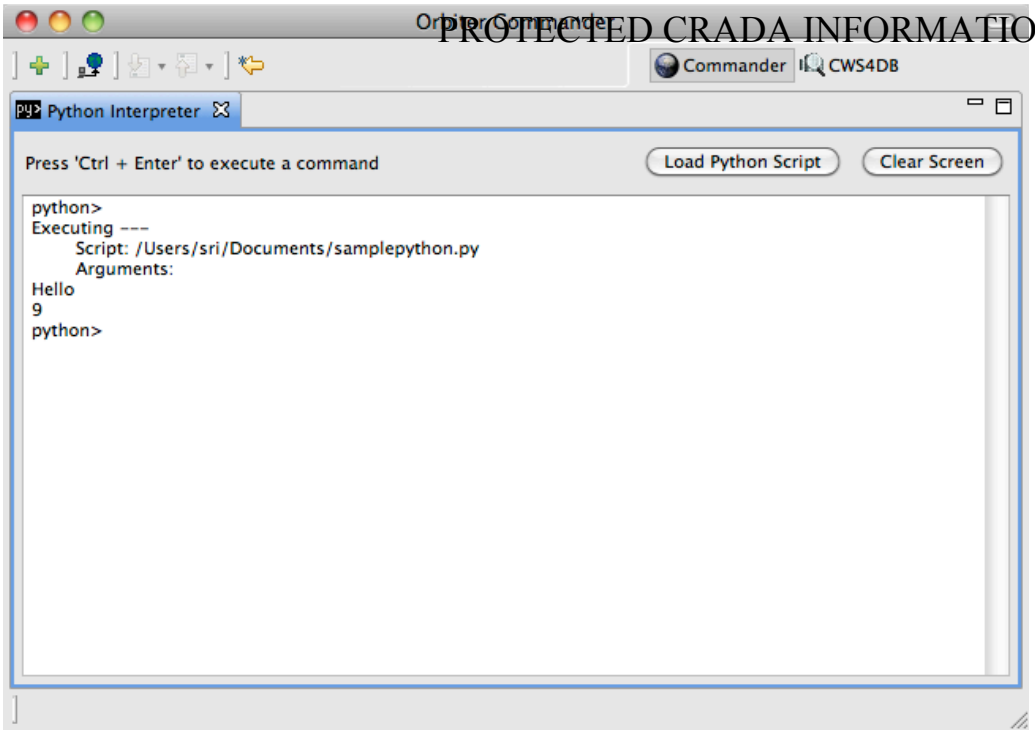
A screen capture of the 'Python Script Loader' before loading any script is shown below



For the sake of example, a sample python script is loaded in the 'Python Script Loader' as shown in the following screen capture



Once the 'OK' button is pressed, the python script is executed in the 'Python Interpreter' as shown in the following screen capture



The executed Python Script can be cleared by clicking on the '**Clear Screen**' button on the upper right-hand side.

4.3 Installation Guide

4.3.1 Introduction

This Installation Guide will explain how to install and configure CWS4DB STAR Web Services and the pre-requisites.

4.3.2 The Pre-requisites

STAR services depend on other components, softwares and database servers to be installed in order to run properly. From dependent applications or softwares to database servers, frameworks and run-time environments, it is the installer's job to make sure that the right versions are installed before the main application is ready to run.

Installation Steps for Pre-requisites

Step 1 : Install and Configure Web Server
Step 2 : Install and Configure PHP 5.3
Step 3 : Install and Configure MySQL 5.5.11

Step 1: Install and Configure Web Server

Prior to installing the lighttpd web server, check to make sure the following packages have been installed.

-**OpenSSL** is required for the transport layer encryption, thereby allowing the sites to be accessed via https.
-**Kerberos5** is a network authentication protocol. It is designed to provide strong authentication for clientserver applications by using secret-key cryptography.
-**zlib** is required for mod_compress for realtime, on-the-fly gzip compression for static content.
-**PCRE** library Perl Compatible Regular Expressions is a set of functions that implement regular expression pattern matching using the same syntax and semantics as Perl 5.

```
./configure --prefix=/Users/userName/OrbiterNetworkNode
 --with-openssl
 --with-kerberos5
 --with-ldap
 --with-zlib
```

A list will appear in the terminal window. Make sure the following lines appear, as they are important for PHP operation once lighted is installed:

```
mod_rewrite : enabled
mod_redirect : enabled
mod_ssi : enabled
mod_cgi : enabled
mod_fastcgi : enabled
mod_proxy : enabled
```

Next, finish with:

```
make && sudo make install
```

In order to configure the server, the configuration file must be in the right folder. This is done by copying the config file from the documentation folder to /etc/lighttpd/ by entering in the CLI:

```
sudo cp ./doc/lighttpd.conf /etc/lighttpd/lighttpd.conf
```

Open the file /etc/lighttpd/lighttpd.conf and uncomment the following lines if they are commented out.

```
server.modules = (
"mod_rewrite",
"mod_setenv",
"mod_secdownload",
"mod_access",
"mod_auth",
"mod_status",
"mod_expire",
"mod_simple_vhost",
"mod_redirect",
"mod_fastcgi",
"mod_cgi",
"mod_compress",
"mod_userdir",
"mod_ssi",
"mod_accesslog" )
```

Next, make the follow changes to the /etc/lighttpd/lighttpd.conf to match your server environment. Also, to insure that the lighttp install can properly log and record errors, Create the directory /var/log/lighttpd/ using the command :

```
sudo mkdir /var/log/lighttpd
```

Step 2: Install and Configure PHP 5.3

Prior to installing the web server, check to make sure the following packages have been installed.

-**bzip2** is also used for mod_compress for static content delivery by clients who use bzip2 compression.
-**GD** library is required for the creation and manipulation of images in PHP.
-**FreeType** is a font rasterization library. It is used for rendering text on to bitmaps and provides support for other font-related operations.
-**libpng** is the official PNG reference library. It supports almost all PNG features and is extensible.
-**ibjpeg** is a free software library written for JPEG image compression
-**GMP** Library allows the user to work with arbitrary-length integers.
-**gettext** library implement an NLS (Native Language Support) API which can be used to internationalize your PHP applications.

Extract the source archive and in the CLI execute :

```
./configure --prefix=/Users/userName/OrbiterNetworkNode/php5x
--mandir=/Users/sburley/OrbiterNetworkNode/share/man
--infodir=/Users/sburley/OrbiterNetworkNode/share/info
--with-zlib
--without-gdbm
--with-openssl
--enable-ftp
--enable-sockets
--without-pear
--disable-rpath
--enable-gd-native-ttf
--with-kerberos
--without-sqlite
--without-sqlite3
--disable-pdo
--with-openssl
--enable-xml
--with-ldap
--with-bz2
--with-gd
--enable-ftp
--enable-shmop
--enable-calendar
```

```
--with-curl
--disable-phar
--enable-sysvsem
--with-gettext
--with-gmp
--enable-phar
--with-sqlite
--with-sqlite3
--enable-pdo
--enable-mbstring
--enable-exif
--with-xsl
--with-zlib
--enable-magic-quotes
--with-pic
--enable-zip
--with-freetype-dir=/usr/bin
--with-jpeg-dir=/opt/local
--without-iconv --enable-wddx
--with-mysql
--with-sqlite
--with-sqlite3
--with-mysqli=mysqlnd
```

Next, finish with:

```
make
sudo make install
```

Note that the install directory ion this case is /Users/username/OrbiterNetworkNode/php5. This will ensure that the native install of PHP will not be stepped on and that the server can have both an Apache and lighttpd installation.

Finally, make sure that the php.ini installed on the system has the correct

Step 3: Install and Configure mySQL

mac OSX :

It is recommended, on a mac OSX system, not to compile and install MySQL ourselves, but instead use the OS X MySQL package. Not only is the install much faster and easier, but the package includes a startup item and a preference panel, and it is tuned by the MySQL team for OS X.

This archive file contains a prebuilt version of MySQL corresponding to Mac OS X Server version 10.6.8 SnowLeopard (10K540). In addition to the default MySQL installation on Server, this archive also contains the MySQL client static libraries (i.e. libmysqlclient.a) and associated C header files.

To install:

```
sudo tar -xzvf MySQL-55.root.tar.gz -C /
```

LINUX :

Although many linux distributions come with mySQL it is still prudent to upgrade to the latest version. Linux distributions of mySQL are installed quickly using the RPM packages

```
rpm -ivh MySQL-server-community-5.1.25-0.rhel5.i386.rpm MySQL-client-community-5.1.25-0.rhel5.i386.rpm
```

4.3.3 Installation of Commander

Installing for Mac OS X (32 and 64 bit)

(1) Check your Java installation by opening a terminal window and running the following commands:

```
~ which java
/usr/bin/java

~ java -version
java version "1.6.0_22"
Java(TM) SE Runtime Environment (build 1.6.0_22-b04-307-10M3261)
Java HotSpot(TM) Client VM (build 17.1-b03-307, mixed mode)
```

If you do not have Java installed or if the version is not at least 1.6, upgrade your system.

(2) You should match your system with the corresponding 32 or 64 bit Commander version. You can determine which version is appropriate using the following command in a terminal window:

```
uname -a
```

If the last argument returned by this command is ”i386” then use the 32 bit version, if it is ”i686” then use the 64 bit version.

(3) Download the zip file corresponding to your system architecture

(4) Unzip the file, this will result in a directory called OrbiterCommander. Move this directory to /Applications, or to another convenient installation area.

(5) Start Commander by double-clicking on the Commander.app file in the unzipped directory.

Installing for Linux (32 and 64 bit)

(1) Check your java installation by opening a terminal window and running the following commands:

```
 ~ which java
/usr/bin/java

~java -version
java version "1.6.0_18"
OpenJDK Runtime Environment (IcedTea6 1.8) (fedora-41.b18.fc13-i386)
OpenJDK Client VM (build 14.0-b16, mixed mode)
```

If you do not have Java installed or if the version is not at least 1.6, upgrade your system by downloading and installing Java from here

(2) Download the zip file corresponding to your system architecture

(3) Unzip the file, this will result in a directory called OrbiterCommander. Move this directory to a convenient installation area.

(4) Start Commander by double-clicking on the Commander executable in the unzipped directory.

4.3.4 Installation

To install the STAR services, copy the "orbiter" services folder into the system root directory "/var/www/html". Then to define the configuration settings go to the appropriate version (ex: trunk) and configuration folder (Local: /var/www/html/orbiter/trunk/webroot/ OrbiterFederation/configuration/conf/local AND Optional: /var/www/html/orbiter/trunk/webroot/ OrbiterFederation/configuration/conf/{server name.config.inc.php}) and define the below configuration settings.

The Local Configuration settings are shown in Figure 51.

The Optional Configuration settings are shown in Figure 52.

Figure 51: Orbiter STAR local configuration settings

```
1. ORBITERMASTER: Set the Master database connection constant.
2. ORBITERSLAVE: Set the Slave database connection constant.
3. SMTPPASSWORD: Set the smtp password.
```

Figure 52: Orbiter STAR configuration settings

```
1. ORBITERSERVICEACCESS: Transport level SSL security access (http:// | https://).
2. ORBITERDEFAULTDBPORT: Orbiter Database default port number.
3. ORBITERMAXURILENGTH: Set the maximum URI attribute length.
4. ORBITERERRORHANDLER: Set this to true use Orbiter Error handler in a service.
5. ORBITERDEBUGBACKTRACE: Set this true to the debug back trace for email and error handling.
6. ORBITERFATALERRORDIE: Set this true to DIE on a fatal error.
7. ORBITERERRORLOGFILE: Define a file for the error log.
8. ORBITERERRORTOEMAIL: Set email address to send error messages.
9. ORBITERCACHEFILELOCATION: Define the Cache file location.
10. ORBITERQUERYCONNECTIONSTRINGS: Define the returned number of STAR Query Connection strings.
11. ORBITERVERSION: Set this to the current Orbiter release version number.
12. ORBITERVERSIONFAMILY: Set this to the current Orbiter release version family.
```

4.3.5 How To Use STAR Services

Example Server Name: http://128.205.41.182 (http://txc02.ccr.buffalo.edu)

Example Base URI: http://128.205.41.182/orbiter/trunk/service/webservice/

Note: There was a significant time difference when we set the address to use https and DNS name. A secure https with a URI "https://txc02.ccr.buffalo.edu/orbiter/trunk/service/webservice/" was taking more time to resolve the DNS name, and when we use the "http" and IP address (http://128.205.41.182/orbiter/trunk/service/webservice/), it's taking 6-7 times less time to process the service request.

Orbiter Star Query Service

This is the STAR query service that is useful to connect and run queries against a query database. This query service can cache the query result and validate against query result and returns the cache file when the same sql query comes in, instead of going to the database to get the result set.

Brief example of use:

```
http://128.205.41.182/orbiter/trunk/service/webservice/OrbiterQueryService.php
/operation/get/format/json/database/Calibrations_svt/query/Select+%2A+from+
Nodes+where+Nodes.name%3D%27Hybrid_02%27+AND+Nodes.versionKey
%3D%27default%27
```

```
Response:
```

[{"name":"Hybrid_02","versionKey":"default","nodeType":"directory","structName"
:"None","elementID":"None","indexName":"Hybrid","indexVal":"2","baseLine":"N"
,"isBinary":"N","isIndexed":"Y","ID":"34","entryTime":"2002-02-14 14:44:45",
"Comment":""}]

```
Service parameters

 // Service name
 OrbiterQueryService.php

 // Service operations
  operation:'api'
  description: 'List the service API.'
  required attributes: ''
  optional attributes: 'format'
  service method: 'GET'

  operation:'get'
  description: 'Get the query result'
  required attributes: 'query', 'database'
  optional attributes: 'format', 'cache', 'vo', 'role', 'useEndPoints', 'validation'
  service method: 'GET'

  operation:'noop'
  description: 'Get the timing'
  required attributes: ''
  optional attributes: 'timing'
  service method: 'GET'

 // Service attribute definitions
   attribute: 'operation'
  description: 'Specify the type of operation service request required.'
  defaultValue: ''
  allowableValues: 'api', 'get', 'noop'

  attribute: 'html'
  description: 'Generate HTML compliant output by replacing escaped character
          and use appropriate tagging and line returns.'
  defaultValue: 'off'
  allowableValues: 'on', 'off'

  attribute: 'query'
  description: 'Specify sql query with urlencode.'
  defaultValue: ''
```

```
allowableValues: ''

attribute: 'database'
description: 'Specify database name on which query can be run.'
defaultValue: ''
allowableValues: ''

attribute: 'timing'
description: 'Specify whether to log response time in the service log.'
defaultValue: 'off'
allowableValues: 'on','off'

attribute: 'cache'
description: 'Specify whether to keep cache on.'
defaultValue: 'on'
allowableValues: 'on', 'off'

attribute: 'validation'
description: 'Compares the database response contents and the
       cache contents when set to "on"'
defaultValue: 'off'
allowableValues: 'on', 'off'

attribute: 'vo'
description: 'Virtual Organization Name'
defaultValue: ''
allowableValues: ''

attribute: 'role'
description: 'Role in the Virtual Organization'
defaultValue: ''
allowableValues: ''

attribute: 'useEndPoints'
description: 'Gets the number of connection strings from load balancer
       when set to more than 0'
defaultValue: '0'
allowableValues: ''

attribute: 'format'
description: 'Specify the service response format.'
defaultValue: 'json'
allowableValues: 'json', 'api', 'xml', 'text', 'custom'
```

Orbiter Simulator Service

This STAR Simulator service reads the sql files, identifies the database and query from the sql file and then generates Orbiter query service requests. This service is also useful to run and get the Orbiter query response and response times.

Brief example of use:

```
http://128.205.41.182/orbiter/trunk/service/webservice/OrbiterSimulator
Service.php/operation/runfile/file//tmp/testfiles/auau7_log.txt/format/
json/cache/on/debug/on

Response: Reads the sql file auau7_log.txt, runs all the sql queries
  against the Orbiter query service
and gets the service response in json format.

// Service name
 OrbiterSimulatorService.php

 // Service operations
  operation:'api'
 description: 'List the service API.'
 required attributes: ''
 optional attributes: 'format'
 service method: 'GET'

 operation:'runfile'
 description: 'Runs Star sql files.'
 required attributes: 'file'
 optional attributes: 'format', 'cache', 'noop', 'output', 'detail',
          'debug', 'trails'
 service method: 'GET'

 // Service attribute definitions
  attribute: 'operation'
 description: 'Specify the type of operation service request required.'
 defaultValue: ''
 allowableValues: 'api', 'noop', 'runfile'

 attribute: 'html'
 description: 'Generate HTML compliant output byt replacing escaped character
       and use appropriate tagging and line returns.'
 defaultValue: 'off'
 allowableValues: 'on', 'off'
```

```
attribute: 'file'
description: 'Specify the absolute file name containing the
      STAR application database queries.'
defaultValue: ''
allowableValues: ''

attribute: 'database'
description: 'Specify the database name for the query to run.'
defaultValue: ''
allowableValues: ''

attribute: 'debug'
description: 'Output the query and response in addition to the normal
      statistics when "on".'
defaultValue: 'off'
allowableValues: 'on', 'off'

attribute: 'cache'
description: 'Set to "on" to cache the queries and use cached queries.'
defaultValue: 'on'
allowableValues: 'on', 'off'

attribute: 'detail'
description: 'Set to "on" to output the cumulative query timing.'
defaultValue: 'off'
allowableValues: 'on', 'off'

attribute: 'noop'
description: 'Specify whether to log the service request.'
defaultValue: 'off'
allowableValues: 'on', 'off'

attribute: 'output'
description: 'Set to output the trial queries and not perform them.'
defaultValue: 'off'
allowableValues: 'on', 'off'

attribute: 'address'
description: 'Set to a valid Orbiter network node or basestation
      with a deployed service.'
defaultValue: ''
allowableValues: ''

attribute: 'trials'
```

```
description: 'Set the number of times to perform trial query set and
      report the average statistics.'
defaultValue: '1'
allowableValues: ''

attribute: 'format'
description: 'Specify the service response format.'
defaultValue: 'json'
allowableValues: 'json', 'api', 'xml', 'space'

attribute: 'timing'
description: 'Specify whether to log response time in the service log.'
defaultValue: 'off'
allowableValues: 'on', 'off'
```

Orbiter Cache File Service

This Orbiter cache file service class is responsible for cache file retrieves/stores the contents from/to cache.

Brief example of use:

```
http://128.205.41.182/orbiter/trunk/service/webservice/OrbiterCache
FileService.php/operation/list

 // Service name
 OrbiterCacheFileService.php

 // Service operations

 operation:'api'
 description: 'List the service API.'
 required attributes: ''
 optional attributes: 'format'
 service method: 'GET'

 operation:'noop'
 description: 'List the time taken to process the request.'
 required attributes: ''
 optional attributes: 'format'
 service method: 'GET'

 operation:'cacheFileHashType'
 description: 'Get the hash type defined.'
 required attributes: ''
 optional attributes: 'format'
 service method: 'GET'

 operation:'cacheFileList'
 description: 'List all the cache files.'
 required attributes: ''
 optional attributes: 'format', 'vo', 'role'
 service method: 'GET'

 operation:'cacheFileGet'
 description: 'Get the contents of cache file.'
 required attributes: 'conStr'
 optional attributes: 'format', 'hashStr', 'vo', 'role'
 service method: 'GET'
```

```
operation:'cacheFileDelete'
description: 'Delete a specific cache file for a given hash string
 or canonical string. When using delete, either conStr or
 hashStr needs to be specified.'
required attributes: 'conStr'
optional attributes: 'hashStr', 'vo', 'role'
service method: 'GET'

operation:'cacheFileDeleteAll'
description: 'Delete all the cache files.'
required attributes: ''
optional attributes: 'vo', 'role'
service method: 'GET'

// Service attribute definitions

    Example Use: http://128.205.41.182/orbiter/trunk/service/webservice/
    OrbiterCacheFileService.php/operation/list
    Result: {"filename":"45026ae00a6f455c7eec7d97adc4f576bfcf051a",
    "filesize":"364"}/operation/get
    - Returns the Cache file for a given hash string or canonical string.
    Example Use: python ./testOrbiterREST.py http://128.205.41.182/
    orbiter/trunk/service/webservice/
    OrbiterCacheFileService.php/operation/get
    hashstr/45026ae00a6f455c7eec7d97adc4f576bfcf051a
    Result: <records><record><name>Sector_07</name><versionKey>
    default</versionKey><nodeType>directory</nodeType>
    <structName>None</structName><elementID>7</elementID>
    <indexName>Sector</indexName><indexVal>7</indexVal>
    <baseLine>N</baseLine><isBinary>N</isBinary><isIndexed>
    Y</isIndexed><ID>13</ID><entryTime>2000-01-12 20:11:07
    </entryTime><Comment></Comment></record></records>

attribute: 'operation'
description: 'Specify the type of operation service request required.'
defaultValue: ''
allowableValues: 'api', 'noop', 'cacheFileHashType', 'cacheFileList',
       'cacheFileGet', 'cacheFileDelete', 'cacheFileDeleteAll'

attribute: 'html'
description: 'Generate HTML compliant output by replacing
      escaped character and use appropriate tagging
       and line returns.'
defaultValue: 'off'
```

```
allowableValues: 'on', 'off'

attribute: 'delete'
description: 'Specify whether to delete the file.'
defaultValue: 'off'
allowableValues: 'on', 'off'

attribute: 'timing'
description: 'Specify whether to log response time in the service log.'
defaultValue: 'off'
allowableValues: 'on', 'off'

attribute: 'noop'
description: 'Specify whether to log the service.'
defaultValue: 'off'
allowableValues: 'on', 'off'

attribute: 'conStr'
description: 'Canonical String.'
defaultValue: ''
allowableValues: ''

attribute: 'hashStr'
description: 'Hash string.'
defaultValue: ''
allowableValues: ''

attribute: 'vo'
description: 'Virtual Organization name.'
defaultValue: ''
allowableValues: ''

attribute: 'role'
description: 'Role in the Virtual Organization.'
defaultValue: ''
allowableValues: ''

attribute: 'format'
description: 'Specify the service response format.'
defaultValue: 'json'
allowableValues: 'json', 'api', 'xml', 'text', 'custom'
```

Orbiter Query Database Load Balancer Service

This service is responsible for load balancing the query and other databases. Also used for updating the database rank and status. The Service response will return connection strings of the type specified by the user based on the rank and status.

Brief example of use:

```
http://128.205.41.182/orbiter/trunk/service/webservice/OrbiterQueryDb
LoadBalancerService.php/operation/get


// Service name
OrbiterQueryDbLoadBalancerService.php

// Service operations
operation:'api'
description: 'List the service API.'
required attributes: ''
optional attributes: 'format'
service method: 'GET'

operation:'get'
description: 'Get the connection strings'
required attributes: 'endpoints', 'type'
optional attributes: 'format'
service method: 'GET'

operation:'update'
description: 'Update the connection string'
required attributes: 'rank', 'conStr'
optional attributes: 'status'
service method: 'GET'

// Service attribute definitions
attribute: 'operation'
description: 'Specify the type of operation service request required.'
defaultValue: ''
allowableValues: 'api', 'get', 'update'

attribute: 'html'
description: 'Generate HTML compliant output by replacing
  escaped character and use appropriate tagging
and line returns.'
```

```
defaultValue: 'off'
allowableValues: 'on', 'off'

attribute: 'conStr'
description: 'Database Connection String'
defaultValue: ''
allowableValues: ''

attribute: 'type'
description: 'Type of the database'
defaultValue: 'star'
allowableValues: ''

attribute: 'endpoints'
description: 'Number of connection strings to be retrieved'
defaultValue: 'ORBITERQUERYCONNECTIONSTRINGS'
allowableValues: ''

attribute: 'status'
description: 'Status of the Connection String'
defaultValue: '1'
allowableValues: '1', '0'

attribute: 'rank'
description: 'Database Rank'
defaultValue: '1'
allowableValues: ''

attribute: 'format'
description: 'Specify the service response format.'
defaultValue: 'json'
allowableValues: 'json', 'api', 'xml', 'text', 'custom'
```

Orbiter Query Database Connection String Service

This service provides the user an ability to add, update, and delete the database connection string information; so that the user can manage database resources on demand.

Brief example of use:

```
http://128.205.41.182/orbiter/trunk/service/webservice/OrbiterQuery
DbConnectionStringService.php/operation/api

// Service name
OrbiterQueryDbConnectionStringService.php

// Service operations
operation:'api'
description: 'List the service API.'
required attributes: ''
optional attributes: 'format'
service method: 'GET'

operation:'insert'
description: 'Insert the connection string'
required attributes: 'conStr', 'type'
optional attributes: 'rank', 'hostType'
service method: 'GET'

operation:'update'
description: 'Update the connection string'
required attributes: 'conStr', 'type'
optional attributes: 'rank', 'status', 'hostType'
service method: 'GET'

operation:'delete'
description: 'Delete the connection string'
required attributes: 'conStr'
optional attributes: ''
service method: 'GET'

// Service attribute definitions
attribute: 'operation'
description: 'Specify the type of operation service request required.'
defaultValue: ''
allowableValues: 'api', 'insert', 'update', 'delete'
```

```
attribute: 'html'
description: 'Generate HTML compliant output by replacing
 escaped character and use appropriate tagging and line returns.'
defaultValue: 'off'
allowableValues: 'on', 'off'

attribute: 'conStr'
description: 'Database Connection String'
defaultValue: ''
allowableValues: ''

attribute: 'type'
description: 'Type of the database'
defaultValue: 'star'
allowableValues: ''

attribute: 'hostType'
description: 'Type of Host'
defaultValue: 'local'
allowableValues: 'local', 'remote'

attribute: 'status'
description: 'Status of the Connection String'
defaultValue: '1'
allowableValues: '1', '0'

attribute: 'rank'
description: 'Database Rank'
defaultValue: '0'
allowableValues: ''

attribute: 'format'
description: 'Specify the service response format.'
defaultValue: 'json'
allowableValues: 'json', 'api', 'xml', 'text', 'custom'
```