caGrid 1.0
Programmer's Guide

Center for Bioinformatics

December 15, 2006

caGrid Development and Management Teams		
Scott Oster (Lead Architect)[1]	Ian Foster[2]	Avinash Shanbhag[9]
Stephen Langella[1]	Patrick McConnell[3]	
Shannon Hastings[1]	David Wellborn[4]	
David Ervin[1]	Val Bragg[4]	
Tahsin Kurc[1]	Vinay Kumar[5]	
Joel Saltz[1]	Joshua Phillips[5]	
Ravi Madduri[2]	Ram Chilukuri[5]	
Jarek Gawor[2]	Srini Akkala[5]	
Frank Siebenlist[2]	Manav Kher[6]	
Mike Wilde[2]	Wendy Erickson-Hirons[7]	
Raj Kettimuthu[2]	Arumani Manisundaram[8]	
Bill Allcock[2]	George Komatsoulis[9]	
[1]Ohio State University - Biomedical Informatics Department	[2]University of Chicago/Argonne National Laboratory	[3]Duke Comprehensive Cancer Center
[4]ScenPro, Inc.	[5]SemanticBits, LLC.	[6]Science Application International Corporation (SAIC)
[7]Northern Taiga Ventures, Inc. (NTVI)	[8]Booz Allen Hamilton	[9]National Cancer Institute Center for Bioinformatics (NCICB)

Other Acknowledgements
GeneConnect – Project - Washington University
GridIMAGE – Project - Ohio State University
caBIO – Project - National Cancer Institute Center for Bioinformatics (NCICB)
caArray – Project - National Cancer Institute Center for Bioinformatics (NCICB)
caTRIP – Project – Duke Comprehensive Cancer Center
GenePattern – Project – Broad Institute
geWorkbench – Columbia University
caBiocondutor – Project – Fred Hutchinson Cancer Research Center
Terpsys – Systems Team - National Cancer Institute Center for Bioinformatics (NCICB)

Contacts and Support	
NCICB Application Support	http://ncicbsupport.nci.nih.gov/sw/ Telephone: 301-451-4384 Toll free: 888-478-4423

LISTSERV Facilities Pertinent to caGrid		
LISTSERV	***URL***	***Name***
cagrid_users-l@list.nih.gov	https://list.nih.gov/archives/cagrid_users-l.html	caGrid Users Discussion Forum

Table of Contents

Chapter 1 About This Guide 1

Purpose 1
Release Schedule 1
Audience 1
Getting Help 1
How to Use This Guide 1
Relevant Documents 2
Document Text Conventions 2

Chapter 2 Overview of caGrid 5

Introduction 5
Standards Compliant 6
Model Driven 6
Semantically Discoverable 8
Security and Manageability 9
Revolutionary Development 12

Chapter 3 caGrid Metadata Infrastructure 15

Metadata API Usage Overview 15
Discovery API Usage Overview 24
caDSR Grid Service Usage Overview 45
EVS API Usage Overview 57

Chapter 4 caGrid Security 65

Dorian Overview 65
Grid Grouper Overview 69

Chapter 5 caGrid Data Services 75

Overview 75
Manipulating CQL Query Results 76
Utility Classes 78
CQL Query Syntax 81
Domain Model Conformance 82
Results Validation 82
CQL Query Processors 83

Federated Query Processor Usage Overview 84
API Details 88

Chapter 6 Reference Implementations 99

Overview 99
Objective 100
Goals 100
Assumptions 100
High-Level Process 100
Deliverables 101
Test Bed 101
caArray Gridification 102
caBioconductor 102
caTRIP 103
GenePattern 104
GeneConnect 105
geWorkbench 106
GridIMAGE 107

Chapter 7 WS-Enumeration 109

Overview 109
Client API 109

Chapter 8 Workflow Management Service 121

Overview 121
Workflow Architecture 121
WorkflowFactoryService API 123
WorkflowManagementService API 125
Security in WorkflowFactory and Context Services 128
Service Selection 128
Provenance Tracking 128
WS-RF Resources in Workflows 128

Chapter 9 caGrid Global Model Exchange 131

Overview 131
GME Client 134

Appendix A References 137

Scientific Publications 137
Technical Manuals/Articles 140

caBIG Material 141
caCORE Material 141
Glossary 141
Index 145

Chapter 1 About This Guide

Purpose

The cancer Biomedical Informatics Grid, or caBIG™, is a voluntary virtual informatics infrastructure that connects data, research tools, scientists, and organizations to leverage their combined strengths and expertise in an open environment with common standards and shared tools. The current test bed architecture of caBIG™, is dubbed caGrid. The software embodiment and corresponding documentation of this architecture constitute the caGrid 1.0 release. This guide describes the APIs provided by caGrid.

Release Schedule

This guide has been updated for the caGrid 1.0 release. It may be updated between releases if errors or omissions are found. The current document refers to the 1.0 version of caGrid, released in December 2006 by caBIG.

Audience

The primary audience of this guide is the programmer who wants to learn about the APIs provided by caGrid and/or requires access to one or more caGrid APIs. For additional information about using caGrid, see the caGrid User's Guide.

This guide assumes that you are familiar with the java programming language and/or other programming languages, database concepts, and the Internet. If you intend to use caGrid resources in software applications, it assumes that you have experience with building and using complex data systems.

Getting Help

NCICB Application Support

http://ncicbsupport.nci.nih.gov/sw/

Telephone: 301-451-4384

Toll free: 888-478-4423

How to Use This Guide

This guide is divided into sections that each describes a different caGrid API. The following list briefly describes the contents of each chapter.

- Chapter 1, this chapter, provides an overview of the guide.
- Chapter 2 provides an overview of the cancer Biomedical Informatics Grid, or caBIG™, a voluntary virtual informatics infrastructure that connects data, research tools, scientists,

and organizations to leverage their combined strengths and expertise in an open federated environment with widely accepted standards and shared tools.

- Chapter 3 describes the caGrid metadata infrastructure and describes the Discovery API, caDSR Grid Service, and EVS APIs.
- Chapter 4 describes using Dorian and Grid Grouper as part of caGrid security.
- Chapter 5 describes the caGrid Data Services infrastructure.
- Chapter 6 describes Reference Implementations, where caBIG-developed projects are aim to adopt the caGrid 1.0 infrastructure before it is released.
- Chapter 7 describes the client-side APIs for enumerations.
- Chapter 8 describes the architecture and APIs for interacting with caGrid workflow.
- Chapter 9 describes the caGrid Global Model Exchange (GME).

Relevant Documents

This Programmer's Guide addresses caGrid Application Programming Interfaces (API) and API examples. Additional information about caGrid architecture, design, user-oriented overview and examples, and tool-specific guides can be found in:

Document	Location
caGrid 1.0 User's Guide	http://gforge.nci.nih.gov/frs/?group_id=25
caGrid 1.0 Design Documents and Tool-specific Guides	https://gforge.nci.nih.gov/plugins/scmcvs/cvsweb.php/cagrid-1-0/Documentation/docs/?cvsroot=cagrid-1-0

Document Text Conventions

The following table shows how text conventions are represented in this guide. The various typefaces differentiate between regular text and menu commands, keyboard keys, and text that you type.

Convention	*Description*	*Example*
Bold & Capitalized Command **Capitalized command > Capitalized command**	Indicates a Menu command Indicates Sequential Menu commands	**Admin > Refresh**
TEXT IN SMALL CAPS	Keyboard key that you press	Press ENTER
TEXT IN SMALL CAPS + TEXT IN SMALL CAPS	Keyboard keys that you press simultaneously	Press SHIFT + CTRL and then release both.
Special typestyle	Used for filenames, directory names, commands, file listings, source code examples and anything that would appear in a Java program, such as	`URL_definition ::= url_string`

Convention	*Description*	*Example*
	methods, variables, and classes.	
Boldface type	Options that you select in dialog boxes or drop-down menus. Buttons or icons that you click.	In the Open dialog box, select the file and click the **Open** button.
Italics	Used to reference text that you type.	Enter *antrun.*
Note:	Highlights a concept of particular interest	**Note:** This concept is used throughout the installation manual.
Hyperlink	Links text to another part of the document or to a URL	Overview

Table 1-1 Document Conventions

Chapter 2 Overview of caGrid

This chapter provides an overview of the cancer Biomedical Informatics Grid, or caBIG™, a voluntary virtual informatics infrastructure that connects data, research tools, scientists, and organizations to leverage their combined strengths and expertise in an open federated environment with widely accepted standards and shared tools.

Topics in this chapter include:

- Introduction on this page
- Standards Compliant on page 6
- Model Driven on page 6
- Semantically Discoverable on page 8
- Security and Manageability on page 9
- Revolutionary Development on page 12

Introduction

The cancer Biomedical Informatics Grid, or caBIG™, is a voluntary virtual informatics infrastructure that connects data, research tools, scientists, and organizations to leverage their combined strengths and expertise in an open federated environment with widely accepted standards and shared tools. The underlying service oriented infrastructure that supports caBIG™ is referred to as caGrid. Driven primarily by scientific use cases from the cancer research community caGrid provides the core enabling infrastructure necessary to compose the Grid of caBIG™. It provides the technology that enables collaborating institutions to share information and analytical resources efficiently and securely, and allows investigators to easily contribute to and leverage the resources of a national-scale, multi-institutional environment

The caGrid 0.5 "test bed" infrastructure was released in September 2005 and included the initial set of software tools to effectively realize the goals of caBIG™. The grid technologies and methodologies adopted for caBIG™, and implemented in caGrid, provide a loosely coupled environment wherein local providers are given freedom of implementation choices and ultimate control over access and management. The technologies harmonize on community accepted virtualizations of the data they use, and make them available using standardized service interfaces and communication mechanisms. While caGrid enables numerous complex usage scenarios, in its simplest base, its goals are to: enable universal mechanisms for providing interoperable programmatic access to data and analytics to caBIG™, create a self-describe infrastructure wherein the structure and semantics of data can be programmatically determined, and provide a powerful means by which resources available in caBIG™ can be programmatically discovered and leveraged. Additional information about the caGrid 0.5 effort, and a good overview of the motivation of the grid approach of caBIG™, can be found in the Bioinformatics Journal article at

http://bioinformatics.oxfordjournals.org/cgi/content/full/22/15/1910.

Building on the foundation of caGrid 0.5, caGrid 1.0 has been extensively enhanced based on feedback and input from early adopters of the caGrid 0.5 infrastructure and additional requirements from the various caBIG™ Domain Workspaces. The release of caGrid version 1.0 represents a major milestone in the caBIG™ program towards achieving the program goals. It provides the implementation of the required core services, toolkits, and wizards for the development and deployment of community provided services, APIs for building client applications, and some reference implementations of applications and services available in the production grid.

Standards Compliant

As a primary principle of caBIG™ is open standards, caGrid is built upon the relevant community-driven standards of the World Wide Web Consortium (W3C http://www.w3.org/) and OASIS (http://www.oasis-open.org). It is also informed by the efforts underway in the Open Grid Forum (OGF http://ogf.org/). The Open Grid Forum (OGF) is a community of users, developers, and vendors leading the global standardization effort for grid computing. The OGF community consists of thousands of individuals in industry and research, representing over 400 organizations in more than 50 countries. As such, while the caGrid infrastructure is built upon the 4.0 version of the Globus Toolkit (GT4 http://globus.org/toolkit/), it shares a Globus goal to be programming language and toolkit agnostic by leveraging existing standards. Specifically, caGrid services are standard WSRF v1.2 services and can be accessed by any specification-compliant client.

caGrid 1.0 also represents an increased involvement in relevant working groups, standards bodies, and organizations involved in the standardization and adoption of grid technologies. The caGrid team consists of several members involved in both the development of the Globus toolkit, and authors on some of the relevant specifications. Furthermore, some of the components developed by caGrid have been published in peer-reviewed articles, have been vetted by the grid community in several invited talks, and are undergoing an incubation process to become part of the Globus toolkit itself.

Model Driven

Extending beyond the basic grid infrastructure, caBIG™ specializes these technologies to better support the needs of the cancer research community. A primary distinction between basic grid infrastructure and the requirements identified in caBIG and implemented in caGrid is the attention given to data modeling and semantics. caBIG™ adopts a model-driven architecture best practice and requires that all data types used on the grid are formally described, curated, and semantically harmonized. These efforts result in the identification of common data elements, controlled vocabularies, and object-based abstractions for all cancer research domains. caGrid leverages existing NCI data modeling infrastructure to manage, curate, and employ these data models. Data types are defined in caCORE UML and converted into ISO/IEC 11179 Administered Components, which are in turn registered in the Cancer Data Standards Repository (caDSR). The definitions draw from vocabulary registered in the Enterprise Vocabulary Services (EVS), and their relationships are thus semantically described.

caGrid 1.0 represents a significant improvement in its leveraging of these technologies and the corresponding information they make available. caGrid 1.0 provides grid service access to both the EVS and caDSR, and its new service metadata standards include significant additions of information extracted from the caDSR and EVS.

In caGrid, both the client and service APIs are object oriented, and operate over well-defined and curated data types. Clients and services communicate through the grid using respectively grid clients and grid service infrastructure. The grid communication protocol is XML, and thus the client and service APIs must transform the transferred objects to and from XML. This XML serialization of caGrid objects is restricted in that each object that travels on the grid must do so as XML, which adheres to an XML schema registered in the Global Model Exchange (GME). As the caDSR and EVS define the properties, relationships, and semantics of caBIG™ data types, the GME defines the syntax of their XML materialization. Furthermore, caGrid services are defined by the Web Service Description Language (WSDL). The WSDL describes the various operations the service provides to the grid. The inputs and outputs of the operations, among other things, in WSDL are defined by XML schemas. As caBIG™ requires that the inputs and outputs of service operations use only registered objects, these input and output data types are defined by the XSDs that are registered in GME. In this way, the XSDs are used both to describe the contract of the service and to validate the XML serialization of the objects that it uses. Figure 2-1 details the various services and artifacts related to the description of and process for the transfer of data objects between client and service.

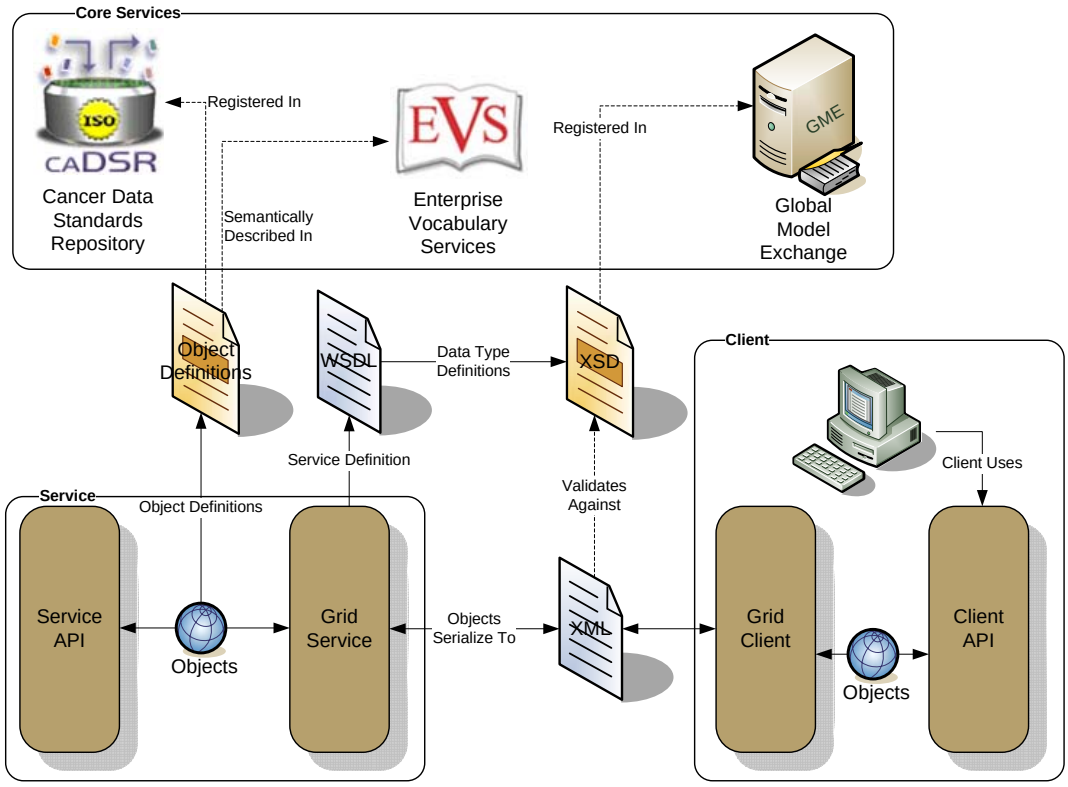


Figure 2-1 caGrid data description infrastructure

Semantically Discoverable

As caBIG™ aims to connect data and tools from 50+ disparate cancer centers and many other institutions, a critical requirement of its infrastructure is that it supports the ability of researchers to discover these resources. caGrid enables this ability by taking advantage of the rich structural and semantic descriptions of data models and services that are available. Each service is required to describe itself using caGrid standard service metadata. When a grid service is connected to the caBIG™ grid, it registers its availability and service metadata with a central indexing registry service (Index Service). This service can be thought of as the "yellow pages" and "white pages" of caBIG™. A researcher can then discover services of interest by looking them up in this registry. caGrid 1.0 provides a series of high-level APIs and user applications for performing this lookup, which greatly facilitate the discovery process.

As the Index Service contains the service metadata of all the currently advertised and available services in caBIG™, the expressivity of service discovery scenarios is limited only by the expressivity of the service metadata. For this reason, caGrid provides standards for service metadata to which all services must adhere. At the base is the common Service Metadata standard that every service in caBIG™ is required to provide. This metadata contains information about the service-providing cancer center, such as the point of contact and the institution's name. Data Services, as a standardized type of caGrid services, also provide an additional Domain Model metadata standard. Both of these standards leverage the data models registered in caDSR and link them to the underlying semantic concepts registered in EVS. The Data Service Metadata details the domain model from which the Objects being exposed by the

service are drawn. Additionally, the definitions of the Objects themselves are described in terms of their underlying concepts, attributes, attribute value domains, and associations to other Objects being exposed. Similarly, the common Service Metadata details the Objects, used as input and output of the services operations, using the same format as the Data Service metadata. In addition to detailing the Objects definitions, the Service Metadata defines and describes the operations or methods the service provides, and allows semantic concepts to be applied to them. In this way, all services fully define the domain objects they expose by referencing the data model registered in caDSR, and identify their underlying semantic concepts by referencing the information in EVS. The caGrid metadata infrastructure and supporting APIs and toolkits are defined with extensibility in mind, encouraging the development of additional domain or application specific extensions to the advertisement and discovery process.

As shown in Figure 2-2, the caGrid discovery API and tools allow researchers to query the Index Service for services satisfying a query over the service metadata. That is, researchers can lookup services in the registry using any of the information used to describe the services. For instance, all services from a given cancer center can be located, data services exposing a certain domain model or objects based on a given semantic concept can be discovered, as can analytical services that provide operations that take a given concept as input.

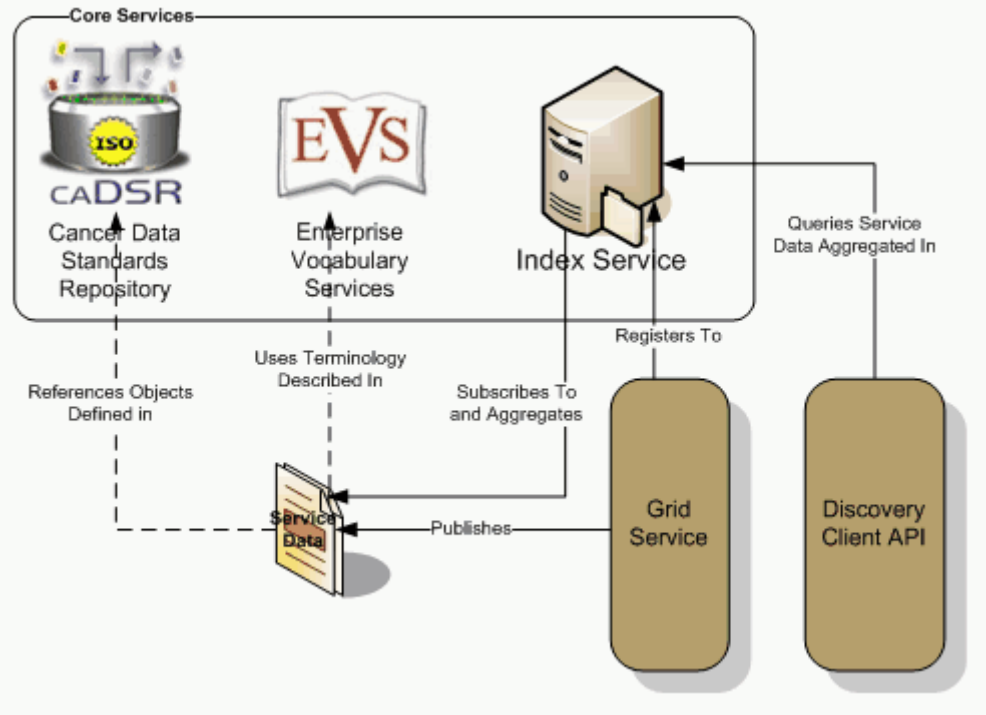


Figure 2-2 caGrid Discovery Overview

Security and Manageability

Security is an especially important component of caBIG™ both for protecting intellectual property and ensuring protection and privacy of patient related and sensitive information. caGrid

1.0 provides a complete overhaul of federated security infrastructure to satisfy caBIG™ security needs, incorporating many of the recommendations made in the caBIG™ Security White Paper and culminating in the creation of the Grid Authentication and Authorization with Reliably Distributed Services (GAARDS) infrastructure. GAARDS provides services and tools for the administration and enforcement of security policy in an enterprise Grid. GAARDS was developed on top of the Globus Toolkit and extends the Grid Security Infrastructure (GSI) to provide enterprise services and administrative tools for:

1) Grid user management
2) Identity federation
3) Trust management
4) Group/VO management
5) Access control policy management and enforcement
6) Integration between existing security domains and the grid security domain

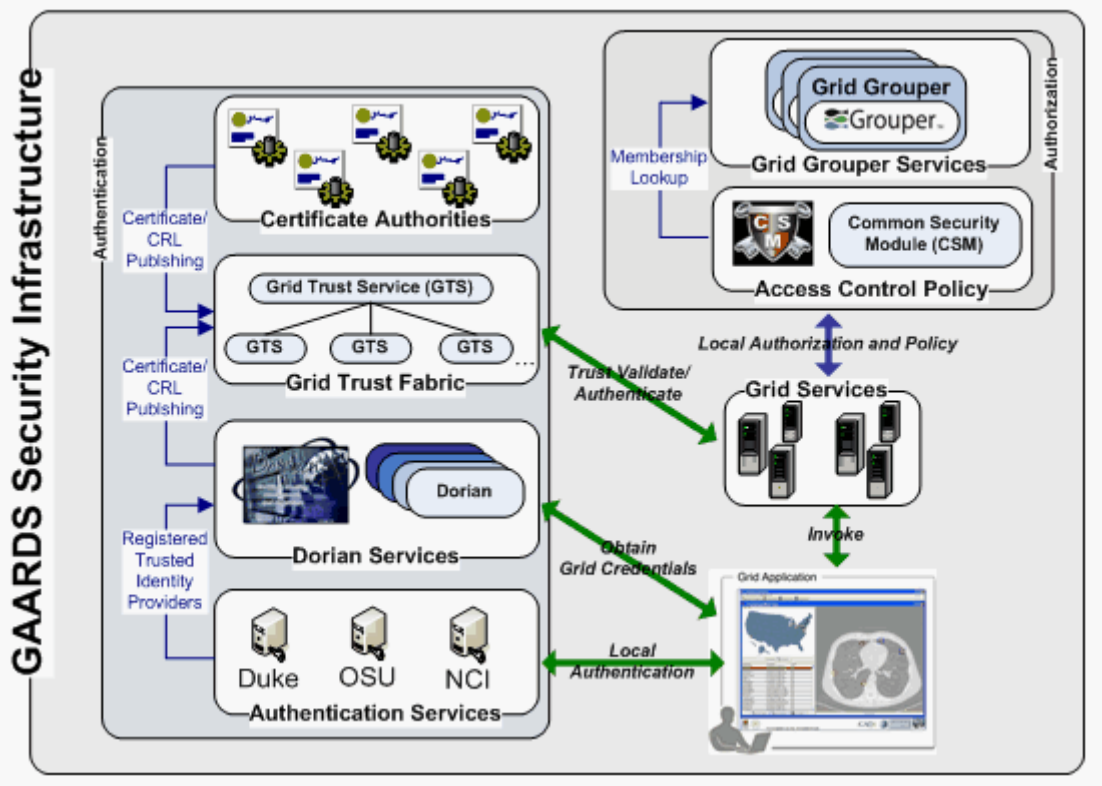


Figure 2-3 GAARDS Security Infrastructure

Figure 2-3 illustrates the GAARDS security infrastructure, in order for users/applications to communicate with secure services, they need grid credentials. Obtaining grid credentials requires having a Grid User Account. Dorian provides two methods for registering for a grid user account: 1) registering directly with Dorian 2) having an existing user account in another trusted security domain. In order to use an existing user account to obtain grid credentials, the existing credential provider must be registered in Dorian as a Trusted Identity Provider. It is anticipated that the majority of grid user accounts will be provisioned based on existing accounts. The

advantages to this approach are: 1) users can use their existing credentials to access the grid 2) administrators only need to manage a single account for a given user. To obtain grid credentials, Dorian requires proof (a digitally signed SAML assertion) that proves that the user locally authenticated. The GAARDS Authentication Service provides a framework for issuing SAML assertions for existing credential providers such that they may be used to obtain grid credentials from Dorian. The Authentication Service also provides a uniform authentication interface in on which applications can be built. Figure 2-3 illustrates the process for obtaining grid credentials, wherein the user/application first authenticates with their local credential provider via the Authentication Service and obtains a SAML assertion as proof they authenticated. They then use the SAML assertion provided by the Authentication Service to obtain grid credentials from Dorian. Assuming the local credential provider is registered with Dorian as a trusted identity provider and that the user's account is in good standing, Dorian will issue grid credentials to the user. It should be noted that the use of the Authentication Service is not required; an alternative mechanism for obtaining the SAML assertion required by Dorian can be used. If s user is registered directly with Dorian and not through an existing credential provider, they may contact Dorian directly for obtaining grid credentials.

Once a user has obtained grid credentials from Dorian they may invoke secure services. Upon receiving grid credentials from a user, a secure service authenticates the user to ensure that the user has presented valid grid credentials. Part of the grid authentication process is verifying that grid credentials presented were issued by a trusted grid credential provider (i.e. Dorian, other certificate authorities). The Grid Trust Service (GTS) maintains a federated trust fabric of all the trusted digital signers in the grid. Credential providers such as Dorian and grid certificate authorities are registered as trusted digital signers and regularly publish new information to the GTS. Grid services authenticate grid credentials against the trusted digital signers in a GTS (shown in Figure 2-3).

Once the user has been authenticated, a secure grid service next determines if a user is authorized to perform what they requested. Grid services have many different options available to them for performing authorization. It is important to note that all authorizing decisions are made by the local provider, but GAARDS provides some services and tools which facilitate some common authorization mechanisms. The GAARDS infrastructure provides two approaches which can each be used independently or can be used together. It is important to note any other authorization approach can be used in conjunction with the GAARDS authentication/trust infrastructure. The Grid Grouper service provides a group-based authorization solution for the Grid, wherein grid services and applications enforce authorization policy based on membership to groups defined and managed at the grid level. Grid services can use Grid Grouper directly to enforce their internal access control policies. Assuming the authorization policy is based on membership to groups provisioned by Grid Grouper; services can determine whether a caller is authorized by simply asking grid grouper whether the caller is in a given group. The caCORE Common Security Module (CSM), an existing component many providers are already using, is a more centralized approach to authorization. CSM is a tool for managing and enforcing access control policy centrally. CSM supports access control policies which can be based on membership to groups in Grid Grouper. Grid services that use CSM for authorization simply ask CSM with a user can perform a given action. Based on the access

control policy maintained in CSM, CSM decides whether or not a user is authorized. In Figure 2-3, the grid services defer the authorization to CSM. CSM enforces its group based access control policy by asking Grid Grouper whether the caller is a member of the groups specified in the policy, and enforces any other local data access policies defined in CSM.

Revolutionary Development

caGrid 1.0 represents a complete rewrite of caGrid to better support the requirements and current standards. Building on lessons learned from caGrid 0.5 and feedback from the community, it provides a large number of additional features, services, and vast improvements in caGrid technologies beyond what is described above. One such example is the development of a unified grid service authoring toolkit, dubbed Introduce. Introduce is an extensible framework and graphic workbench which provides an environment for the development and deployment of caBIG™ compatible grid enabled data and analytical services. The Introduce toolkit reduces the service developer's responsibilities, by abstracting away the need to manage the low level details of the WSRF specification and integration with the Globus Toolkit, allowing them to focus on implementing their business logic. Developers with existing caBIG™ Silver compatible services need only follow simple a wizard-like process for creating the "adapter" between the grid and their existing system. At the same time, extremely complex and powerful new services can be created. All caGrid developed core services were implemented with the Introduce toolkit.

Another significant feature provided by caGrid 1.0 is the addition of service support for orchestration of grid services using the industry standard Business Process Execution Language (BPEL). caGrid provides a workflow management service, enabling the execution and monitoring of BPEL-defined workflows in a secure grid environment. It is expected this work will provide the groundwork for a large number of powerful applications, enabling the harnessing of data and analytics made available as grid services. Another such higher-level support service made available in caGrid 1.0, is the federated query infrastructure. The caGrid Federated Query Infrastructure provides a mechanism to perform basic distributed aggregations and joins of queries over multiple data services. Working in collaboration with the Cancer Translational Research Informatics Platform (caTRIP) project, a caBIG™ funded project, an extension to the standard Data Service query language was developed to describe distributed query scenarios, as well as various enhancements to the Data Service query language itself. The Federated Query Infrastructure contains three main client-facing components: an API implementing the business logic of federated query support, a grid service providing remote access to that engine, and a grid service for managing status and results for queries that were invoked asynchronously using the query service.

Numerous improvements to the handling of large data sets and distributed information processing have been made. Support for the implantation of the WS-Enumeration (http://www.w3.org/Submission/WS-Enumeration/) standard has been implemented and added to the Globus Toolkit. This standard and its corresponding implementation provide the capability for a grid client to enumerate over results provided by a grid service (much like a grid-enabled *cursor*). This provides the framework necessary for clients to access large results from a service. This support has been integrated into the caGrid Data Service tooling providing a mechanism for iterating query results. Another aspect of caGrid expected to facilitate data

exchange in the grid is the initial work on the implementation of a grid wide object identifier framework. This work has been enabled by the integration of the Handle System® from Corporation for National Research Initiatives. caGrid 1.0 represents the initial release of this effort, and future improvements and support are planned for a future release. Additionally, the initial effort to standardize a “bulk data transport” interface for large data has been started in caGrid 1.0, which is intended to provide uniform mechanism by which clients may access data sets form arbitrary services. This initial work currently supports access via WS-Enumeration, WS-Transfer, and GridFTP. Additional enhancements and tooling are expected in a future release of caGrid, based on feedback from the user community.

Lastly, caGrid 1.0 represents a significant improvement in quality of caGrid, as a significant effort was placed on the development of unit, system, and integration testing. Several hundred unit tests are executed every time something is added to the caGrid code base, and a variety of builds and tests are run each night. Interested users may view results of these tests on a centralized dashboard (http://vandelay.bmi.ohio-state.edu:8081/caGrid-1.0/Dashboard/), execute these test frameworks locally, or leverage the testing framework during the development of their own services.

Chapter 3 caGrid Metadata Infrastructure

This chapter describes the caGrid metadata infrastructure and describes the Discovery API, caDSR Grid Service, and EVS APIs.

Topics in this chapter include:

- Metadata API Usage Overview on this page
- Discovery API Usage Overview on page 24
- caDSR Grid Service Usage Overview on page 45
- EVS API Usage Overview on page 57

Metadata API Usage Overview

The following link provides a reference to the technical architecture and design document(s) for caGrid Metadata:

http://gforge.nci.nih.gov/plugins/scmcvs/cvsweb.php/cagrid-1-0/Documentation/docs/metadata/caGrid-metadata-infrastructure-design.doc?cvsroot=cagrid-1-0

All caGrid services are expected to expose a standard set of service metadata. Details about this design and the specifics of the metadata can be found in the caGrid Metadata Design Document. This section describes the high-level API, which can be used to access and manipulate instances of this metadata. All the standard metadata models are representations, which can be used to programmatically interact with the models. Figure 3-1 and Figure 3-2 are the Standard Service (*ServiceMetadata*) and Data Service (*DomainModel*) metadata models respectively. The APIs described here can be used access these models from services, and serialize and deserialize them to and from XML. These methods complement the Discovery API. Once an EPR (End Point Reference) is returned from the Discovery API, these methods can be used to access and inspect the full metadata.

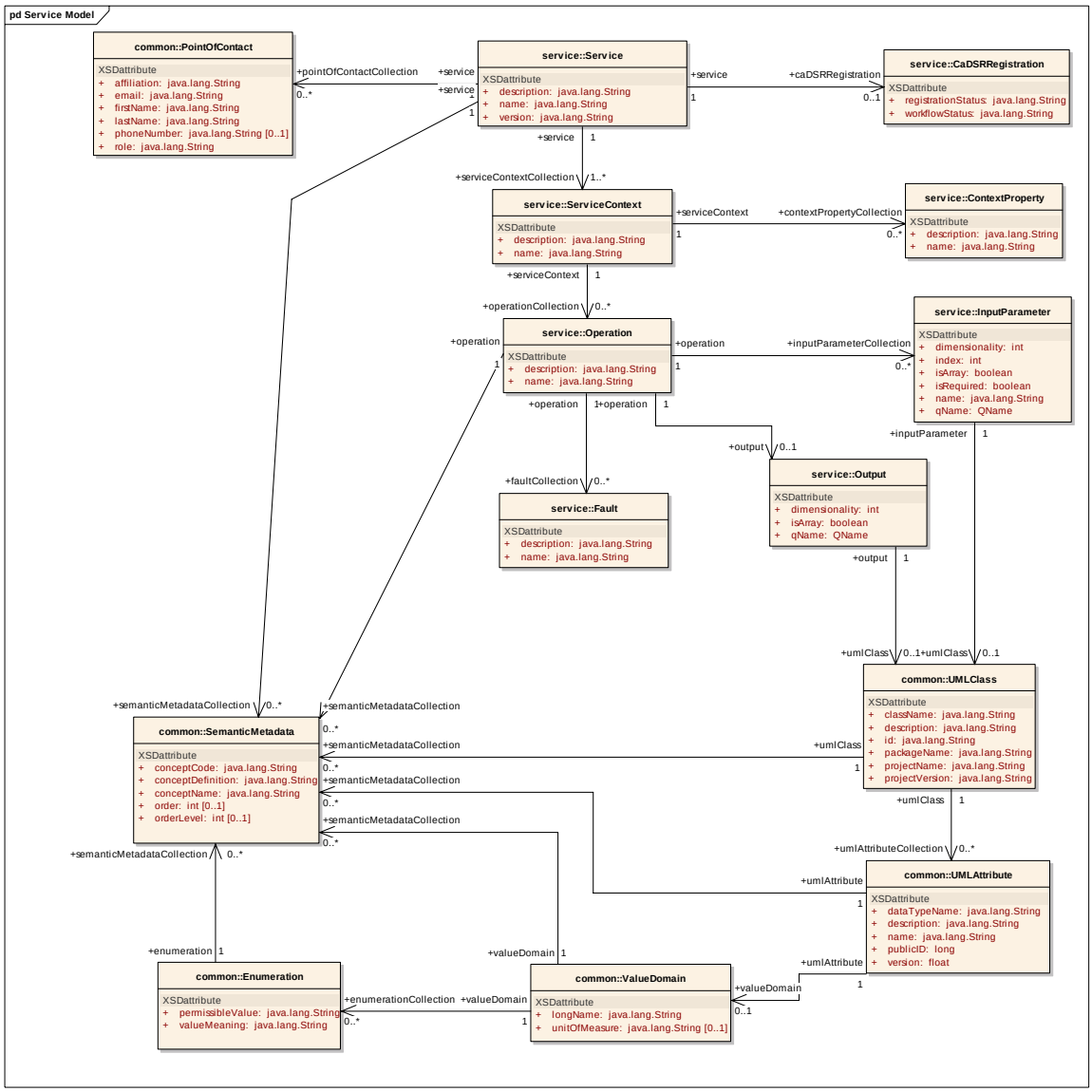


Figure 3-1 Service Model

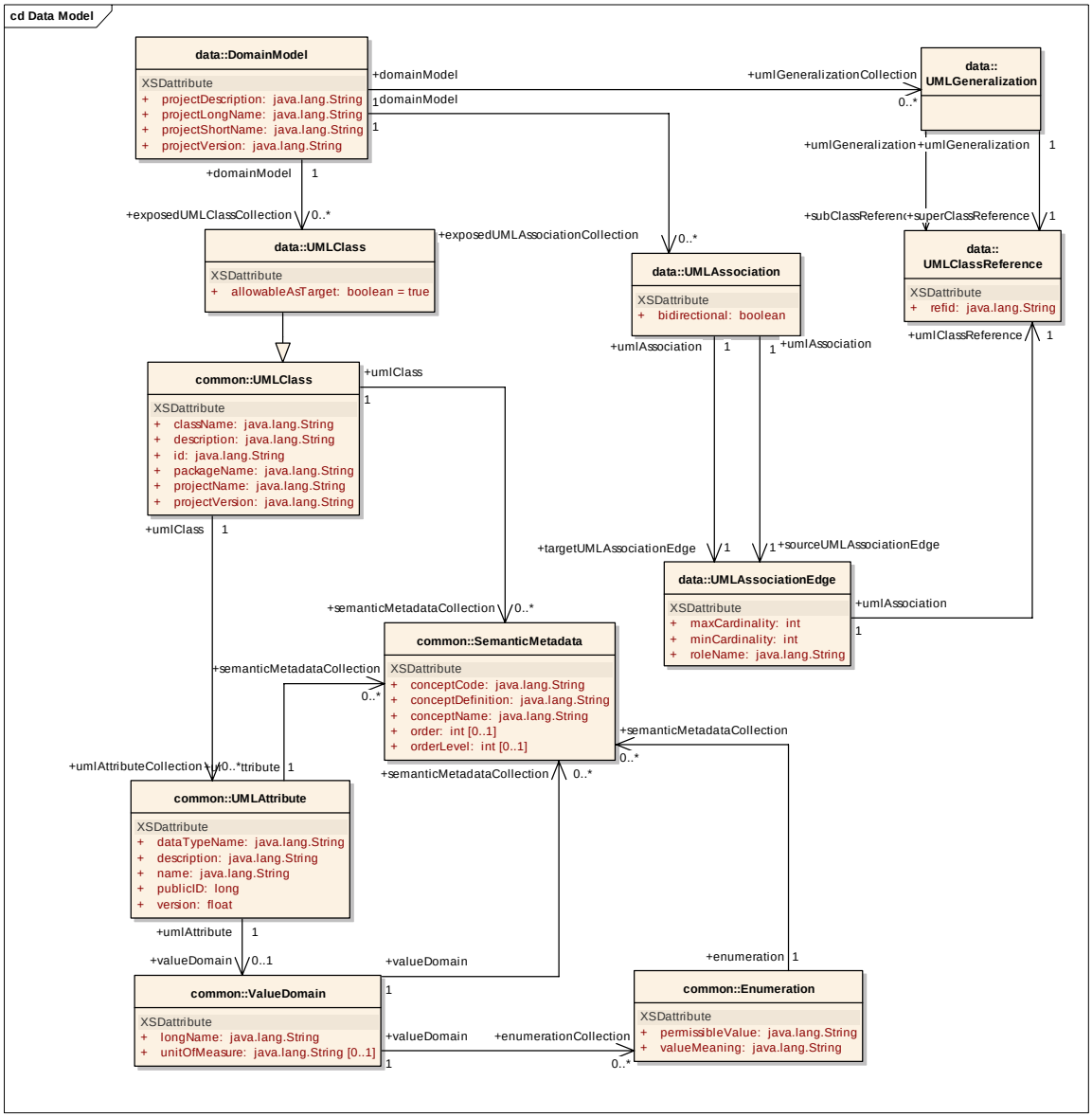


Figure 3-2 Data Service metadata model

The *ResourcePropertyHelper* API, not detailed here, is the lower level API, which can be used to directly gather information about ResourceProperties (which is how metadata is exposed in caGrid). The MetadataUtils, described here, leverage this API, and expose some of its exceptions. The possible exceptions generated by the metadata utility methods are detailed in Figure 3-3.

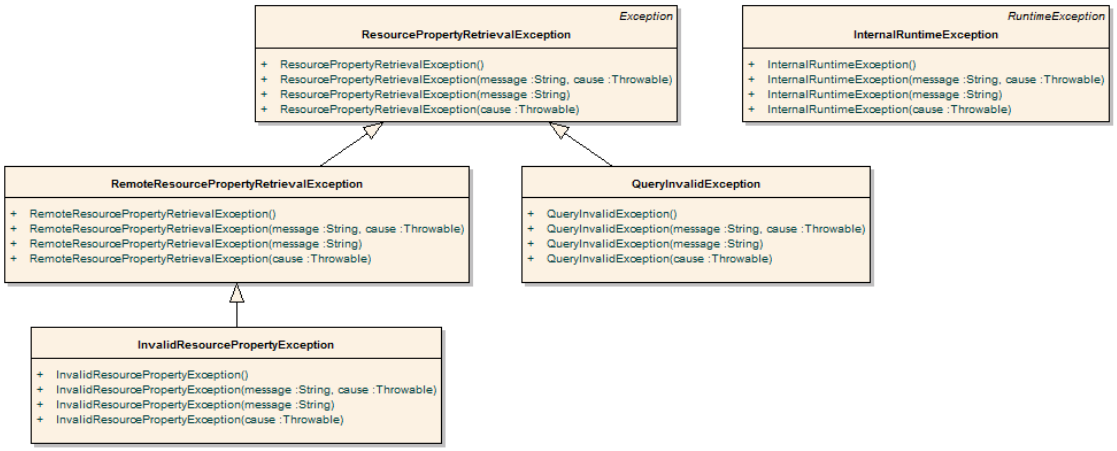


Figure 3-3 Metadata Exceptions

A non-discerning client may simply opt to catch *ResourcePropertyRetrievalException*, as it is the base-checked exception. An additional non-checked exception, *InternalRuntimeException*, can also be thrown but is solely used to represent an internal logic error in the APIs. It is not expected clients can "recover" from such an exception. As such, clients should not attempt to catch this runtime exception for any other reason than to mask the problem.

QueryInvalidException - is thrown if an invalid XPath query is issued. Problems originating from remote services are thrown in the subclass, *RemoteResourcePropertyRetrievalException*. During general use of the metadata utilities, this is the most likely exception clients may see, as it is thrown if a service is not properly exposing the proper metadata. Clients leveraging the lower level resource property APIs should take care to appropriately address each type of exception if they are communicating with services. For example, even though it is a caBIG requirement to expose the standard service metadata, clients should properly handle the case where it is not present. Asking for specific metadata that a service does not provide would yield an *InvalidResourcePropertyException*.

API Details

gov.nih.nci.cagrid.metadata.MetadataUtils

Member Function Documentation

static ServiceMetadata gov.nih.nci.cagrid.metadata.MetadataUtils.getServiceMetadata (EndpointReferenceType serviceEPR) throws InvalidResourcePropertyException, RemoteResourcePropertyRetrievalException, ResourcePropertyRetrievalException ***`[static]`***

Obtain the service metadata from the specified service.

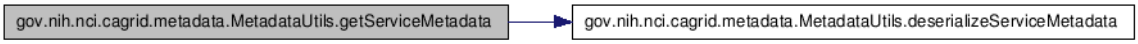
Parameters:

serviceEPR

Returns:

Exceptions:

InvalidResourcePropertyException
RemoteResourcePropertyRetrievalException
ResourcePropertyRetrievalException



static DomainModel gov.nih.nci.cagrid.metadata.MetadataUtils.getDomainModel (EndpointReferenceType serviceEPR) throws InvalidResourcePropertyException, RemoteResourcePropertyRetrievalException, ResourcePropertyRetrievalException `[static]`

Obtain the data service metadata from the specified service.

Parameters:

serviceEPR

Returns:

Exceptions:

InvalidResourcePropertyException
RemoteResourcePropertyRetrievalException
ResourcePropertyRetrievalException

static void gov.nih.nci.cagrid.metadata.MetadataUtils.serializeServiceMetadata (ServiceMetadata metadata, Writer writer) throws Exception `[static]`

Write the XML representation of the specified metadata to the specified writer. If either is null, an IllegalArgumentException will be thrown.

Parameters:

metadata
writer

Exceptions:

Exception

static ServiceMetadata gov.nih.nci.cagrid.metadata.MetadataUtils.deserializeServiceMetadata (Reader xmlReader) throws Exception `[static]`

Create an instance of the service metadata from the specified reader. The reader must point to a stream that contains an XML representation of the metadata. If the reader is null, an IllegalArgumentException will be thrown.

Parameters:

xmlReader

Returns:

Exceptions:

Exception

static void gov.nih.nci.cagrid.metadata.MetadataUtils.serializeDomainModel (DomainModel domainModel, Writer writer) throws Exception `[static]`

Write the XML representation of the specified metadata to the specified writer. If either is null, an IllegalArgumentException will be thrown.

Parameters:

domainModel
writer

Exceptions:

Exception

static DomainModel gov.nih.nci.cagrid.metadata.MetadataUtils.deserializeDomainModel (Reader xmlReader) throws Exception `[static]`

Create an instance of the data service metadata from the specified reader. The reader must point to a stream that contains an XML representation of the metadata. If the reader is null, an IllegalArgumentException will be thrown.

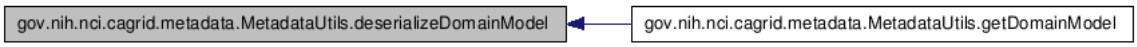
Parameters:

xmlReader

Returns:

Exceptions:

Exception



API Usage Examples

This section describes typical usage of the Metadata API. The exception handling shown in the code examples is not recommended practice, and is simplistic for demonstration purposes. The *MetadataUtils* class is the primary means of accessing and manipulating service metadata. It provides a number of static utility methods that can be directly invoked. This API provides an abstraction layer over lower-level APIs, specializing them to deal with the standard metadata types. Clients wishing to work with custom (or non-standard) metadata, need to use the lower-level APIs, and can consult the source code of the *MetadataUtils* class for guidance.

Accessing Metadata from a Service

In order to access a service's metadata, an End Point Reference pointing to the service must be provided. This can be obtained as a direct result of an invocation of a discovery method from the Discovery API, or manually constructed by specifying the service's *Address*. Examples of both can be found in the Discovery API Usage Overview starting on page 24.

As caBIG requires that standard metadata be made publicly available, client credentials are not necessary for invocation of these methods.

The first example, shown in Figure 3-4, demonstrates accessing a service's standard *ServiceMetadata*, which is common to all caGrid services. As described above, the first step is to obtain an appropriate EPR (line 1). Given this EPR, the *MetadataUtils*'s *getServiceMetadata* method, shown on line 4 in Figure 3-4, can be used to obtain the bean representation of the metadata. Upon successful completion of this method, the fully populated bean can be inspected to obtain the information of interest. Several exceptions, subclassed from the base *ResourcePropertyRetrievalException*, can be thrown by this operation. A non-discriminating client may choose to simply handle this base exception. Additional details on the other exceptions, and why they may be thrown, are described in the Metadata API Usage Overview on page 15, as well as the javadoc of the APIs.

```
1    EndpointReferenceType serviceEPR = /* Some service's EPR */
2    try {
3        ServiceMetadata serviceMetadata =
4            MetadataUtils.getServiceMetadata(serviceEPR);
5    } catch (InvalidResourcePropertyException e) {
6        // TODO handle the case where the service doesn't expose
7        // standard metadata
8        e.printStackTrace();
9    } catch (RemoteResourcePropertyRetrievalException e) {
10       // TODO handle the case where there was some other problem
11       // communicating with the service (unavailable, untrusted, etc)
12       e.printStackTrace();
13   } catch (ResourcePropertyRetrievalException e) {
14       // TODO handle all other general error (such as inability to
15       // deserialize the result)
16       e.printStackTrace();
17   }
```

Figure 3-4 Accessing standard service metadata

The process for accessing data service *DomainModel* metadata, shown in Figure 3-5, is the same as accessing standard metadata. Once the metadata is obtained, in line 3, it can be inspected, as shown in line 4 where the long name of the project being exposed by the data service is printed to the console.

```
1    EndpointReferenceType serviceEPR = /* Some service's EPR */
2    try {
3        DomainModel domainModel = MetadataUtils.getDomainModel(serviceEPR);
4        System.out.println(domainModel.getProjectLongName());
5    } catch (InvalidResourcePropertyException e) {
6        // TODO handle the case where the service doesn't expose
7        // standard data service metadata
8        e.printStackTrace();
9    } catch (RemoteResourcePropertyRetrievalException e) {
10       // TODO handle the case where there was some other problem
11       // communicating with the service (unavailable, untrusted, etc)
12       e.printStackTrace();
13   } catch (ResourcePropertyRetrievalException e) {
14       // TODO handle all other general error (such as inability to
15       // deserialize the result)
16       e.printStackTrace();
```

Figure 3-5 Accessing standard data service metadata

Processing Metadata as XML

In addition to accessing metadata from services, the *MetadataUtils* provide the capability to read and write metadata instances as XML documents. This can not only be useful for storage and display of metadata, but also grid service's expose metadata as XML, and these methods provide a way to inspect the metadata in object (bean) form.

In Figure 3-6, example code is shown that saves an instance of standard service metadata to a file named `seviceMetadata.xml`. The metadata instance, defined in line 1, can be acquired

using code similar to that shown in Figure 3-6, or by some other mechanism. The *serializeServiceMetadata* method can be then passed this instance, and an instance of the *java.io.Writer* interface, as shown on line 11. Any Writer implementation works, but the example below shows using a *FileWriter*, on line 5, to write the metadata to the specified file. After the *MetadataUtils* have been used to write the metadata to XML, the *Writer* used should be closed, as shown on line 18. Though not shown, a similar method, *serializeDomainModel*, exists for writing data service metadata to XML; its usage pattern is the same.

```
1    ServiceMetadata serviceMetadata = /* Some service's Metadata */
2    Writer writer = null;
3    try {
4        //Any Writer implementation will work
5        writer = new FileWriter("serviceMetadata.xml");
6    } catch (IOException e) {
7        // TODO Handle problem setting up writer
8        e.printStackTrace();
9    }
10   try {
11       MetadataUtils.serializeServiceMetadata(serviceMetadata, writer);
12   } catch (Exception e) {
13       // TODO handle problem serializing to the specified writer
14       e.printStackTrace();
15   }
16   try {
17       //be sure to close your Writer
18       writer.close();
19   } catch (IOException e) {
20       // TODO handle problem closing the writer
21       e.printStackTrace();
22   }
```

Figure 3-6 Serializing metadata to a file

As a complement to the serialization methods described and shown above, deserialization methods also exist which read XML representations of metadata and return appropriately populated metadata beans. In Figure 3-7, example code is shown which populates a new *ServiceMetadata* instance from an XML representation stored in a file named `serviceMetadata.xml`. This code, used in conjunction with the previous example, reconstitutes the original metadata instance. Similar to the serialization methods that use a *java.io.Writer*, the deserialization methods use a *java.io.Reader* to read the XML representation. In the example below, a *FileReader* is used on line 4. This *Reader* is then passed to the *deserializeServiceMetadata* method on line 11, and the populated *ServiceMetadata* instance is returned. As with the *Writer* instance in the serialization methods, the *Reader* instance should be closed once used (as shown on line 18). Though not shown, a similar method, de*serializeDomainModel*, exists for reading data service metadata from XML; its usage pattern is the same.

```
1    Reader reader = null;
2    try {
3        // Any Reader will work
4        reader = new FileReader("serviceMetadata.xml");
5    } catch (FileNotFoundException e) {
6        // TODO handle the case where the file is not found
7        e.printStackTrace();
8    }
9    try {
10       ServiceMetadata serviceMetadata =
11           MetadataUtils.deserializeServiceMetadata(reader);
12   } catch (Exception e) {
13       // TODO handle problem deserializing from the reader
14       e.printStackTrace();
15   }
16   try {
17       //be sure to close your reader
18       reader.close();
19   } catch (IOException e) {
20       // TODO handle problem closing the reader
21       e.printStackTrace();
22   }
```

Figure 3-7 Deserializing metadata from a file

Discovery API Usage Overview

The Discovery API provides an abstraction over the standard query operations used to query the Index Service. It provides a number of operations that can be used to discover services of interest. The basic process of use is to construct an instance of the *DiscoveryClient*, optionally specifying the End Point Reference (EPR) of the Index Service to query, and then invoking the appropriate discovery methods. Each method returns an array of EPRs of the matching appropriate services. These returned EPRs can then be used to invoke the services, or ask them for their metadata for further discrimination. It is worth noting that the Index Service, as an aggregated source of distributed information, inherently operates on out of date information. It is possible that services that are running do not yet have their metadata aggregated in the Index Service, and it is possible that services present in the Index Service have recently been taken down. caGrid attempts to strike a balance between performance and reliability of information in the Index Service. The information returned by the Discovery API should be accurate within a few minutes, but applications building upon it should be aware of this, and should not assume a service in the Index Service will always be available when it is invoked.

The DiscoveryClient uses the lower level "metadata utils" project to communicate with the Index Service. It exposes the exceptions generated from this lower level API, instead of wrapping them with discovery-specific exceptions. The possible exceptions which discovery methods can throw are detailed below in Figure 3-8. A non-discerning client may simply opt to catch *ResourcePropertyRetrievalException*, as it is the base checked exception. An additional non-checked exception, *InternalRuntimeException*, can also be thrown, but it is solely used to represent an internal logic error in the APIs and so it is not expected clients can "recover" from

such an exception. As such, clients should not attempt to catch this runtime exception for any other reason than to mask the problem. Generic problems caused by the *DiscoveryClient* itself are thrown in the base *ResourcePropertyRetrievalException*. A subclass of it, *QueryInvalidException*, is thrown if an invalid XPath query is issued. Unless the DiscoveryClient is extended, it is not expected that clients should encounter this. Problems originating from remote services are thrown in the subclass, *RemoteResourcePropertyRetrievalException*. During general use of the metadata utilities, this is the most likely exception clients may see, as it is thrown if a service is not properly exposing the proper metadata. In the context of the DiscoveryClient, it is not expected clients should experience any exceptions unless there is an issue with the Index Service. However, clients leveraging the lower level APIs should take more care to appropriately address each type of exception if they are communicating with other (community provided) services. For example, even though it is a caBIG requirement to expose the standard service metadata, clients should properly handle the case where it is not present. Asking for specific metadata that a service does not provide would yield an *InvalidResourcePropertyException*.

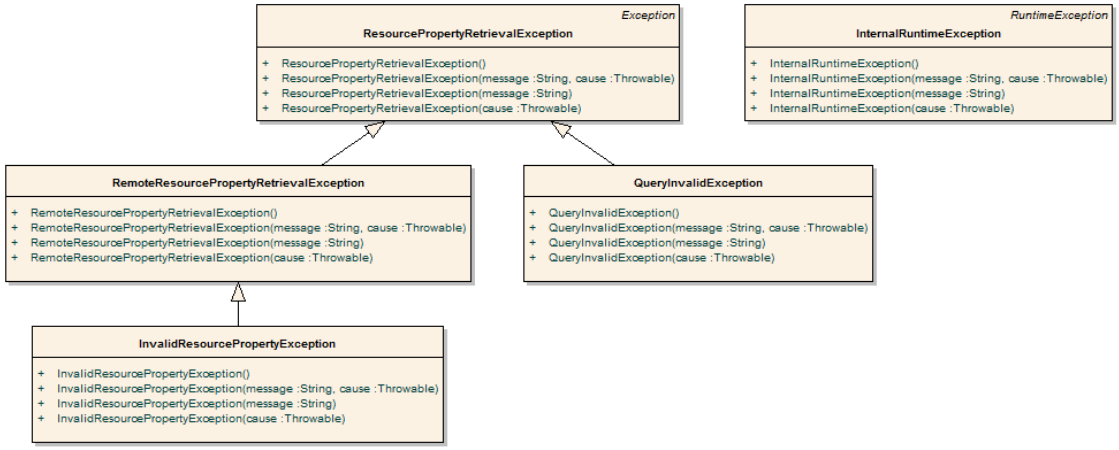


Figure 3-8 Metadata exceptions

While the methods in the API are designed around the caGrid standard metadata, it is also acceptable to have services register additional domain or application specific metadata to the Index Service. The Discovery API is designed for easy extensibility, such that additional application or domain specific discovery scenarios can be provided to compliment such additional metadata. The “business logic” of the *DiscoveryClient*, consists almost entirely of constructing appropriate XPath queries over the appropriate metadata, and leveraging lower-level APIs to actually invoke the queries. These lower-level APIs are made available to extenders of the client, such that they need only construct appropriate XPath queries to implement additional discovery scenarios.

The *DiscoveryClient* uses commons-logging to log general and debugging information. If configured to DEBUG level, the client prints out the XPaths it is sending to the Index Service, which may facilitate the creation of new discovery operations, or help track down problems.

As with most Globus clients, a properly configured *client-config.wsdd* file must be accessible by the underlying Axis engine. The simplest way to do this is to either run with your

$GLOBUS_LOCATION as the "working directory," add $GLOBUS_LOCATION to your classpath, or copy $GLOBUS_LOCATION/client-config.wsdd to your working directory or classpath. If you don't do this, you will most likely see an exception similar to that shown in Figure 3-9, when you run the DiscoveryClient.

```
gov.nih.nci.cagrid.metadata.exceptions.RemoteResourcePropertyRetrievalException:
org.xml.sax.SAXException:SimpleDeserializer encountered a child element, which is NOT
```

Figure 3-9 Common DiscoveryClient exception

API Details

gov.nih.nci.cagrid.discovery.client.DiscoveryClient

DiscoveryClient represents the base discovery API. The client should be constructed passing a URL of an Index Service. Services can then be discovered by calling the discover methods and passing in the necessary criteria. The methods all return an EndPointReferenceType[]. See the main method for examples. This should be extended to provide specialized service-type discovery (beyond data services).

Constructor Documentation

gov.nih.nci.cagrid.discovery.client.DiscoveryClient.DiscoveryClient () throws MalformedURIException

Uses the Default Index Service

Exceptions:

MalformedURIException if the Default Index Service is invalid

DiscoveryClient.java.gov.nih.nci.cagrid.discovery.client.DiscoveryClient.DiscoveryClient (EndpointReferenceType *indexEPR*)

Uses the specified Index Service

Parameters:

iemndexEPR the EPR to the Index Service to use

Member Function Documentation

EndpointReferenceType []

gov.nih.nci.cagrid.discovery.client.DiscoveryClient.getAllServices (boolean

***requireMetadataCompliance*) throws RemoteResourcePropertyRetrievalException, QueryInvalidException, ResourcePropertyRetrievalException**

Query the registry for all registered services

Parameters:

requireMetadataCompliance if true, only services providing the standard metadata will be returned. Otherwise, all services registered will be returned, regardless of whether or not any metadata has been aggregated.

Returns:

EndpointReferenceType[] contain all registered services

Exceptions:

ResourcePropertyRetrievalException
QueryInvalidException
RemoteResourcePropertyRetrievalException

EndpointReferenceType []
gov.nih.nci.cagrid.discovery.client.DiscoveryClient.discoverServicesBySearchString (String *searchString*) throws RemoteResourcePropertyRetrievalException, QueryInvalidException, ResourcePropertyRetrievalException

Searches ALL metadata to find occurrence of the given string. The search string is case-sensitive.

Parameters:

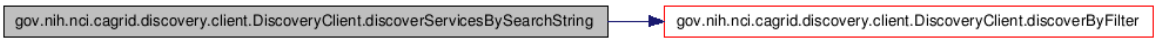
searchString the search string.

Returns:

EndpointReferenceType[] matching the criteria

Exceptions:

ResourcePropertyRetrievalException
QueryInvalidException
RemoteResourcePropertyRetrievalException



EndpointReferenceType []

gov.nih.nci.cagrid.discovery.client.DiscoveryClient.discoverServicesByResearchCenter

(String *centerName*) throws RemoteResourcePropertyRetrievalException,

QueryInvalidException, ResourcePropertyRetrievalException

Searches research center info to find services provided by a given cancer center.

Parameters:

centerName research center name

Returns:

EndpointReferenceType[] matching the criteria

Exceptions:

ResourcePropertyRetrievalException
QueryInvalidException
RemoteResourcePropertyRetrievalException

EndpointReferenceType []

gov.nih.nci.cagrid.discovery.client.DiscoveryClient.discoverServicesByPointOfContact

(PointOfContact *contact*) throws RemoteResourcePropertyRetrievalException,

QueryInvalidException, ResourcePropertyRetrievalException

Searches to find services that have the given point of contact associated with them. Any fields set on the point of contact are checked for a match. For example, you can set only the lastName, and only it will be checked, or you can specify several fields and they all must be equal.

Parameters:

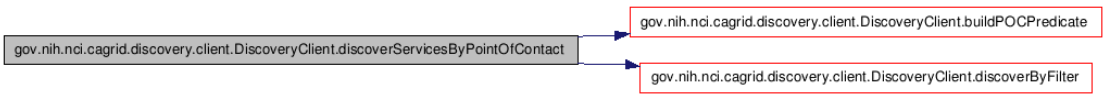
contact point of contact

Returns:

EndpointReferenceType[] matching the criteria

Exceptions:

ResourcePropertyRetrievalException
QueryInvalidException
RemoteResourcePropertyRetrievalException



EndpointReferenceType []

gov.nih.nci.cagrid.discovery.client.DiscoveryClient.discoverServicesByName (String *serviceName*) throws RemoteResourcePropertyRetrievalException, QueryInvalidException, ResourcePropertyRetrievalException

Searches to find services that have a given name.

Parameters:

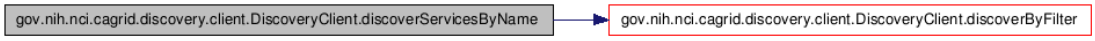
serviceName The service's name

Returns:

EndpointReferenceType[] matching the criteria

Exceptions:

ResourcePropertyRetrievalException
QueryInvalidException
RemoteResourcePropertyRetrievalException



EndpointReferenceType []

gov.nih.nci.cagrid.discovery.client.DiscoveryClient.discoverServicesByConceptCode (String *conceptCode*) throws RemoteResourcePropertyRetrievalException, QueryInvalidException, ResourcePropertyRetrievalException

Searches to find services based on the given concept code.

Parameters:

conceptCode A concept code the service is based upon.

Returns:

EndpointReferenceType[] matching the criteria

Exceptions:

ResourcePropertyRetrievalException
QueryInvalidException
RemoteResourcePropertyRetrievalException

EndpointReferenceType []

gov.nih.nci.cagrid.discovery.client.DiscoveryClient.discoverServicesByOperationName (String *operationName*) throws RemoteResourcePropertyRetrievalException, QueryInvalidException, ResourcePropertyRetrievalException

Searches to find services that have a given operation.

Parameters:

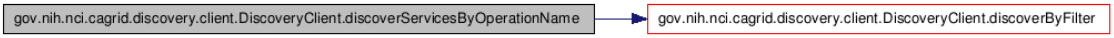
operationName The operation's name

Returns:

EndpointReferenceType[] matching the criteria

Exceptions:

ResourcePropertyRetrievalException
QueryInvalidException
RemoteResourcePropertyRetrievalException



EndpointReferenceType []

gov.nih.nci.cagrid.discovery.client.DiscoveryClient.discoverServicesByOperationInput (UMLClass *clazzPrototype*) throws RemoteResourcePropertyRetrievalException, QueryInvalidException, ResourcePropertyRetrievalException

Searches to find services that have an operation defined that takes the given UMLClass as input. Any fields set on the UMLClass are checked for a match. For example, you can set only the packageName, and only it will be checked, or you can specify several fields and they all must be equal.

NOTE: Only attributes of the UMLClass are examined (associated objects (e.g. UMLAttributeCollection and SemanticMetadataCollection) are ignored).

Parameters:

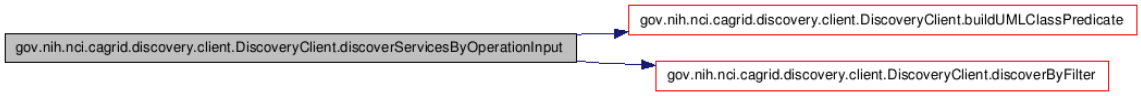
clazzPrototype The prototype UMLClass

Returns:

EndpointReferenceType[] matching the criteria

Exceptions:

ResourcePropertyRetrievalException
QueryInvalidException
RemoteResourcePropertyRetrievalException



EndpointReferenceType []

gov.nih.nci.cagrid.discovery.client.DiscoveryClient.discoverServicesByOperationOutput

(UMLClass *clazzPrototype*) throws RemoteResourcePropertyRetrievalException,

QueryInvalidException, ResourcePropertyRetrievalException

Searches to find services that have an operation defined that produces the given UMLClass. Any fields set on the UMLClass are checked for a match. For example, you can set only the packageName, and only it will be checked, or you can specify several fields and they all must be equal.

NOTE: Only attributes of the UMLClass are examined (associated objects (e.g. UMLAttributeCollection and SemanticMetadataCollection) are ignored).

Parameters:

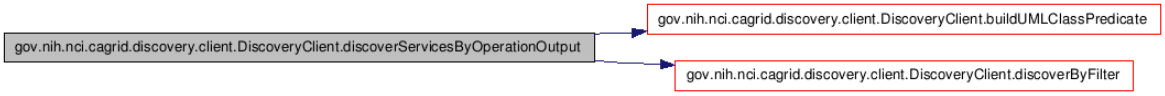
clazzPrototype The prototype UMLClass

Returns:

EndpointReferenceType[] matching the criteria

Exceptions:

ResourcePropertyRetrievalException
QueryInvalidException
RemoteResourcePropertyRetrievalException



EndpointReferenceType []

gov.nih.nci.cagrid.discovery.client.DiscoveryClient.discoverServicesByOperationClass

(UMLClass *clazzPrototype*) throws RemoteResourcePropertyRetrievalException,

QueryInvalidException, ResourcePropertyRetrievalException

Searches to find services that have an operation defined that produces the given UMLClass or takes it as input. Any fields set on the UMLClass are checked for a match. For example, you can set only the packageName, and only it will be checked, or you can specify several fields and they all must be equal.

Note: Only attributes of the UMLClass are examined (associated objects (e.g. UMLAttributeCollection and SemanticMetadataCollection) are ignored).

Parameters:

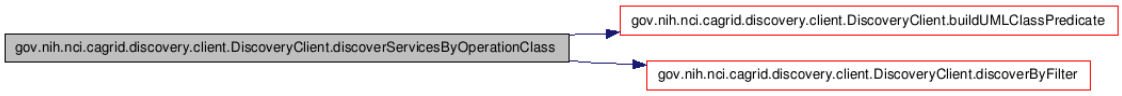
clazzPrototype The prototype UMLClass

Returns:

EndpointReferenceType[] matching the criteria

Exceptions:

ResourcePropertyRetrievalException
QueryInvalidException
RemoteResourcePropertyRetrievalException



EndpointReferenceType []

gov.nih.nci.cagrid.discovery.client.DiscoveryClient.discoverServicesByOperationConceptCode (String *conceptCode*) throws RemoteResourcePropertyRetrievalException, QueryInvalidException, ResourcePropertyRetrievalException

Searches to find services that have an operation based on the given concept code

Parameters:

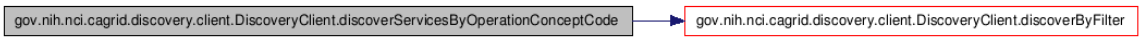
conceptCode The concept to look for

Returns:

EndpointReferenceType[] matching the criteria

Exceptions:

ResourcePropertyRetrievalException
QueryInvalidException
RemoteResourcePropertyRetrievalException



EndpointReferenceType []

gov.nih.nci.cagrid.discovery.client.DiscoveryClient.discoverServicesByDataConceptCode (String *conceptCode*) throws RemoteResourcePropertyRetrievalException, QueryInvalidException, ResourcePropertyRetrievalException

Searches to find services that have an operation defined that produces the or takes as input, a Class with an attribute , attribute value domain , enumerated value meaning, or the class itself based on the given concept code.

Parameters:

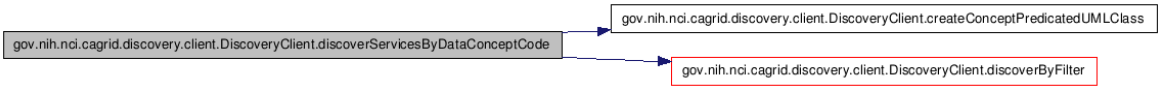
conceptCode The concept to look for

Returns:

EndpointReferenceType[] matching the criteria

Exceptions:

ResourcePropertyRetrievalException
QueryInvalidException
RemoteResourcePropertyRetrievalException



EndpointReferenceType []

gov.nih.nci.cagrid.discovery.client.DiscoveryClient.discoverServicesByPermissibleValue (String *value*) throws RemoteResourcePropertyRetrievalException, QueryInvalidException, ResourcePropertyRetrievalException

Searches to find services that have an operation defined that produces or takes as input, a Class with an attribute allowing the given value.

Parameters:

value The permissible value to look for

Returns:

EndpointReferenceType[] matching the criteria

Exceptions:

ResourcePropertyRetrievalException
QueryInvalidException

RemoteResourcePropertyRetrievalException

String

gov.nih.nci.cagrid.discovery.client.DiscoveryClient.createPermissibleValuePredicatedUM

LClass (String *value*, boolean *isDataService*) `[protected]`

Creates a UMLClass step that is predicated to contain either an attribute, attribute value domain, enumerated value meaning, or the class itself based on that concept.

Parameters:

conceptCode the code to look for

Returns:

String

gov.nih.nci.cagrid.discovery.client.DiscoveryClient.createConceptPredicatedUMLClass

(String *conceptCode*, boolean *isDataService*) `[protected]`

Creates a UMLClass step that is predicated to contain either an attribute, attribute value domain, enumerated value meaning, or the class itself based on that concept.

Parameters:

conceptCode the code to look for

Returns:

EndpointReferenceType []

gov.nih.nci.cagrid.discovery.client.DiscoveryClient.getAllDataServices () throws

RemoteResourcePropertyRetrievalException, QueryInvalidException,

ResourcePropertyRetrievalException

Query the registry for all registered data services

Returns:

EndpointReferenceType[] contain all registered services

Exceptions:

ResourcePropertyRetrievalException

QueryInvalidException
RemoteResourcePropertyRetrievalException

EndpointReferenceType []

gov.nih.nci.cagrid.discovery.client.DiscoveryClient.discoverDataServicesByDomainMode

l (String *modelName*) throws RemoteResourcePropertyRetrievalException,

QueryInvalidException, ResourcePropertyRetrievalException

Searches to find data services that are exposing a subset of given domain (short name or long name).

Parameters:

modelName The model to look for

Returns:

EndpointReferenceType[] matching the criteria

Exceptions:

ResourcePropertyRetrievalException
QueryInvalidException
RemoteResourcePropertyRetrievalException

EndpointReferenceType []

gov.nih.nci.cagrid.discovery.client.DiscoveryClient.discoverDataServicesByModelConce

ptCode (String *conceptCode*) throws RemoteResourcePropertyRetrievalException,

QueryInvalidException, ResourcePropertyRetrievalException

Searches to find data services that expose a Class with an attribute , attribute value domain , enumerated value meaning, or the class itself based on the given concept code.

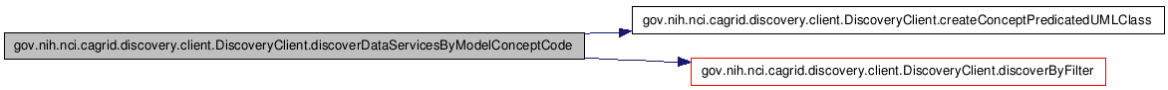
Parameters:

conceptCode The concept to look for

Returns:

Exceptions:

RemoteResourcePropertyRetrievalException
QueryInvalidException
ResourcePropertyRetrievalException



EndpointReferenceType []

gov.nih.nci.cagrid.discovery.client.DiscoveryClient.discoverDataServicesByExposedClass (gov.nih.nci.cagrid.metadata.dataservice.UMLClass *clazzPrototype*) throws RemoteResourcePropertyRetrievalException, QueryInvalidException, ResourcePropertyRetrievalException

Searches for data services that expose the given class. Any fields set on the UMLClass are checked for a match. For example, you can set only the packageName, and only it will be checked, or you can specify several fields and they all must be equal.

Note: Only attributes of the UMLClass are examined (associated objects (e.g. UMLAttributeCollection and SemanticMetadataCollection) are ignored).

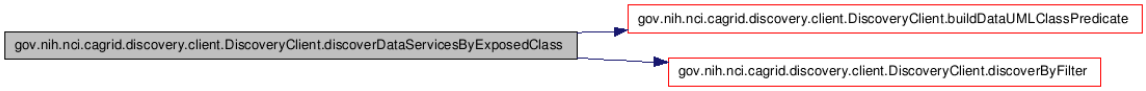
Parameters:

clazzPrototype The prototype UMLClass
clazzPrototype

Returns:

Exceptions:

RemoteResourcePropertyRetrievalException
QueryInvalidException
ResourcePropertyRetrievalException



EndpointReferenceType []

gov.nih.nci.cagrid.discovery.client.DiscoveryClient.discoverDataServicesByPermissibleValue (String *permissibleValue*) throws RemoteResourcePropertyRetrievalException, QueryInvalidException, ResourcePropertyRetrievalException

Searches for data services that expose a class with an attribute allowing the given value.

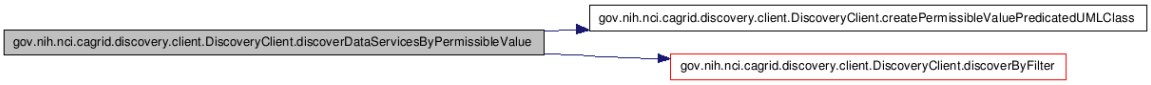
Parameters:

value The permissible value to look for

Returns:

Exceptions:

RemoteResourcePropertyRetrievalException
QueryInvalidException
ResourcePropertyRetrievalException



EndpointReferenceType []

gov.nih.nci.cagrid.discovery.client.DiscoveryClient.discoverDataServicesByAssociations WithClass (gov.nih.nci.cagrid.metadata.dataservice.UMLClass *clazzPrototype*) throws RemoteResourcePropertyRetrievalException, QueryInvalidException, ResourcePropertyRetrievalException

Searches for data services that expose an association to or from the given class. Any fields set on the UMLClass are checked for a match. For example, you can set only the packageName, and only it will be checked, or you can specify several fields and they all must be equal.

Note: Only attributes of the UMLClass are examined (associated objects (e.g. UMLAttributeCollection and SemanticMetadataCollection) are ignored).

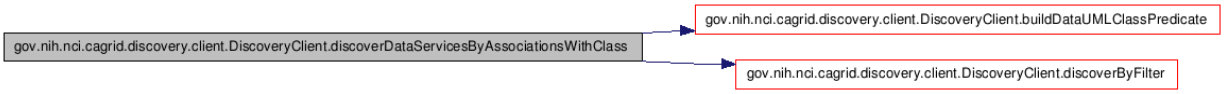
Parameters:

clazzPrototype

Returns:

Exceptions:

RemoteResourcePropertyRetrievalException
QueryInvalidException
ResourcePropertyRetrievalException



static String gov.nih.nci.cagrid.discovery.client.DiscoveryClient.buildPOCPredicate (PointOfContact *contact*) `[static, protected]`

Builds up a predicate for a PointOfContact, based on the prototype passed in.

Parameters:

contact the prototype POC

Returns:

"*" if the prototype has no non-null or non-whitespace values, or the predicate necessary to match all values.

static String gov.nih.nci.cagrid.discovery.client.DiscoveryClient.buildUMLClassPredicate (UMLClass *clazz*) `[static, protected]`

Builds up a predicate for a UMLClass, based on the prototype passed in.

Note: Only attributes of the UMLClass are examined (associated objects (e.g. UMLAttributeCollection and SemanticMetadataCollection) are ignored).

Parameters:

clazz the prototype UMLClass

Returns:

"*" if the prototype has no non-null or non-whitespace values, or the predicate necessary to match all values.

static String gov.nih.nci.cagrid.discovery.client.DiscoveryClient.buildDataUMLClassPredicate (gov.nih.nci.cagrid.metadata.dataservice.UMLClass *clazz*) `[static, protected]`

static String gov.nih.nci.cagrid.discovery.client.DiscoveryClient.addNonNullPredicate (String *name*, String *value*, boolean *isAttribute*) `[static, protected]`

Parameters:

name the element or attribute name to check
value the value to add the predicate filter against if this is null or whitespace only, no predicated is added.
isAttribute whether or not name represents an attribute or element

Returns:

"" or the specified predicate (prefixed with " and ")

EndpointReferenceType []

gov.nih.nci.cagrid.discovery.client.DiscoveryClient.discoverByFilter (String *xpathPredicate*) throws RemoteResourcePropertyRetrievalException, QueryInvalidException, ResourcePropertyRetrievalException `[protected]`

Applies the specified predicate to the common path in the Index Service's Resource Properties to return registered services' EPRs that match the predicate.

Parameters:

xpathPredicate predicate to apply to the "Entry" in Index Service

Returns:

EndpointReferenceType[] of matching services @

Exceptions:

ResourcePropertyRetrievalException
QueryInvalidException
RemoteResourcePropertyRetrievalException

String gov.nih.nci.cagrid.discovery.client.DiscoveryClient.translateXPath (String

***xpathPredicate*) `[protected]`**

Adds the common Index RP Entry filter, and translates the xpath to IndexService friendly XPath.

Parameters:

xpathPredicate

Returns:

the modified xpath

EndpointReferenceType gov.nih.nci.cagrid.discovery.client.DiscoveryClient.getIndexEPR ()

Gets the EPR of the Index Service being used.

void gov.nih.nci.cagrid.discovery.client.DiscoveryClient.setIndexEPR (EndpointReferenceType *indexEPR*)

Sets the EPR of the Index Service to use.

Parameters:

indexEPR the EPR of the Index Service to use.

static void gov.nih.nci.cagrid.discovery.client.DiscoveryClient.main (String[] *args*) `[static]`

testing stub

Parameters:

args optional URL to Index Service to query.

gov.nih.nci.cagrid.discovery.XPathUtils

Member Function Documentation

static String gov.nih.nci.cagrid.discovery.XPathUtils.translateXPath (String prefixedXpath, Map namespaces) `[static]`

This utility takes an XPath that uses namespace prefixes (such as /a:B/a:C) and converts it to one without prefixes, by using the appropriate operators instead (such as /*[namespace-uri()='http://DOMAIN.COM/SCHEMA' and local-name()='B']/*[namespace-uri()='http://DOMAIN.COM/SCHEMA' and local-name()='C']). The only conceivable use for this function is to write sane XPath and convert it to the insane XPath Globus index service supports.

Note: This is not perfect. The known limitations are: 1) its overly aggressive, and will replace QName-looking string literals, 2) it won't work if you have namespaces attributes 3) it will silently not replace any QNames that you haven't supplied a prefix mapping for

Parameters:

prefixedXpath An xpath [optionally] using namespace prefixes in nodetests
namespaces A Map<String,String> keyed on namespace prefixes to resolve in the xpath, where the value is the actual namespace that should be used.

Returns:

a converted, conformant, xpath

API Usage Examples

This section describes typical usage of the Discovery API. The exception handling shown in the code examples is not recommended practice, and is simplistic for demonstration purposes. Additional examples can be found in the source code of the discovery project, in the main of the DiscoveryClient itself, as well as in the test source directory.

The main method of the DiscoveryClient can be run from the project's source folder by entering *ant runClient*. The discovery unit tests can also be run by entering *ant test*. The unit tests do not actually communicate with the Index Service; rather they simulate it with a Mock object.

Configuring an Index Service

The first step in using the Discovery API is constructing an instance of the DiscoveryClient. There are three constructors that can be used. The first, shown in line 7 of Figure 3-10, takes no arguments, and indicates that the "default" Index Service should be used for discovery queries. A second constructor, shown in line 5 of Figure 3-10, takes a String as an argument, and the String is expected to represent the service URL of the Index Service to query. The final constructor, not shown, takes an *EndPointReferenceType*, which can be used to directly

indicate the Index Service Resource to query. The standard caGrid Index Service installation is stateless, and so a resource unqualified EPR can be used, but most clients can just use the shortcut String constructor.

```
1    public static void main(String[] args) {
2        DiscoveryClient client = null;
3        try {
4            if (args.length == 1) {
5                client = new DiscoveryClient(args[0]);
6            } else {
7                client = new DiscoveryClient();
8            }
9        } catch (MalformedURIException e) {
10           System.err.println("Invalid URI specified for Index Service:" + e);
11           e.printStackTrace();
12           System.exit(-1);
13       }
```

Figure 3-10 DiscoveryClient constructor example

The Index Service to use can also be reconfigured at runtime, by invoking the *setIndexEPR* method, shown in line 11 of Figure 3-11. Just as specifying the Index Service in the constructor generates an exception if the Address is not valid, so will the setter method.

```
1        EndpointReferenceType indexEPR = new EndpointReferenceType();
2        try {
3            Address address = new Address(
4                "http://cagrid01.bmi.ohio-state.edu:8080/wsrf/services/DefaultIndexService"));
5            indexEPR.setAddress(address);
6        } catch (MalformedURIException e) {
7            System.err.println("Invalid URI specified for Index Service:" + e);
8            e.printStackTrace();
9            System.exit(-1);
10       }
11       client.setIndexEPR(indexEPR);
```

Figure 3-11 Configuring Index Service code

Discovering Services

Once a *DiscoveryClient* is configured, it can be continually used to discover services of interest. While the client is technically thread safe as long as the Index Service is not reconfigured during use, it is recommended a new *DiscoveryClient* instance is used in each thread context where discovery operations are performed, as it is an extremely light weight object.

The simplest discovery scenario, shown in Figure 3-12, is to query the Index Service for all registered services. The boolean value specified in line 3, indicates whether services should be ignored if they do not expose the caGrid standard metadata. In most application scenarios, a value of "true" is used, but specifying "false" is useful for identifying all services that are attempting to register. It is common for a service running behind a firewall to maintain registration status with the Index Service, but not have caGrid metadata aggregated, as the Index Service is not able to communicate with the inaccessible service

```
EndpointReferenceType[] allServices = null;
try {
    allServices = client.getAllServices(true);
} catch (ResourcePropertyRetrievalException e1) {
    e1.printStackTrace();
    System.exit(-1);
}
```

Figure 3-12 Discovering all services

There are numerous discovery operations which take some form of text input, and all are case sensitive. The simplest discovery operation that takes some form of input is the basic string search operation, *discoverServicesBySearchString*, which is shown in Figure 3-13. This is a full text search that examines all registered metadata values for the specified input. It is not likely this operation will be useful for programmatic discovery (as it is a completely unstructured query), but it is useful for applications that take direct input from the user (such as a web form), and makes a good starting point for applications that provide capability to “drill down” and examine the full metadata of the satisfying services.

```
EndpointReferenceType[] services = null;
try {
    services = client.discoverServicesBySearchString("chemical reaction");
} catch (ResourcePropertyRetrievalException e) {
    e.printStackTrace();
    System.exit(-1);
}
```

Figure 3-13 Discovering by Search String

Beyond the full text search operation, there are many discovery operations that take a search string as input, but perform a more structured search and are more useful for programmatic discovery. For example, services providing a named operation can be discovered using the method *discoverServicesByOperationName*, or Data Services exposing a given model can be discovered, as shown below in Figure 3-14, using the *discoverDataServicesByDomainModel* method. This operation, and all methods named like *discoverDataServices** only return services that implement the standard Data Service operations.

```
EndpointReferenceType[] services = null;
try {
    services = client.discoverDataServicesByDomainModel("caCORE");
} catch (ResourcePropertyRetrievalException e) {
    e.printStackTrace();
    System.exit(-1);
}
```

Figure 3-14 Discovering Data Services by Model Name

Another potentially useful method for discovering services or displaying information about available services on the grid is the *discoverServicesByResearchCenter* method, shown below in Figure 3-15.

```
1        EndpointReferenceType[] services = null;
2        try {
3            services = client.discoverServicesByResearchCenter("Ohio State University");
4        } catch (ResourcePropertyRetrievalException e) {
5            e.printStackTrace();
6            System.exit(-1);
7        }
```

Figure 3-15 Discovering by Research Center

There are several discovery methods that support semantic discovery by allowing search on concept code. The simplest of these methods, *discoverServicesByConceptCode,* shown below in Figure 3-16, searches for services based on concepts applied to the services itself. There is a concept representing "Grid Service" in the ontology and derived concepts such as "Analytical Grid Service" and "Data Grid Service." By determining these concept codes, or any other specialized concepts, this operation provides a simple way to discover services of a certain "type." Similarly, there is a method to discover services by the semantics of the operations they provide using the *discoverServicesByOperationConceptCode* method. At the time of this writing, services operations are not yet semantically annotated, but are expected to be soon. Finally, two methods: *discoverDataServicesByModelConceptCode* and *discoverServicesByDataConceptCode* provide the capability to discover services based on the information about the data types they operate over. Both examine the semantic information of the UML Classes used by the services. The first, *discoverDataServicesByModelConceptCode*, locates Data Services that are exposing access to data based on the concept. The second, *discoverServicesByDataConceptCode*, locates services that directly produce or consume data based on the concept. In both cases, the concept is considered a match if the Class is based on the concept or one of its attributes, attribute value domains, or enumerated value meanings. These methods are all based on direct concept matching; not only ontological operations. However, these methods coupled with the EVS grid service, provide a powerful ability to traverse the caBIG ontology for information of interest, and discover services providing this information, or the ability to manipulate it.

```
1        EndpointReferenceType[] services = null;
2        try {
3            services = client.discoverServicesByConceptCode("C43418");
4        } catch (ResourcePropertyRetrievalException e) {
5            e.printStackTrace();
6            System.exit(-1);
7        }
```

Figure 3-16 Discovering Services by Concept Code

Beyond the simple String based discovery methods, some discovery methods take complex objects as input, such as a *PointOfContact* or *UMLClass*. In these cases, the objects act as a prototype (or "query by example" as in the caCORE APIs), and can be as partially populated as desired. For example, the method show below in Figure 3-17, *discoverServicesByPointOfContact*, searches for services which are associated with a person with the information described by the supplied *PointOfContact* instance; in this case services associated with "Scott Oster" are located. There are many other fields in *PointOfContact* that are not populated in this example, and are ignored.

```
EndpointReferenceType[] services = null;
try {
    PointOfContact poc = new PointOfContact();
    poc.setFirstName("Scott");
    poc.setLastName("Oster");

    services = client.discoverServicesByPointOfContact(poc);
} catch (ResourcePropertyRetrievalException e) {
    e.printStackTrace();
    System.exit(-1);
}
```

Figure 3-17 Discover Services by Point of Contact

There are many discovery methods that take a UMLClass prototype to discover services based on data types; an example is shown below in Figure 3-18. This method, *discoverServiceByOperationInput*, locates services that provide an operation that takes, as input, an instance of the specified data type. In the example below, services provide operations taking caBIO's Gene instances as input. Again, this object can be as partially populated as desired (such as only specifying the package name, or being more explicit in specifying the exact project name and version).

```
EndpointReferenceType[] services = null;
try {
    UMLClass umlClass = new UMLClass();
    umlClass.setClassName("Gene");
    umlClass.setPackageName("gov.nih.nci.cabio.domain");

    services = client.discoverServicesByOperationInput(umlClass);
} catch (ResourcePropertyRetrievalException e) {
    e.printStackTrace();
    System.exit(-1);
}
```

Figure 3-18 Discover Services by Input

caDSR Grid Service Usage Overview

The following link provides a reference to the technical architecture and design document(s) for caGrid Metadata:

http://gforge.nci.nih.gov/plugins/scmcvs/cvsweb.php/cagrid-1-0/Documentation/docs/cadsr/caGrid-cadsr-design.doc?cvsroot=cagrid-1-0

The caGrid caDSR Grid Service provides access to information in the caDSR that is relevant to caGrid, and has capabilities to generate caGrid standard metadata instances. Specifically, the service provides operations to access UML-like information stored in the caDSR. It also has operations to generate Data Service metadata for a described subset of a given project registered in caDSR. Finally, it has an operation that augments a description of an Analytical Service, via a partially populated service metadata instance, with the necessary UML-like and semantic information, extracted from caDSR, to describe the service and its operations.

The caDSR Grid Service is a simple, stateless service, created with Introduce. The service exposes three main categories of operations. The first are operations that expose access to the

UML-like view of caDSR registered items like *findProjects*, *findPackagesInProject*, *findClassesInPackage*, *findAttributesInClass*, etc. These provide basic discovery and access to the UML information in the caDSR. While these operations are stateless, they take sufficient context during each invocation to enable traversal of all registered projects. Aside from the operations to locate Projects, each operation takes a description of the caDSR Project of interest. Each operation in turn throws an *InvalidProjectException* if the Project specified is not valid.

The second set of operations enables clients to generate caGrid standard Data Service metadata. There are four operations that take variations of information specifying what data is to be exposed by the Data Service for which the metadata is being created. Each operation throws an *InvalidProjectException* if the Project specified is not valid or if it ambiguously identifies more than one Project (for example is a version is not specified, yet there are multiple versions of a given Project registered in caDSR). The first operation, *generateDomainModelForProject,* takes only the project description, and generates a model that describes the entire domain model being exposed for the project. The second, *generateDomainModelForPackage*, additionally takes an array of Strings that represent UML package names in the Project to expose. The method generates a model that describes exposing all UML Classes in UML Packages with a name specified in the array. Any associations to UML Classes outside of the specified packages are not exposed. The third method, *generateDomainModelForClasses*, also takes an array of Strings that represent the fully qualified UML Class names to be exposed in the model. Any association between classes not specified is omitted. The final method, *generateDomainModelForClassesWithExcludes*, also takes an additional array of Strings that represent the fully qualified UML Class names to be exposed in the model, but also takes an array of *UMLAssociationExcludes* to be used to exclude specific associations from the model (in addition to the already excluded associations that reference classes not specified in the array of class names). The *UMLAssociationExclude* Class allows the client to specify a *sourceRoleName, sourceClassName, targetRoleName,* and *targetClassName*. Any UML Association that would otherwise be included in the computed subset of the DomainModel is omitted if it meets the criteria described by any of the *UMLAssociationExcludes*. The value of any attribute of the *UMLAssociationExclude* can be the wildcard "*", which indicates it should match anything. As such, specifying an exclude with "*" as the value for all attributes effectively omits all associations from the *DomainModel*. By using no wildcards, a single association can be omitted, and by using a combination of some values and some wildcards, groups of associations can be omitted. For example, specifying an exclude instance with a *sourceClassName* value of "*gov.nih.nci.cabio.domain.Gene*" and wildcards for all other attributes would effectively omit any associations from the *DomainModel* where *gov.nih.nci.cabio.domain.Gene* was the source of the association. Using these methods, in combination with the operations for finding all Projects, Packages, Classes, and Associations, Data Service metadata exposing any subset of Classes and Associations can be created.

The final type of operation is the operation, *annotateServiceMetadata,* which provides clients the ability to augment a *ServiceMetadata* (standard caGrid service metadata) skeleton instance with the information extracted from caDSR. The caGrid common service metadata specifies information about a grid service and its operations. For more information on the model, consult the caGrid metadata design document. The *annotateServiceMetadata* operation takes this model and populates the UML and semantically oriented components by querying the caDSR

appropriately. Specifically, it populates the semantically annotated UML Class information (similar to the type used in Data Service Domain Model metadata) for each input and output type of every operation the service provides. It does this by examining the XML Qualified Name (QName) of each type used in the signature of the operation and locating its UML equivalent in caDSR. In caGrid every grid service operation is required to use data types which are XML representations of UML Classes registered in the caDSR. There is a one to one mapping of UML Class to XML QNames (XML elements). The caGrid Metadata Design Document and caDSR Grid Service Design Document can be consulted for more information on how this binding (XML QName to caDSR type) occurs, and what restrictions it places on the models.

The primary data types used by the caDSR grid service are those which are defined in caCORE 3.1 in the *gov.nih.nci.cadsr.domain* model, which represents the caDSR information and the *gov.nih.nci.cadsr.umlproject.domain,* which represents a UML-like view of information in the caDSR. The umlproject model, shown below in Figure 3-19, is the main model, but it associates with and extends a few classes from the caDSR base model, so it is used as well. As is evident from the figure below, the model provides a UML-like view of the caDSR registered projects. One class of note is the *SemanticMetadata* class which is associated to many UML-like classes, and provides a link to the semantic content of those items. Specifically, it exposes information about the EVS-maintained concepts.

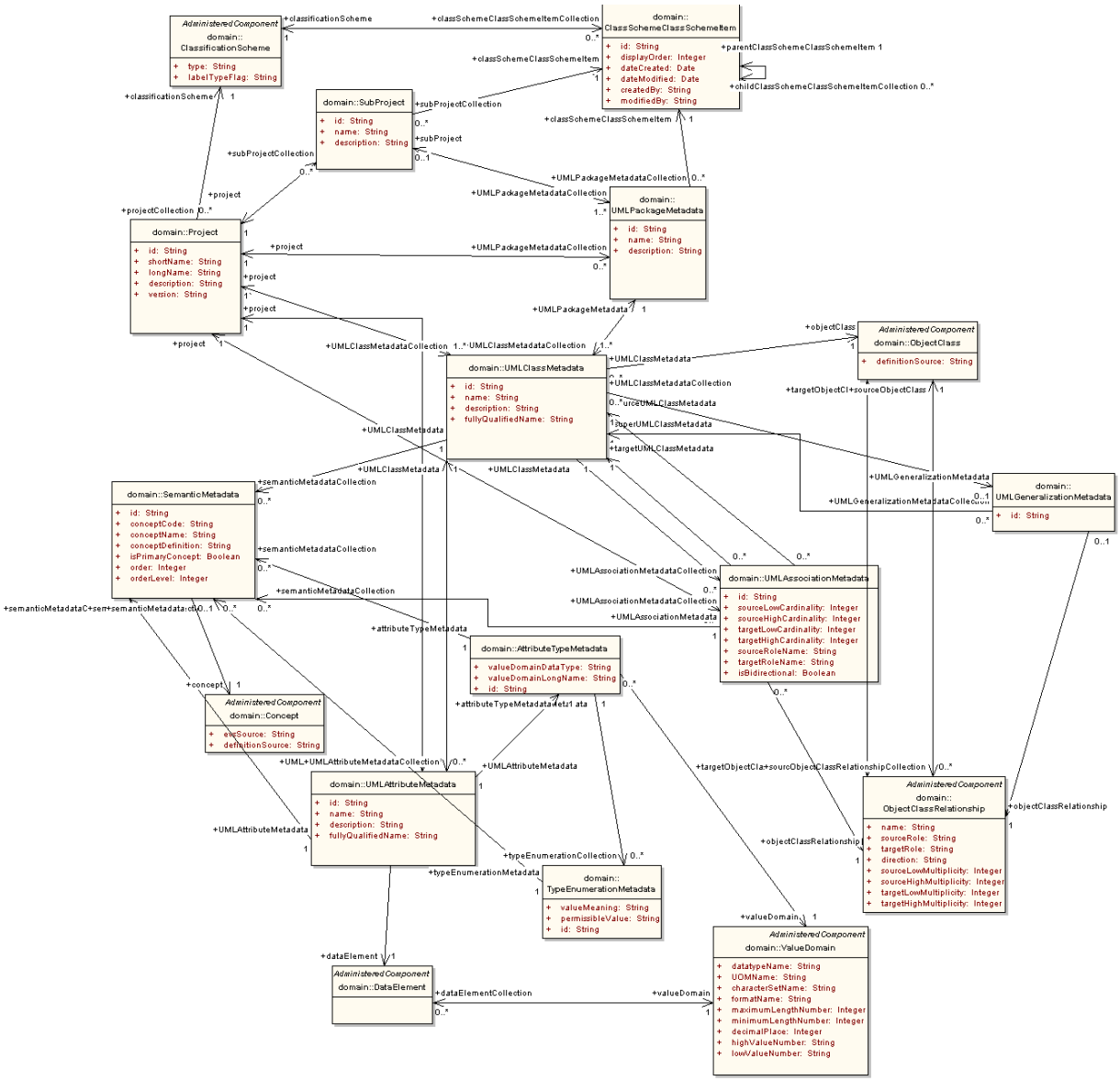


Figure 3-19 UML project model

In addition to the existing caCORE-defined types, the caDSR grid service defines two new data types for exclusive use by its exposed operations. The first is the *UMLAssociationExclude*, described above as the type used to specify UML Associations which should be excluded from a generated *DomainModel*. The second is an alternative representation of a UML Association, namely the *UMLAssocation* class. This has the same semantics as the *UMLAssociationMetadata* class in the umlproject model, but uses an alternate syntactic representation which is more suitable to transport over the grid.

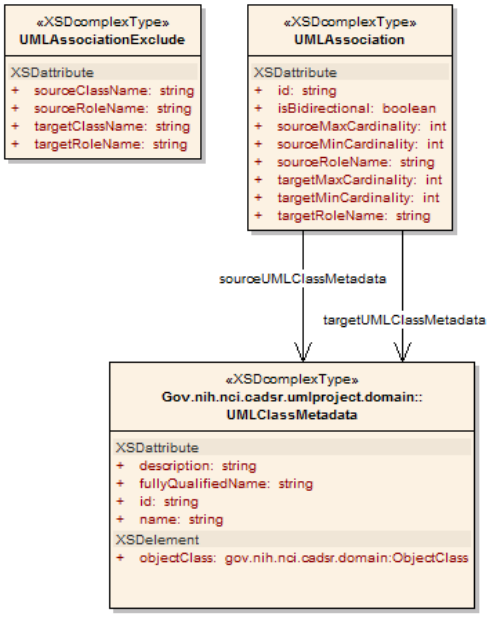


Figure 3-20 caDSR Grid Service Types

Finally the service also makes use of the *ServiceMetadata* and *DomainModel* caGrid metadata models, as it provides operations to manipulate them. For more information on these models, consult the caGrid Metadata Design Document.

Security Considerations

The caDSR grid service requires no grid credentials for any operations. Its typical deployment is in a service container using an open communication channel. However, even if it is deployed to a container making use of transport level security (https), it will not require credentials from the user, and can be communicated with anonymously.

API Details

The caDSRServiceClient is the main client interface to the caDSR grid service (Figure 3-21).

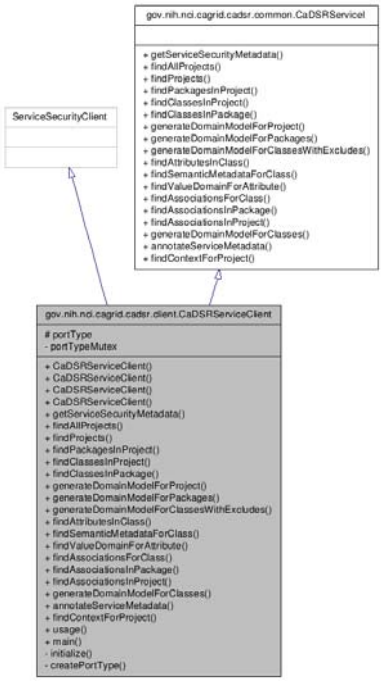


Figure 3-21 CaDSRServiceClient Inheritance Graph

gov.nih.nci.cagrid.cadsr.client.CaDSRServiceClient

Constructor Documentation

CaDSRServiceClient (String url) - throws MalformedURIException, RemoteException

CaDSRServiceClient (String url, GlobusCredential proxy) - throws MalformedURIException, RemoteException

CaDSRServiceClient (EndpointReferenceType epr) - throws MalformedURIException, RemoteException

CaDSRServiceClient (EndpointReferenceType epr, GlobusCredential proxy) - throws MalformedURIException, RemoteException

Member Function Documentation

Access to caGrid Service Security Metadata:

- gov.nih.nci.cagrid.metadata.security.ServiceSecurityMetadata **getServiceSecurityMetadata** ()
    - throws RemoteException
    - **Description**: Returns the caGrid service security metadata

Navigation and discovery of UML-like information:

- gov.nih.nci.cadsr.umlproject.domain.Project[] **findAllProjects** ()

    - throws RemoteException
    - **Description**: Returns all Projects registered in the caDSR
- gov.nih.nci.cadsr.umlproject.domain.Project[] **findProjects** (java.lang.String context)
    - throws RemoteException
    - **Description**: Returns all Projects registered in the caDSR under the given context
- gov.nih.nci.cadsr.umlproject.domain.UMLPackageMetadata[] **findPackagesInProject** (gov.nih.nci.cadsr.umlproject.domain.Project project)
    - throws RemoteException, gov.nih.nci.cagrid.cadsr.stubs.types.InvalidProjectException
    - **Description**: Returns all Packages in the given Project
- gov.nih.nci.cadsr.umlproject.domain.UMLClassMetadata[] **findClassesInProject** (gov.nih.nci.cadsr.umlproject.domain.Project project)
    - throws RemoteException, gov.nih.nci.cagrid.cadsr.stubs.types.InvalidProjectException
    - **Description**: Returns all Classes in the given Project
- gov.nih.nci.cadsr.umlproject.domain.UMLClassMetadata[] **findClassesInPackage** (gov.nih.nci.cadsr.umlproject.domain.Project project, java.lang.String packageName)
    - throws RemoteException, gov.nih.nci.cagrid.cadsr.stubs.types.InvalidProjectException
    - **Description**: Returns all Classes in the given Package
- gov.nih.nci.cadsr.umlproject.domain.UMLAttributeMetadata[] **findAttributesInClass** (gov.nih.nci.cadsr.umlproject.domain.Project project, gov.nih.nci.cadsr.umlproject.domain.UMLClassMetadata clazz)
    - throws RemoteException, gov.nih.nci.cagrid.cadsr.stubs.types.InvalidProjectException
    - **Description**: Returns all Attributes of the given Class
- gov.nih.nci.cadsr.umlproject.domain.SemanticMetadata[] **findSemanticMetadataForClass** (gov.nih.nci.cadsr.umlproject.domain.Project project, gov.nih.nci.cadsr.umlproject.domain.UMLClassMetadata clazz)
    - throws RemoteException, gov.nih.nci.cagrid.cadsr.stubs.types.InvalidProjectException
    - **Description**: Returns the semantic information about the given Class
- gov.nih.nci.cadsr.domain.ValueDomain **findValueDomainForAttribute** (gov.nih.nci.cadsr.umlproject.domain.Project project, gov.nih.nci.cadsr.umlproject.domain.UMLAttributeMetadata attribute)
    - throws RemoteException, gov.nih.nci.cagrid.cadsr.stubs.types.InvalidProjectException
    - **Description**: Returns the Value Domain information for the given Attribute
- gov.nih.nci.cagrid.cadsrservice.UMLAssociation[] **findAssociationsForClass** (gov.nih.nci.cadsr.umlproject.domain.Project project, gov.nih.nci.cadsr.umlproject.domain.UMLClassMetadata clazz)
    - throws RemoteException, gov.nih.nci.cagrid.cadsr.stubs.types.InvalidProjectException
    - **Description**: Returns all Associations of the given Class
- gov.nih.nci.cagrid.cadsrservice.UMLAssociation[] **findAssociationsInPackage** (gov.nih.nci.cadsr.umlproject.domain.Project project, java.lang.String packageName)

    - throws RemoteException, gov.nih.nci.cagrid.cadsr.stubs.types.InvalidProjectException
    - **Description**: Returns all Associations in the given Package
- gov.nih.nci.cagrid.cadsrservice.UMLAssociation[] **findAssociationsInProject** (gov.nih.nci.cadsr.umlproject.domain.Project project)
    - throws RemoteException, gov.nih.nci.cagrid.cadsr.stubs.types.InvalidProjectException
    - **Description**: Returns all Associations in the given Project
- gov.nih.nci.cadsr.domain.Context **findContextForProject** (gov.nih.nci.cadsr.umlproject.domain.Project project)
    - throws RemoteException, gov.nih.nci.cagrid.cadsr.stubs.types.InvalidProjectException
    - **Description**: Returns the Context of the given Project

caGrid standard metadata generation and manipulation:

- gov.nih.nci.cagrid.metadata.dataservice.DomainModel **generateDomainModelForProject** (gov.nih.nci.cadsr.umlproject.domain.Project project)
    - throws RemoteException, gov.nih.nci.cagrid.cadsr.stubs.types.InvalidProjectException
    - **Description**: Generates standard Data Service metadata for the given Project
- gov.nih.nci.cagrid.metadata.dataservice.DomainModel **generateDomainModelForPackages** (gov.nih.nci.cadsr.umlproject.domain.Project project, java.lang.String[] packageNames)
    - throws RemoteException, gov.nih.nci.cagrid.cadsr.stubs.types.InvalidProjectException
    - **Description**: Generates standard Data Service metadata for the given Packages
- gov.nih.nci.cagrid.metadata.dataservice.DomainModel **generateDomainModelForClasses** (gov.nih.nci.cadsr.umlproject.domain.Project project, java.lang.String[] fullClassNames)
    - throws RemoteException, gov.nih.nci.cagrid.cadsr.stubs.types.InvalidProjectException
    - **Description**: Generates standard Data Service metadata for the given Classes
- gov.nih.nci.cagrid.metadata.dataservice.DomainModel **generateDomainModelForClassesWithExcludes** (gov.nih.nci.cadsr.umlproject.domain.Project project, java.lang.String[] fullClassNames, gov.nih.nci.cagrid.cadsrservice.UMLAssociationExclude[] associationExcludes)
    - throws RemoteException, gov.nih.nci.cagrid.cadsr.stubs.types.InvalidProjectException
    - **Description**: Generates standard Data Service metadata for the given Classes, excluding the specified (if any) Associations
- gov.nih.nci.cagrid.metadata.ServiceMetadata **annotateServiceMetadata** (gov.nih.nci.cagrid.metadata.ServiceMetadata serviceMetadata)
    - throws RemoteException
    - **Description**: Annotates the given standard service metadata instance with semantically annotated UML class information for all operations inputs and outputs

API Usage Examples

This section describes typical usage of the caDSR Grid Service Client API. The exception handling shown in the code examples is not recommended practice, and is simplistic for demonstration purposes.

Examining a Project's Information Model

The example shown below in Figure 3-22, shows simple logic that can be used to list out the Projects, Packages, Classes, and Attributes registered in the caDSR. It is a simplified extract of the main method of the CaDSRServiceClient, and can run in full from the project by typing "`ant runClient`." The first step in communicating with the service, as with all caGrid services, is to construct an instance of the client, passing the URL of the service. This can be seen in line 1 of the example. Next in line 4, the example asks the caDSR for all of the registered Projects, and then loops over them in line 6. For each returned Project, the caDSR is then asked for the Packages registered in that Project, shown in line 9. This demonstrates the basic process which is used to navigate the UML-like information in the caDSR, wherein context of the previous operation (in this case a Project), is passed to a successive operation which provides more detailed information. This can be seen again in line 15, line 22, and line 31, where respectively the Classes in a Package, Attributes in a Class, and Associations in a Class are accessed. The result of running this example is a printout out of a "tree like" view of all caDSR registered Projects. While generally one does not want to navigate all information in the caDSR, this basic model can be followed to find whatever information is desired (first locating the Project of interest, then passing it to operations which provide more detail).

```
1   CaDSRServiceClient cadsr = new CaDSRServiceClient(
2       "http://cagrid-service.nci.nih.gov:8080/wsrf/services/cagrid/CaDSRService");
3
4   Project[] projs = client.findAllProjects();
5   if (projs != null) {
6       for (int i = 0; i < projs.length; i++) {
7           Project project = projs[i];
8           System.out.println("\n" + project.getShortName());
9           UMLPackageMetadata[] packs = client.findPackagesInProject(project);
10          if (packs != null) {
11              for (int j = 0; j < packs.length; j++) {
12                  UMLPackageMetadata pack = packs[j];
13                  System.out.println("\t-" + pack.getName());
14
15                  UMLClassMetadata[] classes = client.findClassesInPackage(
16                      project, pack.getName());
17                  if (classes != null) {
18                      for (int k = 0; k < classes.length; k++) {
19                          UMLClassMetadata clazz = classes[k];
20                          System.out.println("\t\t-" + clazz.getName());
21                          UMLAttributeMetadata[] atts =
22                              client.findAttributesInClass(project, clazz);
23                          if (atts != null) {
24                              for (int l = 0; l < atts.length; l++) {
25                                  UMLAttributeMetadata att = atts[l];
26                                  System.out.println("\t\t\t-" + att.getName());
27                              }
28                          }
29
30                          UMLAssociation[] assocs =
31                              client.findAssociationsForClass(project, clazz);
32                          if (assocs != null) {
33                              for (int index = 0; index < assocs.length; index++) {
34                                  UMLAssociation assoc = assocs[index];
35                                  System.out.println("\t\t\t("
36                                      + assoc.getSourceRoleName()
37                                      + ")---> ("+ assoc.getTargetRoleName()+ ")"
38                                      + assoc.getTargetUMLClassMetadata()
39                                          .getUMLClassMetadata().getFullyQualifiedName());
40                              }
41                          }
42                      }
43                  }
44              }
45          }
46      }
47  }
48
```

Figure 3-22 Navigating projects registered in the caDSR

Generating Data Service Metadata

As described above, there are four operations which provide the ability to generate instances of caGrid standard Data Service metadata. Three examples of using this capability are shown below, wherein the resulting DomainModel is written to a local file, and some basic information about the model is printed to the screen. It should be noted that most users will not need to make use of these APIs, as the Introduce toolkit automatically does this for them when creating a Data Service, using the appropriate information based on the configuration of the service.

The first example, shown in Figure 3-23, demonstrates how to create metadata which describes an entire Project being exposed; in this case the 3.1 version of caCORE. Lines 4-6 demonstrate

the construction of a “prototype” Project instance, which describes the Project being exposed. Then, in line 8, this is passed to the *generateDomainModelForProject* operation of the caDSR grid service, and the resulting metadata instance is returned. This is the simplest of the Data Service metadata generation operations, and does not allow and configuration or restriction of the model. However, the resulting object could then be manipulated by the client appropriately.

The following lines (10-19) are common to all the following examples and show how the *DomainModel* can be written to a local file (lines 10-13), and how the model can be inspected as a Java Bean. Line 17 shows how the descriptions of the Associations of the model are accessed, and line 19 shows how the descriptions of the exposed Classes can be accessed.

```
1    CaDSRServiceClient cadsr = new CaDSRServiceClient(
2        "http://cagrid-service.nci.nih.gov:8080/wsrf/services/cagrid/CaDSRService");
3
4    Project project = new Project();
5    project.setVersion("3.1");
6    project.setShortName("caCORE");
7
8    DomainModel domainModel = cadsr.generateDomainModelForProject(project);
9
10   FileWriter fw = new FileWriter(project.getShortName()
11       + "_" + project.getVersion() + "_DomainModel.xml");
12   MetadataUtils.serializeDomainModel(domainModel, fw);
13   fw.close();
14
15   System.out.println("Domain Model generation complete.");
16   System.out.println("\t exposed Association count:"
17       + domainModel.getExposedUMLAssociationCollection().getUMLAssociation().length);
18   System.out.println("\t exposed Class count:"
19       + domainModel.getExposedUMLClassCollection().getUMLClass().length);
```

Figure 3-23 Generating a Domain Model for a Whole Project

The next example, shown in Figure 3-24, demonstrates how a metadata instance that describes exposing a subset of a Project can be created. In this case, on line 9, an array of package names, containing only caBIO, is constructed and passes to the *generateDomanModelForPackages* operation. This operation restricts the returned model to only contain Classes that are in the specified package(s), and Associations between those Classes.

```
1   CaDSRServiceClient cadsr = new CaDSRServiceClient(
2       "http://cagrid-service.nci.nih.gov:8080/wsrf/services/cagrid/CaDSRService");
3
4   Project project = new Project();
5   project.setVersion("3.1");
6   project.setShortName("caCORE");
7
8   DomainModel domainModel =cadsr.generateDomainModelForPackages(project,
9       new String[]{"gov.nih.nci.cabio.domain"});
10
11  FileWriter fw = new FileWriter(project.getShortName()
12      + "_" + project.getVersion() + "_DomainModel.xml");
13  MetadataUtils.serializeDomainModel(domainModel, fw);
14  fw.close();
15
16  System.out.println("Domain Model generation complete.");
17  System.out.println("\t exposed Association count:"
18      + domainModel.getExposedUMLAssociationCollection().getUMLAssociation().length);
19  System.out.println("\t exposed Class count:"
20      + domainModel.getExposedUMLClassCollection().getUMLClass().length);
```

Figure 3-24 Generating a domain model for a package

The final example, shown in Figure 3-25, demonstrates the most powerful domain model generation operation, which allows the explicit specification of the Classes which are being exposed and the restriction of Associations between those Classes. Shown in lines 8 and 9, an array of Class names is constructed, in this case the generated model will specify the Data Service is only exposing 4 classes: Gene, Chromosome, Taxon, and Tissue. If we generated a model using these Classes, all Associations between them would be included. However, in the example some of these Associations are specified to be excluded from the model, signifying clients may not query over these associations. Lines 10 and 11, first show the construction of an *AssociationExclude* that explicitly specifies to exclude the Association from Gene to Chromosome where Gene is the source, with the role "geneCollection" and Chromosome is the target with "chromosome" as the role. This only matches a single association in the model. Next, in lines 12-13, a broader exclude is constructed which specifies that an Association with Tissue as the target should be excluded. Both these restrictions are then added to an array, in lines 14 and 15, and passed to the service with the Project and Class names on line 16.

```
CaDSRServiceClient cadsr = new CaDSRServiceClient(
    "http://cagrid-service.nci.nih.gov:8080/wsrf/services/cagrid/CaDSRService");

Project project = new Project();
project.setVersion("3.1");
project.setShortName("caCORE");

String classNames[] = new String[]{Gene.class.getName(), Chromosome.class.getName(),
    Taxon.class.getName(), Tissue.class.getName()};
UMLAssociationExclude exclude1 = new UMLAssociationExclude(Gene.class.getName(),
    "geneCollection", Chromosome.class.getName(), "chromosome");
UMLAssociationExclude exclude2 = new UMLAssociationExclude("*", "*",
    Tissue.class.getName(), "*");
UMLAssociationExclude associationExcludes[] = new UMLAssociationExclude[]{
    exclude1, exclude2};
DomainModel domainModel = cadsr.generateDomainModelForClassesWithExcludes(project,
    classNames, associationExcludes);

FileWriter fw = new FileWriter(project.getShortName()
    + "_" + project.getVersion() + "_DomainModel.xml");
MetadataUtils.serializeDomainModel(domainModel, fw);
fw.close();

System.out.println("Domain Model generation complete.");
System.out.println("\t exposed Association count:"
    + domainModel.getExposedUMLAssociationCollection().getUMLAssociation().length);
System.out.println("\t exposed Class count:"
    + domainModel.getExposedUMLClassCollection().getUMLClass().length);
```

Figure 3-25 Generating a restricted domain model

EVS API Usage Overview

The following link provides a reference to the technical architecture and design document(s) for the EVS API:

http://gforge.nci.nih.gov/plugins/scmcvs/cvsweb.php/cagrid-1-0/Documentation/docs/evs/EVS%20Grid%20Service%20Design%20and%20Implementation.doc?cvsroot=cagrid-1-0

getMetaSources

This API provides a list of vocabularies presented by the NCI Metathesaurus. The NCI Metathesaurus maps terms from one standard vocabulary to another, facilitating collaboration, data sharing and data pooling for clinical trials and scientific data services. The Metathesaurus is based on the National Library of Medicine's (NLM) Unified Medical Language System (UMLS) and is composed of over 70 biomedical vocabularies.

Input:

None

Output:

gov.nih.nci.evs.domain.Source[]

Exception:

RemoteException

Examples of Use

Following is example of java client code invoking the API:

```
EVSGridServiceClient client = new EVSGridServiceClient(endpoint);
gov.nih.nci.evs.domain.Source[] sources = client.getMetaSources();
```

where *endpoint* provides **Endpoint Reference** (EPR) to the EVS core grid service deployed on caGrid 1.0

getVocabularyNames

This API provides a list of vocabularies whose concepts are programmatically accessible to the users via the Description Logic representation. All the vocabularies that are accessible via the caCORE 3.1 EVS API are supported by this API.

Input:

None

Output:

gov.nih.nci.cagrid.evs.service.DescLogicConceptVocabularyName []

Exception:

RemoteException

Examples of Use

Following is example of java client code invoking the API:

```
EVSGridServiceClient client = new EVSGridServiceClient(endpoint);
gov.nih.nci.cagrid.evs.service.DescLogicConceptVocabularyName[] dlcNames =
client.getVocabularyNames();
```

where *endpoint* provides **Endpoint Reference** (EPR) to the EVS core grid service deployed on caGrid 1.0

searchDescLogicConcept

This API provides access to concepts and terms that are published by the vocabularies using the Description Logic representation and exposed via the caCORE 3.1 EVS API.

The input to the API (*EVSDescLogicConceptSearchParams*) consists of appropriate vocabulary to query for concepts (*vocabularyName*), the concept name or code to search (*searchTerm*) and a tuning parameter (*limit*) to restrict the amount of objects returned by the API.

The API returns an array of Description Logic concepts with most of the attributes populated and the user can navigate associated data based on the caCORE 3.1 EVS domain model.

Input:

gov.nih.nci.cagrid.evs.service.EVSDescLogicConceptSearchParams

Output:

gov.nih.nci.evs.domain.DescLogicConcept[]

Exception:

RemoteException

InvalidInputExceptionType

Examples of Use

Following is example of java client code invoking the API:

```
EVSDescLogicConceptSearchParams  evsSearchParams = new
EVSDescLogicConceptSearchParams();

          evsSearchParams.setVocabularyName("NCI_Thesaurus");

          evsSearchParams.setSearchTerm(searchTerms[count]);

          evsSearchParams.setLimit(100);

EVSGridServiceClient client = new EVSGridServiceClient(endpoint);

DescLogicConcept[] dlc= client.searchDescLogicConcept(evsSearchParams);
```

where *endpoint* provides **Endpoint Reference** (EPR) to the EVS core grid service deployed on caGrid 1.0.

API-Specific Considerations

The API has following restrictions:

1. Instances of *gov.nih.nci.evs.domain.EdgeProperties* are not populated currently by the API. These objects are used to specify relationship between a concept and its immediate parent when a DefaultMutableTree is generated. And, the EVS 1.0 grid service is not required to support the generation of a Tree.
2. Instances of *gov.nih.nci.evs.domain.Qualifier* are not populated by the caGrid EVS 1.0 service.
3. Instances of *gov.nih.nci.evs.domain.TreeNode* are not populated by the caGrid EVS 1.0 service.

Exception Handling

The API validates inputs and throws *InvalidInputExceptionType* if any of the following invalid values are present in the input to the API:

1. If the input instance *EVSDescLogicConceptSearchParams* is null.
2. If the value of the tuning parameter(*limit*) is less than or equal to zero.

3. If vocabulary name (*vocabularyName*) is not set (null or empty string) or the vocabulary name is not present in the list of available vocabularies supported by caCORE 3.1 EVS API for concepts based on Description Logic representation.

getHistoryRecords

This API provides complete *History* for concepts tracings the evolution of the concept as they are created, merged, modified, split, or retired. This history mechanism is provided completely for the NCI Thesaurus, published by the NCI EVS team while for all other vocabularies that provide concepts based on Description Logic representation, a dummy value for *History* is provided.

The input to the API (*EVSHistoryRecordsSearchParams*) consists of appropriate vocabulary to query for concepts (*vocabularyName*) and a valid concept code to search (*conceptCode*).

The API returns an array of History records for the specified Description Logic Concept with attributes populated and the user can navigate associated data based on the caCORE 3.1 EVS domain model.

Input:

gov.nih.nci.cagrid.evs.service.EVSHistoryRecordsSearchParams

Output:

gov.nih.nci.evs.domain.HistoryRecord[]

Exception:

RemoteException

InvalidInputExceptionType

Examples of Use

Following is example of java client code invoking the API:

```
        EVSHistoryRecordsSearchParams  evsHistoryParams = new
EVSHistoryRecordsSearchParams();

        evsHistoryParams.setVocabularyName("NCI_Thesaurus");

        evsHistoryParams.setConceptCode("C16612");


        EVSGridServiceClient client = new EVSGridServiceClient(endpoint);

        HistoryRecord[] historys = client.getHistoryRecords(evsHistoryParams);
```

where *endpoint* provides **Endpoint Reference** (EPR) to the EVS core grid service deployed on caGrid 1.0

API-Specific Considerations

None

Exception Handling

The API validates inputs and throws *InvalidInputExceptionType* if any of the following invalid values are present in the input to the API:

1. If the input instance *EVSHistoryRecordsSearchParams* is null.
2. If the concept code (*conceptCode*) is not set (null or empty string) or an invalid concept code is passed to the API... All concept codes are expected to start with the letter “C”.
3. If vocabulary name (*vocabularyName*) is not set (null or empty string) or the vocabulary name is not present in the list of available vocabularies supported by caCORE 3.1 EVS API for concepts based on Description Logic representation.

searchMetaThesaurus

This API provides access to concepts that are supported by the NCI Metathesaurus. The input to the API (*EVSMetaThesaurusSearchParams*) consist of a search term or a concept unique identifier (CUI) (*searchTerm*), a valid Metathesaurus source (*source*) and tuning parameters to control the amount of results (*limit*, *shortResponse* and *score*). The API returns an array of concepts from the Metathesaurus with all the attributes populated and the user can navigate associated data based on the caCORE 3.1 EVS domain model.

Input:

gov.nih.nci.cagrid.evs.service.EVSMetaThesaurusSearchParams

Output:

gov.nih.nci.evs.domain.MetaThesaurusConcept[]

Exception:

RemoteException

InvalidInputExceptionType

Examples of Use

Following is example of java client code invoking the API:

```
EVSMetaThesaurusSearchParams evsMetaThesaurusSearchParam = new
EVSMetaThesaurusSearchParams();

  evsMetaThesaurusSearchParam.setLimit(100);

  evsMetaThesaurusSearchParam.setSource("*");
```

```
        evsMetaThesaurusSearchParam.setCui(false);

        evsMetaThesaurusSearchParam.setShortResponse(false);

        evsMetaThesaurusSearchParam.setScore(false);

        EVSGridServiceClient client = new EVSGridServiceClient(endpoint);

        MetaThesaurusConcept[] metaConcept =
client.searchMetaThesaurus(metaParams);
```

where *endpoint* provides **Endpoint Reference** (EPR) to the EVS core grid service deployed on caGrid 1.0

API-Specific Considerations

The API has following constraints:

- The source input to the API has to be a valid metathesaurus source abbreviation (present in the list from the API *getMetaSources*. A value of "*" indicates that the search term will be queried against ALL available sources
- Search term refers to the concept "name" to be searched in the appropriate source or all sources as indicated above
- The appropriate settings of the tuning parameters to indicate whether the search term is a Concept Unique Identifier (CUI) or a regular search term. If the search term is a CUI, then, the search term is validated to adhere to the following restrictions:
    - The CUI begins with the letter "C"
    - The CUI has a maximum length of 8 characters

Exception Handling

The API validates inputs and throws *InvalidInputExceptionType* if any of the following invalid values are present in the input to the API:

1. If the input instance *EVSMetaThesaurusSearchParams* is null.
2. If the search term (*searchTerm*) is not set (null or empty).
3. If the search term is a CUI, then if an invalid CUI is passed to the API. All CUIs are expected to start with the letter "C" and are 8 characters long.
4. If the value of the tuning parameter (*limit*) is less than or equal to zero.
5. If the Metathesaurus source abbreviation (*source*) is not set (null or empty string) or the source is not present in the list of available vocabularies supported by caCORE 3.1 EVS API ("*" is valid source abbreviation indicating ALL sources)

searchSourceByCode

This API provides access to concepts that are supported by the NCI Metathesaurus. The input to the API (*EVSSourceSearchParams*) consists of a concept unique identifier (CUI) (*code*) and a valid Metathesaurus source (*source*). The API returns an array of concepts from the Metathesaurus with all the attributes populated and the user can navigate associated data based on the caCORE 3.1 EVS domain model.

Input:

gov.nih.nci.cagrid.evs.service.EVSSourceSearchParams

Output:

gov.nih.nci.evs.domain.MetaThesaurusConcept[]

Exception:

RemoteException

InvalidInputExceptionType

Examples of Use

Following is example of java client code invoking the API:

```
EVSSourceSearchParams evsSourceParam = new EVSSourceSearchParams();
evsSourceParam.setCode("0000001800");
evsSourceParam.setSourceAbbreviation("AOD2000");
EVSGridServiceClient client = new EVSGridServiceClient(endpoint);
MetaThesaurusConcept[] mC = client.searchSourceByCode(evsSourceParam);
```

where *endpoint* provides **Endpoint Reference** (EPR) to the EVS core grid service deployed on caGrid 1.0

API-Specific Considerations

The API will take following input:

- A valid Meta Thesaurus source abbreviation (present in the list from the API *getMetaSources*. The ALL sources value of "*" is not permitted for this API.
- A valid value for "code". Some of the concepts in the Meta Thesaurus do not have code associated with them and are displayed in the NCI Meta Thesaurus Browser with the value of "NOCODE". The "NOCODE" value is not valid input for the API.

Exception Handling

The API validates inputs and throws *InvalidInputExceptionType* if any of the following invalid values are present in the input to the API:

1. If the input instance *EVSSourceSearchParams* is null.
2. If the concept code (*code*) is not set (null or empty) or has value "NOCODE".
3. If the Metathesaurus source abbreviation (*source*) is not set (null or empty string) or the source is not present in the list of available vocabularies supported by caCORE 3.1 EVS API.

Chapter 4 caGrid Security

This chapter describes using Dorian and Grid Grouper as part of caGrid security.

Topics in this chapter include:

- Dorian Overview on this page
- Grid Grouper Overview on page 69

Dorian Overview

Managing users and provisioning accounts in the grid is complex, the Globus Toolkit implements support for security via its Grid Security Infrastructure (GSI). The GSI utilizes X509 Identity Certificates for identifying a user. An X509 Certificate with its corresponding private key forms a unique credential or so-called “grid credential” within the Grid. Since Grid credentials are long term credentials and are not directly used in authenticating users to the grid, rather a short term credential called a *grid proxy* is used. *Grid Proxies* consist of a private key and corresponding long term certificate signed by the long term grid credential private key. A *grid proxy* is an extension to traditional X509 certificates providing the ability to delegate you credentials to other services, for example in the case of workflow. Although this approach is very effective and secure, it is difficult to manage in a multi-institutional environment. Using the base Globus toolkit, the provisioning of grid credentials is a manual process, which is far too complicated for users. The overall process is further complicated if a user wishes to authenticate from multiple locations, as a copy of their private key and certificate has to be present at every location. Not only is this process complicated, securely distributing private keys is error prone and poses a security risk. There are also many complexities in terms of provisioning user accounts in an environment consisting of tens of thousands of users from hundreds of institutions, each of which most likely has a user account at their home institution. A practical solution to this problem, both from the point of view of the users’ and their institutions, is to allow those users to authenticate with the grid through the same mechanism in which they authenticate with their institution. Dorian is a grid user management service that (1) hides the complexities of creating and managing grid credentials from the users and (2) provides a mechanism for users to authenticate using their institution’s authentication mechanism, assuming a trust agreement is in place between Dorian and the institution.

Dorian provides a complete Grid-enabled solution, based on public key certificates and SAML, for managing and federating user identities in a Grid environment. Grid technologies have adopted the use of X509 identity certificates to support user authentication. The Security Assertion Markup Language (SAML) has been developed as a standard for exchanging authentication and authorization statements between security domains. Note that Grid certificates and SAML assertions serve different purposes. SAML is mainly used between institutions for securely exchanging authentication information coming from trusted identity providers. The primary use of the certificates is to uniquely identify Grid users, facilitate authentication and authorization across multiple resource providers, and to enable secure

delegation of credentials such that a service or a client program can access resources on behalf of the user. A salient feature of Dorian is that it provides a mechanism for the combined use of both SAML and Grid certificates to authenticate users to the Grid environment through their institution's authentication mechanism.

One of the challenges in building an identity management and federation infrastructure is to create an architecture that incorporates multiple differing authentication mechanisms used by various institutions. In addressing this challenge we identify two possible approaches. The first is to build an infrastructure that would allow pluggable authentication modules, wherein a module would be developed for each authentication mechanism. In this architecture, a user's authentication information would be routed to the appropriate module that contains the logic for authenticating the user with its institution. Although this approach solves the problem, it requires at least one module be developed for each authentication mechanism. This would require the Grid infrastructure administrators to become intimately familiar with each institution's authentication mechanisms, and would increase the system's complexity with each new module added.

Another approach would be for the infrastructure to accept an institutionally supplied, standard "token" as a method of authentication. In this approach users would first authenticate with their institution's identity management system. Upon successfully authentication the institution's identity management system issues a token which can then be given to the federated grid identity management system in exchange for grid credentials. The benefit of this approach over the first is that it does not require writing a plug-in every time a new institutional authentication mechanism comes online. It does, however, require every institutional authentication system to agree upon and be able to provide a common token. As SAML has been adopted by many institutions, we have chosen that token format as the basis of the second approach for Dorian.

The Security Assertion Markup Language (SAML) is an XML standard for exchanging authentication and authorization data between security domains. Generally the exchange of authentication and authorization data is made between an *Identity Provider* (IdP) and another party. An institution's authentication system or identity management system is an example of an IdP. Dorian uses SAML authentication assertions as the enabling mechanism for federating users from local institutions to the grid.

Figure 4-1 illustrates an example usage scenario for Dorian. To obtain grid credentials or a proxy certificate, users authenticate with their institution using the institution's conventional mechanism. Upon successfully authenticating the user, the local institution issues a digitally signed SAML assertion, vouching that the user has authenticated. The user then sends this SAML assertion to Dorian in exchange for grid credentials. Dorian will only issue grid credentials to users that supply a SAML assertion from a Trusted Identity Provider. Dorian's grid service interface provides mechanisms for managing trusted identity providers; this will be discussed in greater detail later in this document. For example, in Figure 4-1, where a Georgetown user wishes to invoke a grid service that requires grid credentials, they first supply the application with their username and password to the Georgetown Authentication Service as they would normally do. The application client authenticates the Georgetown user with the Georgetown Authentication Service, receives a signed SAML assertion which it subsequently passes to Dorian in exchange for grid credentials. These credentials can then be used to invoke the grid services. This illustrates how Dorian can leverage an institution's existing authentication

mechanism and bring its users to the grid.

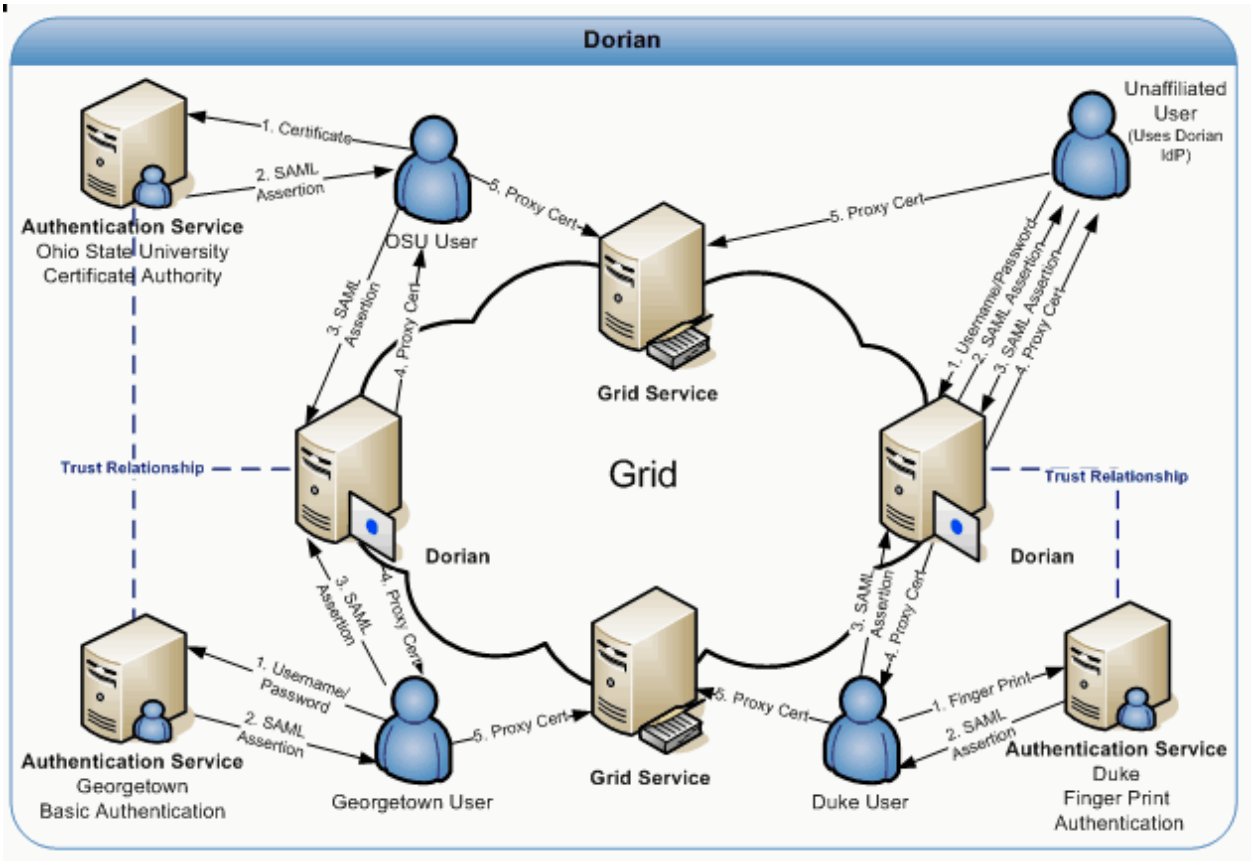


Figure 4-1 Dorian

To facilitate smaller groups or institutions without an existing IdP, Dorian also has its own internal IdP. This allows users to authenticate to Dorian directly, thereby enabling them to access the grid. It provides administrators with facilities for approving and managing users. All of the Dorian IdP's functionality is made available through a grid service interface. Details of the Dorian IdP are provided later in this document. Figure 4-1 illustrates a scenario of a client using the Dorian IdP to authenticate to the Grid. In this scenario, the unaffiliated User wishes to invoke a grid service. Given that this unaffiliated user has registered and been approved for an account, she is able to authenticate with the Dorian IdP by supplying their username and password. Upon successfully authenticating the user, the Dorian IdP issues a SAML Assertion just like institutional IdPs, which can be presented to Dorian in exchange for grid credentials. The credentials can be used to invoke the grid service.

Creating a Grid Proxy Programmatically

Figure 4-2 provides an example of how to create a grid proxy programmatically with Dorian. In order to create a grid proxy using Dorian you must first obtain a signed SAML Assertion from an Identity Provider Trusted by Dorian. caGrid's Authentication Service provides a common interface and client tooling for exposing a local Identity Provider, such that a user may

authenticate using their local credentials and obtain a SAML assertion using a common client or *AuthenticationClient*. Although it is not required to obtain the SAML Assertion from a caGrid Authentication Service, it is the recommended approach and the approach used in Figure 4-2. Besides obtaining a SAML assertion, Dorian also requires the specification of a *proxy lifetime* and a *delegation path length* in order to create a grid proxy. The *proxy lifetime* specifies the amount of time that the proxy is valid for. A *proxy lifetime* is specified in terms of hour, minutes, and seconds. The *delegation path length* specifies how many times a proxy can be delegated to other services. Once you have obtained a SAMLAssertion and specified a *ProxyLifetime* and *delegation path length*, you can use Dorian's *IFSUserClient* to create a proxy with Dorian.

```
try{
    String authURI = "http://some.service.uri";
        String dorianURI = "http://some.dorian.uri";

    //Create an instance of my institution provided credentials
    Credential localCredential = new Credential();
    BasicAuthenticationCredential userPass = new BasicAuthenticationCredential();
    userPass.setUserId("MyUserId");
    userPass.setPassword("MyPassword");
    localCredential.setBasicAuthenticationCredential(userPass);

        //User the caGrid common authentication client to authenticate with the local
        //IdP and obtain a SAML Assertion

        AuthenticationClient auth = new AuthenticationClient(authURI,
localCredential);
        SAMLAssertion saml = auth.authenticate();

    //Specify the lifetime of the desired proxy
    ProxyLifetime lifetime = new ProxyLifetime();
    lifetime.setHours(12);
    lifetime.setMinutes(0);
    lifetime.setSeconds(0);
```

Figure 4-2 Programmatically creating a grid proxy with Dorian

Grid Grouper Overview

Grid Grouper provides a group based authorization solution for caGrid, where grid services and applications enforce authorization policy based on membership to groups defined and managed at the grid level. Grid Grouper is built on top of Grouper an internet2 initiative focused on providing tools for group management. Grouper is a java object model which currently supports: basic group management by distributed authorities; subgroups; composite groups (whose membership is determined by the union, intersection, or relative complement of two other groups); custom group types and custom attributes; trace back of indirect membership; delegation. Applications interact with Grouper by embedding the Grouper's java object model within applications. Grouper does not provide a service interface for accessing groups. Grid Grouper (Figure 4-3) is a grid enabled version of Grouper, providing a web service interface to the Grouper object model. Grid Grouper make groups managed by Grouper available and manageable to applications and other services in the grid. Grid Grouper provides an almost identical object model to the Grouper object model on the grid client side. Applications and services can use the Grid Grouper object model much like they would use the Grouper object model to access and manage groups as well as enforce authorization policy based on membership to groups. Grid Grouper provides a fully functional administrative UI for accessing and administrating groups in Grid Grouper.

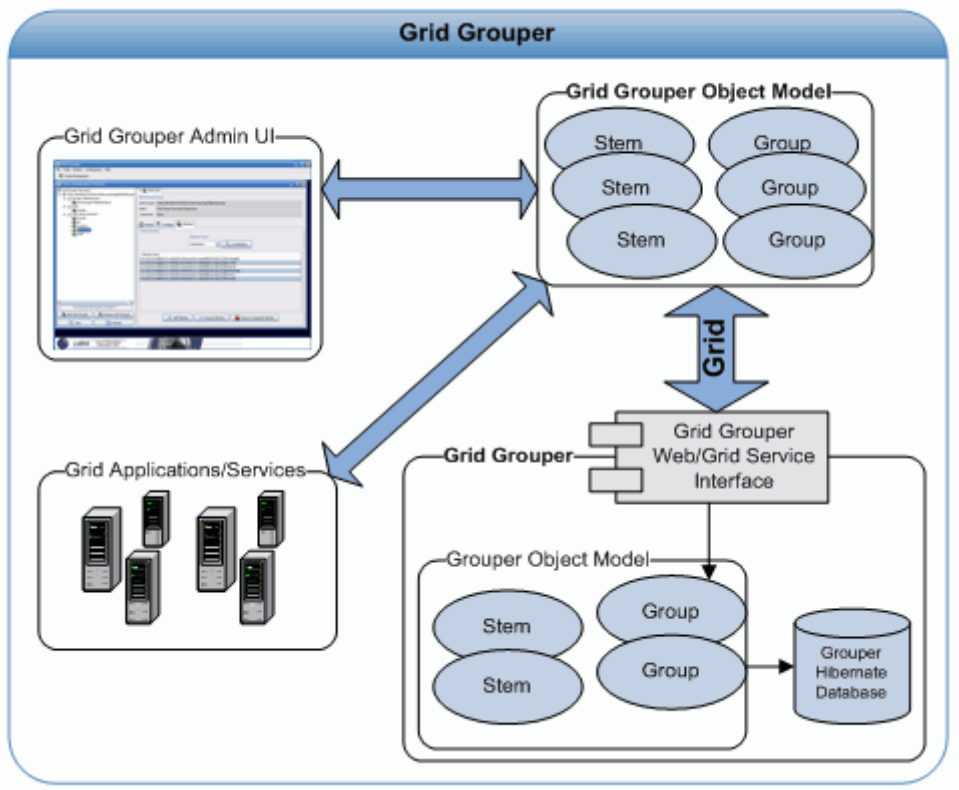


Figure 4-3 Grid Grouper

In Grouper/Grid Grouper groups are organized into namespaces or stems. Each stem can have

a set of child stems and set of child groups with exception to the root stem which cannot have any child groups. The Stem hierarchy in Grid Grouper is publicly visible to anyone accessing the service; however the ability to view a group within a stem publicly depends on the privileges for the group. A Stem can have two types of privileges associated with it, the "Stem Privilege" and the "Create Privilege". Users with the "Stem Privilege" can create, modify, and remove child stems. Users with the "Create Privilege" can create, modify, and remove child groups.

In Grouper/Grid Grouper groups are compromised of a set of metadata describing the group, a set of members in the groups, and a set of privileges assigned to users for protecting access to the group. Grid Grouper provides three mechanisms for adding members to a group: 1) Directly adding a member 2) Adding a subgroup to a group 3) Making a group a composite of other groups. Directly adding a user as a member to a groups is straight forward, these members are referred to as "Immediate Members". Adding a subgroup to a group makes all the members of the subgroup members of the group in which the subgroup was added. Members in a group whose membership is granted by membership in a sub group are referred to as "Effective Members". A group can also be set to be a Composite group. A composite group consists of a set operation (Union, Intersection, Complement) on two other groups. For example a composite group consisting of the Intersection of Group X and Group Y would contain all the members that are both member of Group X and Group Y. Members whose membership is granted through a composite group are referred to as "Composite Members".

To protect access to groups in Grid Grouper, users can be assigned the following privileges on a group: View, Read, Update, Admin, Optin, and Optout. Users with the View privilege can see that the group exists. Users with the Read privilege can read basic information about the group. Users with the Update Privilege can manage memberships to the group as well as administer View, Read, and Update privileges. Users with the Admin privilege can modify/administer anything on the group: metadata, privileges, and memberships. Users with the Optin privilege can add themselves as a member to a group; similarly users with the Opout privilege can remove themselves from a group. By default Grid Grouper grants Read and View privileges to all users on each group.

Initially grid grouper has a root stem with on child stem named "Grouper Administration" (grouperadministration). The Grouper Administrative stem contains one group named "Grid Grouper Administrators" (grouperadministration:gridgrouperadministrators). The "Grid Grouper Administrators" is the super user group for Grid Grouper, all members of this group will have admin privileges on all the stems and groups within Grid Grouper. This group is initially empty, but at least one administrative user must be added during Grid Grouper installation. This can be done using the GridGrouperBootstrapper command line tool.

Grid Grouper Object Model

The Grid Grouper object model provides an API for applications and services to access groups managed by Grid Grouper. The object model can be used to enforce access control policies in applications. For example the object model can be used for determining membership to a group in an application that allows access to a specific area of the application if the user is a member of a specified group. The Grouper object model can also be used to administrate Grid Grouper. As a testament to this the Grid Grouper Admin UI application was built on top of the Grid Grouper object model. The Grid Grouper object model consists of several objects: GridGrouper,

Stem, Group, Member, Membership, NamingPrivilege, and AccessPrivilege. The GridGrouper object corresponds to an instance of a Grid Grouper service; it provides high level operations such as finding stems and groups or determining whether or not a user is a member of a group, etc. The Stem object represents an instance of a stem within Grid Grouper. The Stem object provides operations for managing the stem: viewing metadata, managing child stems, managing child groups, managing stem privileges, etc. The Group object models a group instance within Grid Grouper, providing operations for managing metadata, managing privileges, and managing members. In the remainder of this section we will provide several code examples of performing common tasks with the Grid Grouper object model.

Determining if a Subject is a Member of a Group

```
try {
        String uri ="https://localhost:8443/wsrf/services/cagrid/GridGrouper";
        String user = "/O=OSU/OU=BMI/OU=caGrid/OU=Dorian/OU=cagrid05/CN=jdoe";
    String group = "MyStem:MyGroup";

    //Create a Grid Grouper Instance
    GrouperI grouper = new GridGrouper(uri);

    //Determiner if the user is a member of the group.
    boolean isMember = grouper.isMemberOf(user, group);

    if(isMember){
        System.out.println("The user "+user+" is a member of "+group);
    }else{
       System.out.println("The user "+user+" is NOT a member of "+group);
    }
} catch (Exception e) {
    e.printStackTrace();
    }
```

List All Members of a Group

```
try {
    String uri = "https://localhost:8443/wsrf/services/cagrid/GridGrouper";
    String group = "MyStem:MyGroup";

    // Create a Grid Grouper Instance
    GrouperI grouper = new GridGrouper(uri);

Obtain a handle to the group object.
    GroupI mygroup = grouper.findGroup(group);



    Set s = mygroup.getMembers();
    Iterator itr = s.iterator();
    //Iterate over and print out the members of the group
    while (itr.hasNext()) {
        Member m = (Member) itr.next();
        System.out.println("The user " + m.getSubjectId()
                        + " is a member of " + mygroup.getDisplayExtension());
    }
 } catch (Exception e) {
       e.printStackTrace();
 }
```

Add a Member to a Group

```
try {
    //Group Administrators Grid Credentials
    GlobusCredential adminProxy = ProxyUtil.getDefaultProxy();
    String uri = "https://localhost:8443/wsrf/services/cagrid/GridGrouper";
    String newMember = "/O=OSU/OU=BMI/OU=caGrid/OU=Dorian/OU=cagrid05/CN=jdoe";
    String group = "MyStem:MyGroup";

    // Create a Grid Grouper Instance
     GrouperI grouper = new GridGrouper(uri,adminProxy);

    // Obtain a handle to the group object
    GroupI mygroup = grouper.findGroup(group);

    //Add the member to the group
    mygroup.addMember(SubjectUtils.getSubject(newMember));

    System.out.println("Successfully added the user " + newMember + " as a member of the
group " + group);
} catch (Exception e) {
    e.printStackTrace();
}
```

Removing a Member from a Group

```
try {
    //Group Administrators Grid Credentials
    GlobusCredential adminProxy = ProxyUtil.getDefaultProxy();
    String uri = "https://localhost:8443/wsrf/services/cagrid/GridGrouper";
    String member = "/O=OSU/OU=BMI/OU=caGrid/OU=Dorian/OU=cagrid05/CN=jdoe";
    String group = "MyStem:MyGroup";

    // Create a Grid Grouper Instance
     GrouperI grouper = new GridGrouper(uri,adminProxy);

    // Obtain a handle to the group object
     GroupI mygroup = grouper.findGroup(group);

    //Remove the member to the group
    mygroup.deleteMember(SubjectUtils.getSubject(member));

    System.out.println("Successfully removed the user " + member + "from the group " +
group);
} catch (Exception e) {
    e.printStackTrace();
}
```

Chapter 5 caGrid Data Services

This chapter describes the caGrid Data Services infrastructure.

Topics in this chapter include:

- Overview on this page
- Manipulating CQL Query Results on page 76
- Utility Classes on page 78
- CQL Query Syntax on page 81
- Domain Model Conformance on page 82
- Results Validation on page 82
- CQL Query Processors on page 83
- Federated Query Processor Usage Overview on page 84
- API Details on page 88

Overview

The caGrid Data Services infrastructure provides generic client classes for invoking the base methods of a data service. These classes provide developers with a consistent means of accessing a data service without requiring the client class generated specifically for a service instance by the Introduce toolkit.

Two client classes are provided. The first is the gov.nih.nci.cagrid.data.client.DataServiceClient class. This class provides access to the `query()` method of any caGrid data service. This class is able to invoke any data service's query method because that method is provided to each service by a pre-defined WSDL and implementation class. This ensures that the messages necessary for the query invocation are in a consistent namespace and follow the same definition. A sample usage of this class is provided below:

```
import gov.nih.nci.cagrid.common.Utils;
import gov.nih.nci.cagrid.cqlquery.CQLQuery;
import gov.nih.nci.cagrid.cqlquery.Object;
import gov.nih.nci.cagrid.cqlresultset.CQLQueryResults;
import gov.nih.nci.cagrid.data.DataServiceConstants;

public class SampleDataServiceInvocation {

   public static void main(String[] args) {
      try {
         DataServiceClient client = new DataServiceClient(args[0]);
         CQLQuery query = new CQLQuery();
         Object target = new Object();
         target.setName("some.class.name");
```

```
                query.setTarget(target);
                CQLQueryResults results = client.query(query);
                Utils.serializeDocument("myResults.xml", results,
                    DataServiceConstants.CQL_RESULT_COLLECTION_QNAME);
            } catch (Exception ex) {
                ex.printStackTrace();
                System.exit(1);
            }
        }
    }
```

This small sample will create a new data service client using a URL specified on the command line and submit a query to it for all objects of the type "some.class.name". The results will be stored on disk in an XML document named "myResults.xml". The `DataServiceConstants` class used in this example provides static Strings and QNames used throughout the data service infrastructure. The constant `CQL_RESULT_COLLECTION_QNAME` is the QName which defines the XML type for result sets returned from the data service's query method.

The other client class provided with the data service infrastructure is the `gov.nih.nci.cagrid.data.enumeration.client.EnumerationDataServiceClient` class. This class provides uniform access to the `enumerationQuery()` method of a caGrid Data Service when the WS-Enumeration feature is enabled at service creation time. This client cannot be used with data services that do not have this feature enabled, and will throw an exception when the method is invoked against such services. Its usage is similar to that of the base data service client, but the query method returns an `EnumerateResponse` object, which contains an `EnumerationContext`. This can be used in conjunction with the Globus provided WS-Enumeration client implementation classes.

The client classes provided with the data service infrastructure, as well as any other clients generated by the Introduce toolkit, should not be assumed to be thread safe. Each thread communicating with a data service should have its own instance of the client class. Since client instances are unique, multiple data service clients may be used within the same thread or JVM to communicate with multiple data services simultaneously.

Manipulating CQL Query Results

Iteration

When a query is performed using the standard caGrid Data Service client's query method, a `CQLQueryResults` object is returned. This object is a container for both the results themselves and some information pertaining to their type. This container can contain object results, attribute name / value pairs, caBIG identifiers (not yet implemented), or a count of the total number of items in the result set. The difficulty of manipulating a container which may have such a wide variety of result types stored in it is handled by an iterator class provided with the data service infrastructure.

The class `gov.nih.nci.cagrid.data.utilities.CQLQueryResultsIterator` implements the `java.util.Iterator` interface, and so can be used in a `while()` loop like any other iterator over a Java collection. Depending on what the query to the data service asked

for, calls to the `next()` method of this iterator will return different types of objects.

- If the query was for object results, then:
  - the iterator returns objects of the type specified as the target for the query.
  - objects that need custom serialization and/or deserialization require that the iterator be configured with an `InputStream` to the `client-config.wsdd` file containing the type mappings for the objects.
  - alternatively, the iterator can be configured to return only the XML representation of those objects.
- If the query was for attribute results, including distinct attributes, then:
  - the iterator returns an array of `TargetAttribute` types. These types contain the name of the attribute and its value. The value is null if the value was null on the object satisfying the query. Each array of `TargetAttributes` corresponds to one object instance that satisfied the CQL query criteria.
- If the query was for a count of object instances, then:
  - the iterator returns a single `java.lang.Long` value.

An example usage of this iterator is below:

```
import gov.nih.nci.cagrid.cqlquery.CQLQuery;
import gov.nih.nci.cagrid.cqlresultset.CQLQueryResults;
import gov.nih.nci.cagrid.data.utilities.CQLQueryResultsIterator;

import java.util.Iterator;

public class SampleDataServiceInvocation {

   public static void main(String[] args) {
      try {
         DataServiceClient client = new DataServiceClient(args[0]);
         CQLQuery query = new CQLQuery();
         // build up the query
         CQLQueryResults results = client.query(query);
         Iterator iter = new CQLQueryResultsIterator(results,
            SampleDataServiceInvocation.class.getResourceAsStream(
               "client-config.wsdd"));
         while (iter.hasNext()) {
            java.lang.Object result = iter.next();
            // do something with the result object
         }
      } catch (Exception ex) {
         ex.printStackTrace();
         System.exit(1);
      }
   }
}
```

Utility Classes

Utilities

The caGrid data services infrastructure includes several utility classes which can be used to ease development and use of data services. These classes are found in the `gov.nih.nci.cagrid.data.utilities` package distributed with the data service infrastructure.

CastorMappingUtil

This class provides a function for rewriting a castor mapping XML file so that all classes in a specified package will have their namespace set to the one specified. This class is used internally in the data service extension to the Introduce toolkit to change the mapping file provided with caCORE services if the service developer specifies a schema other than the one expected by caCORE.

The class has only one public static method:

public static String changeNamespaceOfPackage(String mapping, String packageName, String namespace)

- mapping - this parameter is the full XML text of the castor mapping file
- packageName - the name of the package to change the namespace mapping of
- namespace - the namespace to remap the package's classes to

This method returns a String containing the modified text of the castor mapping file.

CQLResultsCreationUtil

This class provided convenience methods for creating CQLQueryResults objects for object results, attribute results, and a counting result. A convenience method for identifier results may be added in the future. The class provides three public static methods, one for each type of results currently supported.

public static CQLQueryResults createObjectResults(List objects, String targetName, Mappings classToQname)

- objects – a list of Java objects to be placed in a new CQLQueryResults object
- targetName - the name of the class targeted by the query which produced these object results. All items in the objects list should be of this type.
- classToQname – a mapping from class name to QName. This is a generated Java bean from the XML schema for the data service infrastructure and contains an array of name/value pairs that map class names to QNames.

public static CQLQueryResults createAttributeResults(List attribArrays, String targetClassname, String[] attribNames)

- attribArrays – a List of Object arrays. Each array should have one value for one attribute of an object. These values may be null. The values must be in an order corresponding the ordering of attribute names
- targetClassname – the name of the class targeted by the query which produced these attribute results. All attribute arrays should have some from this type.
- attribNames – the names of the attributes returned by the query. These should be in the same ordering used by the attribute arrays.

public static CQLQueryResults createCountResults(**long** count, String targetClassname)

- count – the number of resulting items (objects, attribute sets) from a query
- targetClassname – the name of the class which was the target of the query

DataServiceIterator

The data service iterator is an interface which provides for a query to be submitted to a data service and an Iterator over the result set to be returned. There are two implementations of this interface, one for the standard data service, and one for data services with enumeration enabled.

DataServiceHandle

The data service handle is the implementation of the data service iterator class for a base caGrid Data Service. It has three constructors, all of which take a DataServiceClient instance. The default constructor needs only this parameter. The other two constructors should be used when custom serialization and deserialization of types has been specified for the service. The extra parameter can be either the filename of a wsdd file containing this mapping information, or an InputStream to the same information.

EnumDataServiceHandle

The enum data service handle is the implementation of the data service iterator class for a WS-Enumeration enabled caGrid Data Service. It has two constructors, both of which take an enumeration data service client. The default constructor needs only this parameter. The second constructor takes an `IterationConstraints` instance, which contains information about how data should be requested from the enumeration data service.

DomainModelUtils

The domain model utils provide a means to extract useful information from a domain model.

public static UMLClass getReferencedUMLClass(DomainModel model, UMLClassReference reference)

To save on document size, domain models do not duplicate class information when an

association is defined, but rather use class references based on ID values. These reference values can be traced back to their original UML Class instance with this function.

public static UMLClass[] getAllSuperclasses(DomainModel model, String className)

Superclasses of a UML Class can be determined by traversing UML class references and generalization information. There are two methods which perform this task in the Domain Model Utils class. One uses a class name and the other extracts the name from an `UMLClass` instance and passes it to the other.

WsddUtil

The wsdd utility class contains functions to set parameters on a wsdd file. This class is used internally to the Introduce data service extension to edit the wsdd files and change the castor mapping file name.

public static void setGlobalClientParameter(String clientWsddFile, String key, String value)

- clientWsddFile - the name of the client side wsdd file to edit. When edits are complete, the changed file is saved to the same location.
- key - the key of the parameter. This is the name by which the parameter can be accessed.
- value - the value stored in the parameter

public static void setServiceParameter(String serverWsddFile, String serviceName, String key, String value)

- serverWsddFile - the name of the server side wsdd file to edit. When edits are complete, the changed file will be saved to the same location
- key - the key of the parameter. This is the name by which the parameter can be accessed
- value - the value stored in the parameter

CQL Query Syntax

The caGrid Data Service infrastructure provides mechanisms to validate CQL queries for syntactic correctness. While the Axis engine prevents malformed XML from ever being turned into CQL objects, it does not handle XML that doesn't conform to certain schema restrictions. For this reason, CQL syntax validation can be enabled on a caGrid data service. This mechanism will reject invalid queries before they ever reach a CQL Query Processor implementation, saving the processor's developer from having to handle them. This same validation can be performed either on the client side or offline completely by using the query validation utilities. For syntax validation, the interface `gov.nih.nci.cagrid.data.cql.validation.CqlStructureValidator` is provided, as are two implementations of this interface. The interface provides the `validateCqlStructure()` method, which takes a single `CQLQuery` instance parameter, and will throw a `MalformedQueryException` if an error is encountered. The default implementation of this interface is the `gov.nih.nci.cagrid.data.cql.validation.ObjectWalkingCQLValidator` class. As its name suggests, this class walks through the CQL object model, seeking out inconsistencies with the published CQL schema. This class also has a `main()` method, which allows it to be run from the command line with a list of CQL query XML files specified as arguments. The data service infrastructure uses this class by default when query validation is enabled. This can be changed for any other class which implements the `CqlStructureValidator` interface by editing the value of the

`dataService_cqlValidatorClass` service property in a generated data service.

Domain Model Conformance

The Data Service infrastructure also provides mechanisms to validate a structurally sound CQL query against a Domain Model to ensure its restrictions are supported by the domain model's exposed structure. Doman Model validation may be enabled for a caGrid data service, and will be performed on every query submitted to the service before it is passed to the CQL query processor. The interface `gov.nih.nci.cagrid.data.cql.validation.CqlDomainValidator` is provided, along with a single implementation. The interface provides the `validateDomainModel()` method, which takes a single `CQLQuery` instance parameter, and will throw a `MalformedQueryException` if an error is encountered. The lone implementation provided with the caGrid Data Service infrastructure is the `gov.nih.nci.cagrid.data.cql.validation.DomainModelValidator` class. Like the CQL validation instance, this class has a `main()` method, which allows it to be run from a command line. The arguments should be first a domain model XML file, then a list of CQL query files to be validated. The data service infrastructure uses this class when domain model validation is enabled. This implementation may be substituted for another by editing the value of the `dataService_domainModelValidatorClass` service property in a generated data service.

Results Validation

The data service infrastructure also provides a means to both validate the results of a CQL query against a known set of targets, and determine what target data types are allowed to be returned by a caGrid Data Service. Every data service exposes a schema through its WSDL which enumerates the data types which may be returned by the data service. This schema appears in generated services under the schemas/<ServiceName> directory as <ServiceName>_CQLResultTypes.xsd. The utility class `gov.nih.nci.cagrid.data.utilities.validation.CQLQueryResultsValidator` has been provided to both retrieve this file and verify that a `CQLQueryResults` instance conforms to this schema. An instance of this class can be constructed with either the full path to a data service's WSDL file, or an endpoint reference to a running data service

The validator exposes two public methods:

public void saveRestrictedCQLResultSetXSD(File fileLocation) **throws** SchemaValidationException

- fileLocation - a file into which the restriction XSD will be saved

This method locates the restriction XSD file and saves its contents into the file specified.

public void validateCQLResultSet(CQLQueryResults resultSet) **throws** SchemaValidationException

- resultSet - a set of results generated by a query into a caGrid Data Service. The object contents of this result set will be processed against the restriction XSD

The `CQLQueryResultsValidator` class also has a `main()` method, which takes two

arguments. The first argument is a URL to a caGrid Data Service, which will be used to retrieve the result restriction schema. The second argument should be the filename of a `CQLQueryResults` instance serialized to an XML document.

CQL Query Processors

Overview

The CQL query processor is the portion of all caGrid data services that links the provided grid interface to the backend data resource. A query processor implementation is reflect-loaded at runtime when a query is submitted to the data service. All query processor extend from a common base class, and must implement a method to process CQL and return a CQLResultSet. The CQL query processor is instantiated several times without actually processing a query so that various parts of the data service infrastructure and Introduce toolkit extension can extract information from it. For this reason, the constructor for a CQL query processor implementation should be as simple as possible.

Implementation

CQL Query Processors are designed to be configurable at runtime by a set of properties. These properties are modifiable via the data service extension to the Introduce toolkit, or manually by editing a configuration file once a service has been built. The base CQL query processor class provides a method to retrieve required configuration parameters and their associated default values.

```
public Properties getRequiredParameters()
```

This method is provided by default and returns an empty `java.util.Properties` object. CQL implementers who require properties should override this method to return a populated `Properties` object. If a property is optional, set its value to be an empty string. All property keys must start with a lowercase letter, and be valid Java identifiers. That means there cannot be any spaces or punctuation in the key.

The query processor base class has two protected methods which provide access to any user configured parameters and an input stream to the server side wsdd configuration file. `getConfiguredParameters()` returns a `java.util.Properties` instance containing all the keys defined in the properties returned by `getRequiredParameters()`, but with either the default or a user configured value specified for each. The method `getConfiguredWsddStream()` returns an `InputStream` instance which will read in the contents of the server side wsdd configuration file. The return values of these methods are populated just before the call to process a CQL query is made, and so will return null at any other time.

```
/**
    * Processes the CQL Query
    * @param cqlQuery
    * @return The results of processing a CQL query
```

```
     * @throws MalformedQueryException
     *       Should be thrown when the query itself does not conform to the
     *       CQL standard or attempts to perform queries outside of
     *       the exposed domain model
     * @throws QueryProcessingException
     *       Thrown for all exceptions in query processing not related
     *       to the query being malformed
     */
        public abstract CQLQueryResults processQuery(CQLQuery cqlQuery)
      throws MalformedQueryException, QueryProcessingException;
```

The only method which is required to be implemented by CQL query processors is the `processQuery()` method. This is the method which executes the CQL query against its data source and generates an appropriate set of results. There are utilities (discussed earlier) to make generation of this result set a simpler process. At the time this method is called, the return values of `getConfiguredParameters()` and `getConfiguredWsddStream()` will be non-null.

The `processQuery()` method throws both a `MalformedQueryException` and a `QueryProcessingException`. Malformed query exceptions should be thrown under conditions where the query is somehow incorrect syntactically, or uses features of the CQL language which are not yet supported in the query processor implementation. If query syntax validation is enabled in the data service infrastructure, then it may be assumed that all queries reaching the `processQuery()` method are at least well formed CQL. Query processing exceptions should be thrown when some error occurs which prevents successful resolution of the query request. These conditions may include database errors, file system problems, or misconfiguration of properties.

Federated Query Processor Usage Overview

The caGrid Federated Query Infrastructure provides a mechanism to perform basic distributed aggregations and joins of queries over multiple data services. As caGrid data services all use a uniform query language, CQL, the Federated Query Infrastructure can be used to express queries over any combination of caGrid data services. Federated queries are expressed with a query language, DCQL, which is an extension to CQL to express such concepts as joins, aggregations, and target services. The infrastructure is composed of a core engine and grid services which provide access to and management of the use of the core engine.

DCQL, the language used to express federated queries, is an extension to CQL, the language used to express single data service queries. Both CQL and DCQL use a declarative approach to describe the desired data by identifying the nature of the instance data with respect to its containing UML information model. That is, a query can be seen as identifying a class in a UML model, and restricting its instances to those which meet criteria defined over that class's UML attributes and UML associations.

The primary additions to CQL, which DCQL provides, are the introduction of the ability to specify multiple target services (aggregations), and the ability to specify object restrictions through relationships to Objects on remote data services (joins). The other primary difference between the languages is that CQL is context dependent, meaning the language must be interpreted against the service receiving the query, and DCQL itself specifies the context of the queries (by

identifying the target services). As such, services accepting DCQL (such as the FQP service), generally don't expose any local data. Details on DCQL can be found in the Federated Query Processor design document.

An example DCQL query, represented in XML, is shown below in Figure 5-1. In this fictitious example, a PersonRegistry Data Service is joined with a StudyRegistry Data Service. The query specifies Persons in the PersonRegistry should be returned which have an "ssn" that is equal to that of a Participant's "patientSSN" and the Participant should have an "age" greater than 18. The specification of the target service can be seen on line 18 in the example (in this case only one service is targeted, though may could have been listed). Additionally, the "join" is specified starting on line 6, wherein the second target service is identified, and the join condition is defined. The join condition creates a link between the containing Object (in this case, Person), and an Object (in this case Participant, as defined on line 10) in the second target service. The condition specifies a predicate to be evaluated against an attribute in each of the two linked Objects (in this case Person.ssn and Participant.patientSSN). It is worth noting that as DCQL is a recursive language, the ForeignObject defined on line 10 could have also specified a join to a third Data Service, or other more complex criteria.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <DCQLQuery xmlns:cql="http://CQL.caBIG/1/gov.nih.nci.cagrid.CQLQuery"
3      xmlns="http://caGrid.caBIG/1.0/gov.nih.nci.cagrid.dcql">
4      <TargetObject name="com.example.Person">
5          <Group logicRelation="AND">
6              <ForeignAssociation targetServiceURL=
7                  "https://host2:8443/wsrf/services/cagrid/StudyRegistry">
8                  <JoinCondition localAttributeName="ssn"
9                      foreignAttributeName="patientSSN" predicate="EQUAL_TO"/>
10                 <ForeignObject name="org.example.Participant">
11                     <Attribute name="age" value="18" predicate="GREATER_THAN"/>
12                 </ForeignObject>
13             </ForeignAssociation>
14             <Attribute name="lastName" value="Foo%" predicate="LIKE"/>
15         </Group>
16     </TargetObject>
17     <targetServiceURL>
18         https://host1:8443/wsrf/services/cagrid/PersonRegistry
19     </targetServiceURL>
20 </DCQLQuery>
```

Figure 5-1 Example DCQL Query

The Federated Query Engine is a simple but powerful design. The main functionality of the engine is to process a DCQL query by converting it into regular CQL queries to the targeted data services, appropriately aggregating results. As such, all of the actual "joining" of data is offloaded to the remote data services. This allows the engine to be reused as a client API as no databases or complex service infrastructure is needed; it's simply a client-side querying tool. The engine requires no special support from data services. Each service which is contacted to satisfy the distributed query is only sent one or more standard, but potentially complex, CQL queries. It is possible to construct a DCQL query which is essentially a standard CQL query, with the addition of specifying one or more target data services. In this case, the engine simply

"forwards" that query on to the targeted services, and aggregates their results. Details on the implementation and query processing logic of the engine can be found in the Federated Query Processor design document.

The Federated Query Processing Infrastructure contains three main client-facing components: an API implementing the business logic of federated query support, a grid service providing remote access to that engine, and a grid service for managing status and results for queries that were invoked asynchronously using the query service.

Federated Query Engine

For clients not wishing to use the grid service, the *FederatedQueryEngine* is the client-facing entry point to the engine. It provides two methods which accept DCQL queries, and return the results. Each of the two methods provides a different variant on how results are represented. The first method is the executeAndAggregateResults method, which returns the standard *CQLQueryResults* (the same result type returned by data services' query method). Each *CQLResult* obtained from each targeted data service is merged into an aggregate list, and a master CQLQueryResults object is constructed which contains them all. The information about which result came from which data service is lost in this scenario, but this provides the ability to reuse existing data service tooling and APIs when that information is not relevant.

In cases where it is important to know from which data service a given result came, the second query method called execute can be used. This method returns a new type called *DCQLQueryResultsCollection*. The *DCQLQueryResultsCollection* contains a list of *DCQLResult*, wherein each *DCQLResult* specifies a *CQLQueryResults* object, and the data service URL from which it came. That is, the result type is a collection of tuples containing the standard data service results, and that service's URL.

Both query methods will throw a RemoteDataServiceException in the event a queried data service returns invalid results (such as the wrong target class type), or if a data service itself throws an exception when being queried, or if there is any problem querying the data service. Additionally, a *FederatedQueryProcessingException*, which is the parent class of *RemoteDataServiceException*, may be thrown if there is a problem processing the query itself.

Federated Query Processor Service

The Federated Query Processor service is main service interface to the federated query engine. It provides three query execution operations. The first two are: *execute* which takes a DCQL query and returns a *DCQLQueryResultsCollection*, and *executeAndAggregateResults* which returns a *CQLQueryResults*. These are both simple grid service wrappers for the corresponding methods in the *FederatedQueryEngine* API. The third operation, *executeAsynchronously*, provides asynchronous, non-blocking, access to the *execute* method, and returns a *FederatedQueryResultsReference*. The *FederatedQueryResultsReference* is a typed container for an EPR to the Federated Query Results service. The Federated Query Results service client API can be used to subsequently retrieve the *DCQLQueryResultsCollection*.

Federated Query Results Service

The Federated Query Results service is the service responsible for providing access to query results and processing status for asynchronously executed queries. The service can only be contacted with a resource-qualified EPR, provided by the Federated Query Processor service. Whenever the query processor service is requested to execute an asynchronous query, a Resource is created and an EPR, which identifies that Resource in the results service, is returned. The Federated Query Results service's only purpose is to expose information about, and management of, these Resource instances. This Resource contains the current status of the query it corresponds to, any exceptions which occurred during processing, and eventually the results of the query. It supports standard WSRF Resource Lifetime behavior. As such, it exposes, as Resource Properties, the current time (as believed by the local system), and the termination time of the Resource. Once created, the resource will be terminated/destroyed by the service once its termination time is past. This lifetime is initially controlled by a setting in the grid service. The client can also immediately destroy the resource with the *Destroy* operation, or change its termination time with the *SetTerminationTime* operation. Both of these operations are standardized operations for resources supporting Resource Lifetime and as such corresponding common Resource Lifetime clients may be used (though the Federated Query Results client API also makes these operations available).

In addition to the operations and resource properties necessary to support Resource Lifetime on the resource, the service also provides the *getResults* and *isProcessingComplete* operations. The *isProcessingComplete* operation returns a simple Boolean value, indicating whether or not the query processing has completed. Once the query processing has completed, the results can be accessed via the *getResults* operation, which returns a *DCQLQueryResultsCollection*. If the operation is invoked prior to the processing being complete, a *ProcessingNotCompleteFault* fault will be thrown. If the processing is complete, but an exception occurred, a *FederatedQueryProcessingFault* will be thrown, and its cause will be the exception that occurred during query processing.

Security Considerations

The Federated Query Processor services support two deployment scenarios. The first is an insecure deployment, wherein no security (authentication or authorization) is enforced by the container. In this scenario, no encryption is used and no protection of query results is enforced. That is, anonymous communication is used over an open channel, and it is possible for one client to manipulate the query resources of another, given it knows the EPR.

The recommended second scenario is when the services are deployed securely, such as with transport level security (https). Deployments using transport level security ensure integrity and privacy of the communication channel (and obviously the data traveling over it). In this scenario, no authorization is performed by the query service, but any query results created via asynchronous queries are protected such that only the issuer of the query can view or manipulate the results. In this scenario, clients should use credentials to ensure proper protection of query results.

The services do not make use of delegated credentials, and as such, remote data services are accessed either or anonymously, or with the Federated Query Processor's credentials (depending on the deployment scenario and the settings of the remote data services). While future work may enable this feature, clients needing this capability (credentialed access of data services) now may leverage the federated query engine directly using the API.

API Details

gov.nih.nci.cagrid.fqp.client.FederatedQueryProcessorClient

The FederatedQueryProcessorClient is the main client interface to the federated query service (Figure 5-2).

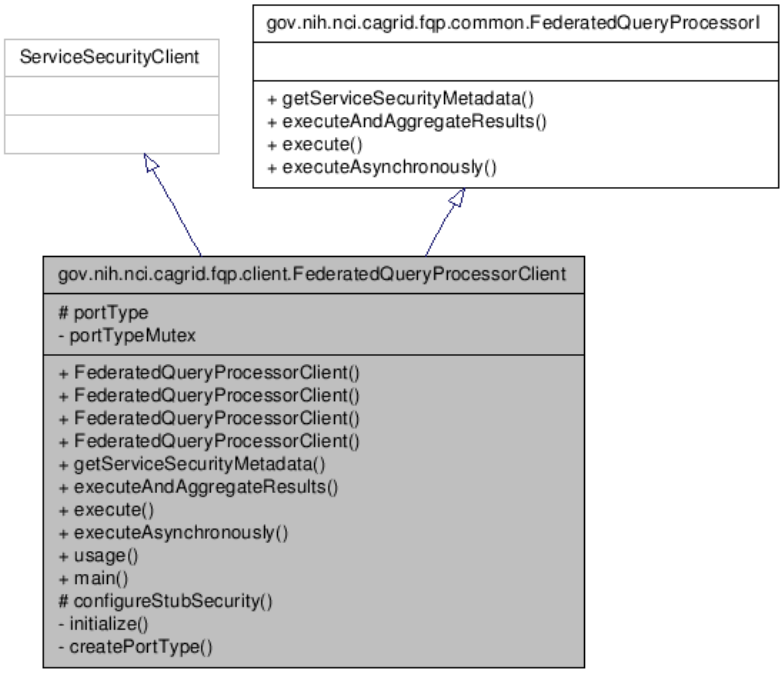


Figure 5-2 FederatedQueryProcessor Inheritance Model

Constructor Documentation

- **FederatedQueryProcessorClient** (String url)
  - throws MalformedURIException, RemoteException
- **FederatedQueryProcessorClient** (String url, GlobusCredential proxy)
  - throws MalformedURIException, RemoteException
- **FederatedQueryProcessorClient** (EndpointReferenceType epr)
  - throws MalformedURIException, RemoteException
- **FederatedQueryProcessorClient** (EndpointReferenceType epr, GlobusCredential proxy)
  - throws MalformedURIException, RemoteException

Member Function Documentation

Access to caGrid Service Security Metadata:

- gov.nih.nci.cagrid.metadata.security.ServiceSecurityMetadata **getServiceSecurityMetadata** ()
    - throws RemoteException
    - **Description**: Returns the caGrid service security metadata

Distributed Query Methods:

- gov.nih.nci.cagrid.cqlresultset.CQLQueryResults **executeAndAggregateResults** (gov.nih.nci.cagrid.dcql.DCQLQuery query)
    - throws RemoteException, gov.nih.nci.cagrid.fqp.stubs.types.FederatedQueryProcessingFault
    - **Description**: Executes the DCQL query, aggregating and returning them as standard data service results
- gov.nih.nci.cagrid.dcqlresult.DCQLQueryResultsCollection **execute** (gov.nih.nci.cagrid.dcql.DCQLQuery query)
    - throws RemoteException, gov.nih.nci.cagrid.fqp.stubs.types.FederatedQueryProcessingFault
    - **Description**: Executes the DCQL query, aggregating and returning them as standard data service results
- **gov.nih.nci.cagrid.fqp.results.client.FederatedQueryResultsClient executeAsynchronously** (gov.nih.nci.cagrid.dcql.DCQLQuery query)
    - throws RemoteException, org.apache.axis.types.URI.MalformedURIException
    - **Description**: Executes the DCQL query asynchronously, returning a results client which can be used to access the results

gov.nih.nci.cagrid.fqp.results.client.FederatedQueryResultsClient

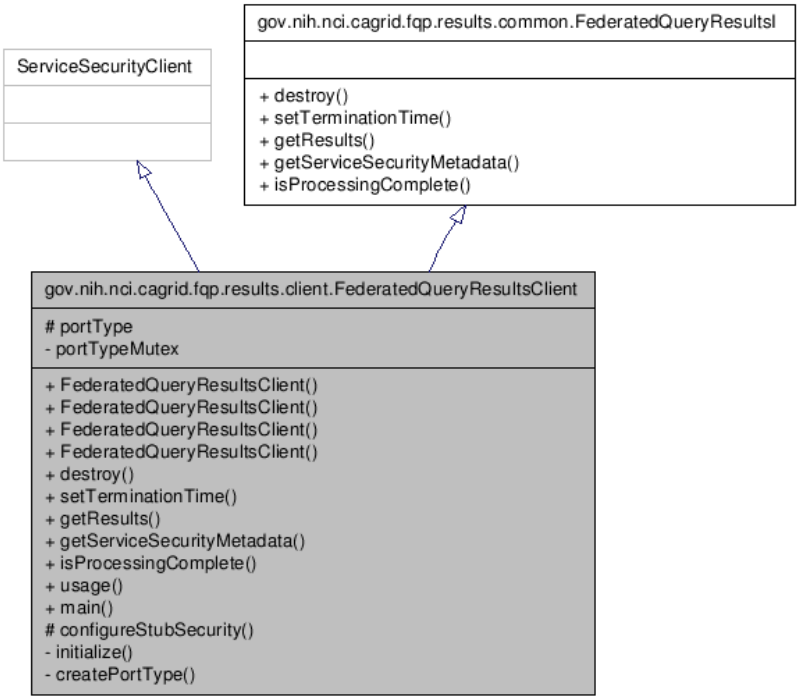


Figure 5-3 FederatedQueryResultsClient Inheritance Model

Constructor Documentation

- **FederatedQueryResultsClient** (String url)
  - throws MalformedURIException, RemoteException
- **FederatedQueryResultsClient** (String url, GlobusCredential proxy)
  - throws MalformedURIException, RemoteException
- **FederatedQueryResultsClient** (EndpointReferenceType epr)
  - throws MalformedURIException, RemoteException
- **FederatedQueryResultsClient** (EndpointReferenceType epr, GlobusCredential proxy)
  - throws MalformedURIException, RemoteException

Member Function Documentation

Access to caGrid Service Security Metadata:

- gov.nih.nci.cagrid.metadata.security.ServiceSecurityMetadata **getServiceSecurityMetadata** ()
  - throws RemoteException
  - **Description**: Not used.

Resource Lifetime Methods:

- org.oasis.wsrf.lifetime.DestroyResponse **destroy** (org.oasis.wsrf.lifetime.Destroy params)
  - throws RemoteException

    - **Description**: Destroys the corresponding resource and query results.
- org.oasis.wsrf.lifetime.SetTerminationTimeResponse **setTerminationTime** (org.oasis.wsrf.lifetime.SetTerminationTime params)
    - throws RemoteException
    - **Description**: Sets the time at which the resource and corresponding query results should be destroyed.

Query Results Methods:

- gov.nih.nci.cagrid.dcqlresult.DCQLQueryResultsCollection **getResults** ()
    - throws RemoteException, gov.nih.nci.cagrid.fqp.results.stubs.types.ProcessingNotCompleteFault, gov.nih.nci.cagrid.fqp.stubs.types.FederatedQueryProcessingFault, gov.nih.nci.cagrid.fqp.stubs.types.InternalErrorFault
    1. **Description**: Returns the query results, if processing is complete. If processing is not complete, throws ProcessingNotCompleteFault. If processing is complete, but threw an exception, that exception is then rethrown.
- boolean **isProcessingComplete** ()
    - throws RemoteException
    - **Description**: Returns true if and only if processing of the query is complete.

gov.nih.nci.cagrid.fqp.processor.FederatedQueryEngine

Constructor Documentation

gov.nih.nci.cagrid.fqp.processor.FederatedQueryEngine.FederatedQueryEngine ()

Default constructor

Member Function Documentation

DCQLQueryResultsCollection

gov.nih.nci.cagrid.fqp.processor.FederatedQueryEngine.execute (DCQLQuery

dcqlQuery) throws FederatedQueryProcessingException

Call Federated Query Processor, and send the generated CQLQuery to each targeted service, placing each result into a single DCQLQueryResults object.

Parameters:

dcqlQuery

Returns:

Exceptions:

FederatedQueryProcessingException

CQLQueryResults

gov.nih.nci.cagrid.fqp.processor.FederatedQueryEngine.executeAndAggregateResults (DCQLQuery *dcqlQuery*) throws FederatedQueryProcessingException

Call Federated Query Processor, and send the generated CQLQuery to each targeted service, aggregating the results into a single CQLQueryResults object.

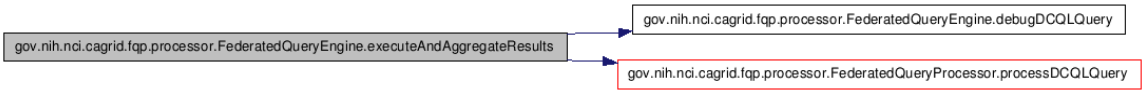
Parameters:

dcqlQuery

Returns:

Exceptions:

FederatedQueryException



API Usage Examples

The examples below show various uses of the federated query infrastructure; both execution the grid services and the engine's stand alone API. The exception handling is omitted from the examples for demonstration purposes. Additionally, the result processing shown is simplistic. The results returned from the federated query APIs, *DCQLResultsCollection* instances, are simple container objects for standard Data Service results, *CQLQueryResults*. A *DCQLResult* is just a *CQLQueryResult* and the address of the service which created the result. Additional details on how Data Service results can be processed can be found in the Data Service section of this document.

Executing a Blocking Query

The first example, shown below in Figure 5-4, demonstrates the simplest case of executing a query using the Federated Query Processor grid service, via the *FederatedQueryProcessorClient*. An instance of the client is constructed on lines 1 and 2, wherein the address of the service is provided. Next, a query document is created by loading it from a file (in this case the "exampleDCQL.xml" file) on the local file system. The caGrid

common utilities are used to accomplish this on lines 4 and 5. Though not shown, DCQL queries can also be constructed programmatically, just as CQL queries can. On line 7, the service is requested to perform the query as described in the *DCQLQuery*; this operation will block until the processing is complete. The result of the invocation is a *DCQLResultsCollection*, which contains an array of *DCQLResult*. Lines 10-16 loop over these results, and then print out the URL of service which yielded the result, and the number of data instances it produced.

```
1    FederatedQueryProcessorClient queryClient = new FederatedQueryProcessorClient(
2        "http://localhost:8080/wsrf/services/cagrid/FederatedQueryProcessor");
3
4    DCQLQuery dcql = (DCQLQuery) Utils.deserializeDocument(
5        "exampleDCQL.xml", DCQLQuery.class);
6
7    DCQLQueryResultsCollection dcqlResultsCol = queryClient.execute(dcql);
8    DCQLResult[] dcqlResults = dcqlResultsCol.getDCQLResult();
9    if (dcqlResults != null) {
10       for (int i = 0; i < dcqlResults.length; i++) {
11           DCQLResult result = dcqlResults[i];
12           String targetServiceURL = result.getTargetServiceURL();
13           System.out.println("Got results ("
14               + result.getCQLQueryResultCollection()
15               .getObjectResult().length + ") from:"+ targetServiceURL);
16       }
17   } else {
18       System.out.println("Got no results.");
19   }
```

Figure 5-4 Executing a Distributed Query

Executing a Non-Blocking Query

The next example, shown in Figure 5-5, illustrates the capability of the federated query service infrastructure to execute queries in the background, and allow clients to process the results later (suitable for long running queries, or distributed processing of results). It begins to diverge from the example above when it invokes *executeAsynchronously* on line 8 (as opposed to execute, as shown in Figure 5-4). This operation returns a new instance of the *FederatedQueryResultsClient*, which can be used to access the results once they are completed. This instance is returned before the service actually starts processing the query. The *FederatedQueryProcessorClient* abstracts the details that an EPR was actually returned by the service, and it directly provides the caller the appropriate API to communicate with the stateful service container the not yet populated query results. Lines 10-15 demonstrate checking the status of the processing, printing out a "." to the screen each second until the processing is complete. Future versions of the service will support subscriptions and notifications of query status and completion. The *isProcessingComplete* operation will return false until the query results are available, or the processing is terminated by failure. Once processing is complete, the results can be accessed via the *getResults* operation, shown on line 17. This returns a DCQLQueryResultsCollection, just as execute did in Figure 5-4. At this point, in terms of result processing, there is no longer a difference between asynchronous (this example) and synchronous (the previous example) query logic. However, to further demonstrate how to

process the results, the example below loops over each DCQLResult (as before), and prints out the service which yielded it (on line 23). It then, on line 24-27, accesses the standard Data Service query result type, and constructs a *CQLQueryResultsIterator* iterate over each Object result. This iteration, shown on lines 29-33, prints the XML representation of each Object, as the iterator was constructed to use "xml only" by passing true to the second argument of its constructor on line 27.

```
1   FederatedQueryProcessorClient queryClient = new FederatedQueryProcessorClient(
2       "http://localhost:8080/wsrf/services/cagrid/FederatedQueryProcessor");
3
4   DCQLQuery dcql = (DCQLQuery) Utils.deserializeDocument(
5       "exampleDCQL.xml", DCQLQuery.class);
6
7   FederatedQueryResultsClient resultsClilent =
8       queryClient.executeAsynchronously(dcql);
9
10  System.out.print("Waiting for completion");
11  while (!resultsClilent.isProcessingComplete()) {
12      Thread.sleep(1000);
13      System.out.print(".");
14  }
15  System.out.println("\nProcessing Complete.");
16
17  DCQLQueryResultsCollection dcqlResultsCol = resultsClilent.getResults();
18  DCQLResult[] dcqlResults = dcqlResultsCol.getDCQLResult();
19  if (dcqlResults != null) {
20      for (int i = 0; i < dcqlResults.length; i++) {
21          DCQLResult result = dcqlResults[i];
22          String targetServiceURL = result.getTargetServiceURL();
23          System.out.println("Got results from:" + targetServiceURL);
24          CQLQueryResults queryResultCollection =
25              result.getCQLQueryResultCollection();
26          CQLQueryResultsIterator iterator =
27              new CQLQueryResultsIterator(queryResultCollection, true);
28          int resultCount = 0;
29          while (iterator.hasNext()) {
30              System.out.println("===== RESULT [" + resultCount++ + "] =====");
31              System.out.println(iterator.next());
32              System.out.println("===== END RESULT=====\n\n");
33          }
34      }
35  }
```

Figure 5-5 Executing a Non-Blocking Distributed Query

Removing Query Results

As mentioned above, the Federated Query Results grid service is a stateful WSRF service, and executing asynchronous queries with the Federated Query Processor service generates Resources on it. These resources house the query results for clients, and can be accessed multiple times. The example shown below in Figure 5-6, demonstrates how, after the client is done with the results, they should be removed from the service by invoking the destroy operation, of the FederatedQueryResultsClient, shown on line 12. The results will eventually "expire" and be automatically removed after a service-specified lifetime, unless their lifetime is extended as shown in Figure 5-7, but it is good practice to manually release unneeded

resources.

```
1    FederatedQueryProcessorClient queryClient = new FederatedQueryProcessorClient(
2        "http://localhost:8080/wsrf/services/cagrid/FederatedQueryProcessor");
3
4    DCQLQuery dcql = (DCQLQuery) Utils.deserializeDocument(
5        "exampleDCQL.xml", DCQLQuery.class);
6
7    FederatedQueryResultsClient resultsClilent =
8        queryClient.executeAsynchronously(dcql);
9
10   //   ... Process the results ...
11
12   resultsClilent.destroy(new Destroy());
```

Figure 5-6 Destroying Query Results

Scheduling Removal of Query Results

As mentioned above, query results will expire after a service-configure default lifetime. However, this can be controlled by the client by specifying a new termination time for the resource. The example shown below in Figure 5-7, demonstrates how one client may request the service to perform a federated query, set the lifetime of the yet to be created results, and hand off an EPR to those results to some other processing thread or client. In this example, the query is executed asynchronously just as before (on lines 7-8). However, rather than waiting on completion of the results, the example uses, on line 13, the setTerminationTime operation of the results client to request the results live on the service for 4 hours. This is accomplished by creating a Resource Lifetime standard SetTerminationTime request, and providing it a Calendar instance configured to be 4 hours later than the current time, as shown on lines 11 and 12. The result of this operation is a Resource Lifetime standard SetTerminationTimeResponse, which indicates the current time, as believed by the service, and the time at which the resource will be destroyed. The example prints these out, as well as the value of the local Calendar instance, on lines 18-23. Finally, the code demonstrates, on line 25, how an EPR to the result Resource can be accessed from the client with the getEndPointReference method.

It should be noted that if the client's system clock and the service's system clock have significant differences in believe of the current time, this example code could cause the resource to terminate either earlier or later than expected. In cases where this may be an issue for clients, the results service provides as a Resource Property, the current time as believed by the service. A client could access this resource property, and construct the Calendar instance from it, rather than using its local system clock.

```
1   FederatedQueryProcessorClient queryClient = new FederatedQueryProcessorClient(
2       "http://localhost:8080/wsrf/services/cagrid/FederatedQueryProcessor");
3
4   DCQLQuery dcql = (DCQLQuery) Utils.deserializeDocument(
5       "exampleDCQL.xml", DCQLQuery.class);
6
7   FederatedQueryResultsClient resultsClilent =
8       queryClient.executeAsynchronously(dcql);
9
10  SetTerminationTime termTime = new SetTerminationTime();
11  Calendar terminateAt = Calendar.getInstance();
12  terminateAt.add(Calendar.HOUR, 4);
13  termTime.setRequestedTerminationTime(terminateAt);
14
15  SetTerminationTimeResponse response =
16      resultsClilent.setTerminationTime(termTime);
17
18  System.out.println("Service's current time "
19      + response.getCurrentTime().getTime());
20  System.out.println("Requested termination time "
21      + terminateAt.getTime());
22  System.out.println("Scheduled termination time "
23      + response.getNewTerminationTime().getTime());
24
25  EndpointReferenceType resultEPR = resultsClilent.getEndpointReference();
26  // ... Hand off result EPR to some other process or client ...
```

Figure 5-7 Scheduling the Destruction of Query Results

Using the Engine Directly to Access Protected Data Services

The final example, shown below in Figure 5-8, demonstrates how the federated query infrastructure can be used to locally execute federated queries (as opposed to requesting the service to execute them). While most clients will opt to use the service interfaces, there are many reasons why a client may wish to invoke the engine locally, such as minimizing data movement. The most common reason to use the engine locally is if the Data Services being targeted requires authorization to access the data of interest. The federated query processor service infrastructure does not currently have the capability to assume the identity of the client requesting the query; it queries Data Services either anonymously, or with its own credentials (depending on deployment scenarios). When executing the engine locally, the client has the ability to use credentials when it queries data services. When executing locally, the engine will make use of the "default" Globus credentials if the Data Service being accessed does not allow anonymous access. Consult the caGrid security documentation on how to get and set default Globus credentials.

Using the engine locally is very similar to using the processor service. The engine can be constructed, as shown on line 1, by instantiating a *FederatedQueryEngine*. This has the same query execution methods as the grid service, except it does not provide asynchronous execution (as this is easily done locally). The example below shows the ability of the engine to aggregate DCQL results and return them as standard Data Service Results, shown on line 4. It then uses the *CQLQueryResultsIterator* to print the XML representation of the results, just as in Figure 5-5.

```
1    FederatedQueryEngine fqp = new FederatedQueryEngine();
2    DCQLQuery dcql = (DCQLQuery) Utils.deserializeDocument("exampleDCQL.xml",
3        DCQLQuery.class);
4    CQLQueryResults results = fqp.executeAndAggregateResults(dcql);
5    CQLQueryResultsIterator iterator = new CQLQueryResultsIterator(results, true);
6    int resultCount = 0;
7    while (iterator.hasNext()) {
8        System.out.println("=====RESULT [" + (resultCount++) + "] =====");
9        System.out.println(iterator.next());
10       System.out.println("=====END RESULT=====\n\n");
11   }
```

Figure 5-8 Invoking the FederatedQueryEngine Locally

Chapter 6 Reference Implementations

This chapter describes Reference Implementations, where caBIG-developed projects are aim to adopt the caGrid 1.0 infrastructure before it is released.

Topics in this chapter include:

- Overview on this page
- Objective on page 100
- Goals on page 100
- Assumptions on page 100
- High-Level Process on page 100
- Deliverables on page 101
- Test Bed on page 101
- caArray Gridification on page 102
- caBioconductor on page 102
- caTRIP on page 103
- GenePattern on page 104
- GeneConnect on page 105
- geWorkbench on page 106
- GridIMAGE on page 107

Overview

The primary goal of caGrid is to provide a middleware to share applications and data in the Cancer Research community. Based on the use cases collected in the inception phase, the caGrid team implemented an automated process to virtualize caBIG resources like data and analytical tools. The implementation includes advertisement, discovery, query, and invocation with secure or without secure services. Current grid technologies are very rich in functionality but, because of the complexity, are difficult to use; caGrid project is using grid and service oriented concepts to leverage Globus as the core infrastructure to virtualize caBIG resources.

The reference implementations consist of grid services and leverage core caGrid services in their applications. The overall objectives of having reference implementation are to gather feedback on features, identify bugs in the caGrid software, and provide grid examples to other caBIG development projects. Feedback on features was gathered through informal meetings with the development project teams, as well as using the bug tracker on GForge. This process is separate from the initial use case and requirements gathering that take place at the beginning of caGrid development. Bugs will be identified by both the development project teams in their use of the software, as well as by the caGrid team as the reference implementation services are incorporated into system and quality assurance tests. The reference implementation projects will be provided on GForge and be downloadable by other caBIG project developers.

This chapter describes the goals, process, objectives, and description of each reference implementation. Reference implementation projects are listed in Table 11-1:

Project	Institution	Service Type
caArray	NCICB	Data

Project	Institution	Service Type
caBioconductor	Fred Hutchinson Cancer Research Center	Analytical
caTRIP	Duke Comprehensive Cancer Center	Data
GenePattern	Broad Institute	Analytical
GeneConnect	Washington University	Data
geWorkbench	Columbia University	Analytical
GridIMAGE	Ohio State University	Analytical

Table 6-1 Reference implementation projects

Objective

Virtualize caBIG reference implementations by using the process implemented in the caGrid project.

Goals

1. To have at least four caBIG applications virtualized with caGrid infrastructure, which should include at least two data services and two analytical services.
2. Test and update the virtualization process to expose caBIG resources available to caBIG community.
3. To validate compatibility document and provide recommendations.

Assumptions

1. Reference implementations have their domain models registered in the caDSR.
2. Reference implementations have exposed at least one caGrid service and it is advertising to the production index service.
3. Reference implementations have begun the silver compliance process.
4. The caGrid team has reference implementation schedules to plan the testing.
5. Access to computational resources in the research center where we are testing reference implementations.
6. Reference implementations leverage caGrid 1.0 beta or later codebase.

High-Level Process

For each institution that provides the reference implementations, the research center team with the assistance of the caGrid team will perform the following tasks:

1. Install the base technology stack (Java, Globus, Tomcat)
2. Test the Globus installation
3. Test connectivity with NCICB
4. Install caGrid
5. Test caGrid

6. Register the UML domain model
7. Construct one or more caGrid services
8. Plug business logic into the grid skeleton
9. Test and document

Deliverables

- caGrid resources virtualized by caGrid infrastructure
- caGrid virtualization process updates and improved based on lessons learned from testing caBIG resource virtualization.

Test Bed

The following deployment diagram (Figure 6-1) represents the infrastructure for the reference implementation. The diagram shows core services and Cancer research services. Core services are security, schema management (GME), and registry (Index services). Cancer research services are caBIO, caArray, PIR and rProteomics. Cancer research services are visualized using the caGrid infrastructure. caDSR and EVS systems can be accessed with the caBIO/caCORE service and with the caCORE client in a cancer research infrastructure.

Note: caDSR and EVS systems can be accessed by the Cancer research services either directly by using caCORE API's or indirectly by using respective business service layers.

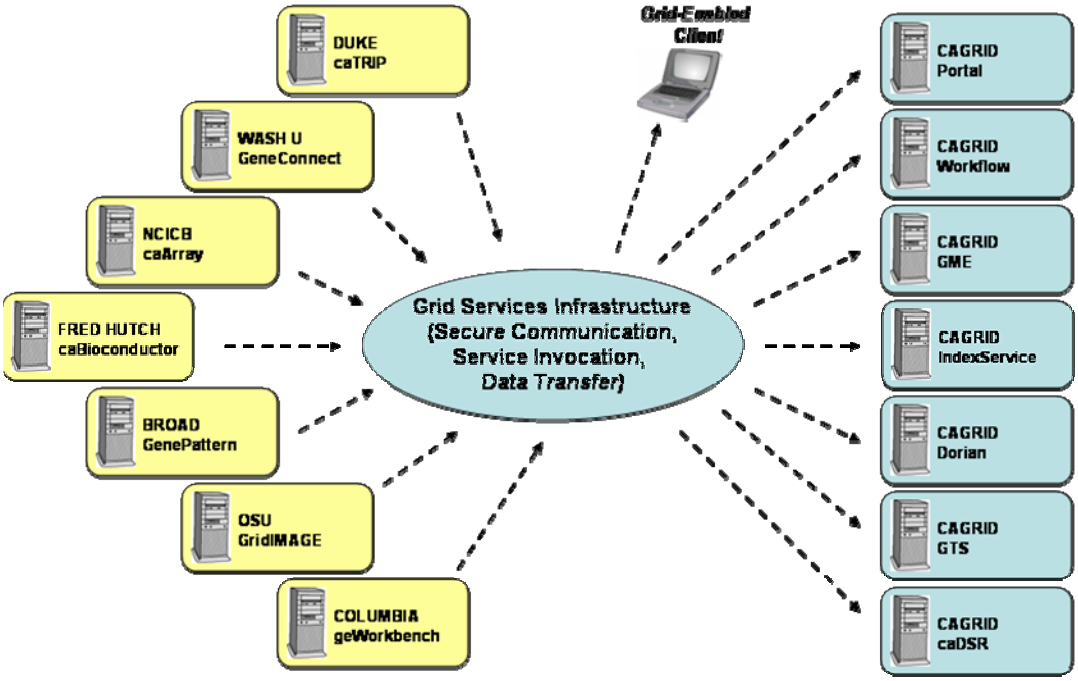


Figure 6-1 caGrid 1.0 Reference Implementation test bed

caArray Gridification

Introduction

NCICB's caArray project is a microarray database with open interfaces, strong security, and a user interface that is designed to make MIAME 1.1 level annotations as easy as possible. Its N-tier architecture will allow integration with other NCICB data sources: clinical data, animal models, genomic data, ontologies and controlled vocabularies. The caArray database can be deployed locally at the NCI designated cancer centers and other affiliated organizations. The system will allow day to day management and analysis of microarray data, and facilitates data exchange between research centers. The data can be easily migrated to the central caArray database at the NCI when the data is published.

caArray consists of a microarray database, web-portal and API for accessing microarray data. It is a standards based data repository of microarray experiment data using MIAME standard. The MIAME standard describes the Minimum Information about a Microarray Experiment that is needed to enable the interpretation of the experiment results unambiguously and, potentially, to reproduce the experiment.

Functionality Exposed to the Grid

caArray is a data service. All objects within the MAGE-OM which are exposed by the API are accessible.

Project Information

Institution: NCICB

Project team:

- Joshua Phillips (joshua.phillips@semanticbits.com)
- Vinay Kumar (vinay.kumar@semanticbits.com)

Project link: http://caarraydb.nci.nih.gov/caarray/

caBioconductor

Introduction

Bioconductor is a collection of open-source software components based on the R programming language. Bioconductor is used for gene expression and other high-throughput analysis in molecular biology. R packages are collections of algorithms grouped to facilitate particular analyses.

The caBioconductor module allows R package developers to expose the functionality of their package as analytic services on caGRID. The primary concern of the project is the development of tools for converting existing Bioconductor packages to caGRID analytic services. As proof of concept, the caBioconductor team is exposing portions of three Bioconductor packages as

caGrid 1.0 analytical services.

Functionality Exposed to the Grid

caBioconductor is an analytical service. It exposes functions for normalization of gene expression data derived from the Affymetrix platform (caAffy; based on the 'expresso' functionality of the affy Bioconductor package), mass-spec peak identification (caPROcess; based on the workflow of PROcess), and DNA copy number variation (caDNAcopy, based on the workflow of DNAcopy).

Project Information

Institution: Fred Hutchinson Cancer Research Center

Project team:

- Martin Morgan (mtmorgan@fhcrc.org)
- Nianhua Li (nli@fhcrc.org)
- Robert Gentleman (rgentlem@fhcrc.org)
- Seth Falcon (sfalcon@fhcrc.org)
- Duncan Temple Lang (duncan@wald.ucdavis.edu)

Project link: http://gforge.nci.nih.gov/projects/bioconductor/

caTRIP

Introduction

The Cancer Translational Research Informatics Platform (caTRIP) project aims to solve the difficult translational research problem of outcomes analysis. This involves the querying of a number of different data elements, such as pathology biomarkers, as well as dates of diagnosis, treatment, and death. A number of different grid services will be queried in a metadata-driven manner, including caTissue CORE, CAE, the Duke Tumor Registry, and the caIntegrator SNP database.

caTRIP is divided into three major components: domain services, distributed query engine, and graphical user interface. The domain services are implemented using the individual domain models from a number of existing tools, including caTissue CORE, CAE, caTIES, and caIntegrator. The caTRIP team has created a new domain model to capture and expose Tumor Registry data. These domain models and tools will be used to expose data from Duke's legacy systems, namely MAW3, the Tumor Registry, and GEMS datasets. The domain services themselves will be exposed as grid services, with users being authenticated by a Duke Identity Provider and authorized by custom modules potentially plugged into the CSM. A distributed query engine has been developed and incorporated into caGrid 1.0 to digest a CDE/Object based distributed query into individual queries that will be sent to each domain service. The

engine will then perform joins based on common data elements across the results and return resulting objects as its output. It will be exposed as a secure grid service that will delegate user credentials to the federated domain services. The graphical user interface will be used to compose, store, and retrieve queries. These queries will be built graphically using common data elements. Results will be displayed to users in a tabular fashion and will support drill-down features for data mining.

Functionality Exposed to the Grid

caTRIP consists of four data services. It exposes caTissue CORE, CAE, Tumor Registry, and caIntegrator SNP services.

Project Information

Institution: Duke Comprehensive Cancer Center

Project team:

- Patrick McConnell (patrick.mcconnell@duke.edu)
- Ram Chilukuri (ram.chilukuri@semanticbits.com)
- Srini Akkala (srini.akkala@semanticbits.com)
- Bill Mason (bmason@5amsolutions.com)
- Sanjeev Agarwal (sanjeev.agarwal@semanticbits.com)

Project link: http://gforge.nci.nih.gov/projects/catrip/

GenePattern

Introduction

GenePattern is a flexible analysis platform developed to support multidisciplinary biomedical research. GenePattern puts the power of sophisticated computational methods into the hands of non-programming users. It also provides an environment for rapid development and deployment of new analytic techniques.

GenePattern has a modular architecture that allows the inclusion of additional analytic or visualization modules. To be integrated into GenePattern, modules must provide a command-line interface allowing them to be called and have parameters passed to them from a normal command line. This facilitates the independent development and testing of Modules external to the GenePattern environment.

GenePattern has been publicly available since January 2003. Development of the GenePattern Server and most of its modules has been supported by NIH grants outside of the caBIG system and development of the GenePattern server and modules remains external to caBIG. Currently GenePattern is in use by over 2600 individuals in 500+ organizations throughout the world. While it is primarily used for micro array and proteomic analysis, GenePattern has also been adapted for use in computational chemistry, materials science and other application areas.

The general approach for exposing GenePattern modules is to create an application proxy layer that 'speaks' both caGRID protocols and the GenePattern web service protocol. The modules to be exposed as caGRID analytical services include:

- Preprocess Dataset
- Gene Set Enrichment Analysis (GSEA)
- Comparative Marker Selection.

Functionality Exposed to the Grid

GenePattern is an analytical service. It exposes functionality for gene expression analysis.

Project Information

Institution: Broad Institute

Project team:

- Ted Liefeld (liefeld@broad.mit.edu)
- Jared Nedzel (jnedzel@broad.mit.edu)

Project link: http://gforge.nci.nih.gov/projects/genepattern/

GeneConnect

Introduction

The NCI caBIG™ project is creating a common, extensible informatics platform that integrates diverse data types and supports interoperable analytic tools. This platform will allow research groups to tap into the rich collection of emerging cancer research data while supporting their individual investigations. However, it is very likely that joins cannot be performed across many developer projects because they will use non-overlapping sets of genomic identifiers in their object models. For example, while the Function Express project may use the Entrez Gene ID to reference genes, the Cancer Molecular Pages project may employ the Ensembl Gene ID to reference genes.

GeneConnect is the mapping service that will facilitate interoperability by interlinking VCDE approved genomic identifiers (http://gforge.nci.nih.gov/frs/?group_id=108 and https://cabig.nci.nih.gov/workspaces/VCDE/Data_Standards/GeneIdentifier.zip). These include:

- Ensembl Gene ID
- Ensembl Transcript ID
- Ensembl Protein ID
- Entrez Gene ID
- UniGene ID
- GenBank mRNA Accession Number

- GenBank Protein Accession Number
- RefSeq mRNA Accession Number
- RefSeq Protein Accession Number
- UniProtKB Primary Accession Number

To interlink all of these identifiers, database annotations will be used to construct pair wise connections, and then all-to-all relationships will be calculated by traversing all possible combinations of edges in the graph using every node as the starting point. Next, using reciprocality calculations performed over the data sources, the set of non-commutable results will be determined with each record given a confidence score based on the ratio of commutable results (i.e. all the graphs whose identifier at each position is identical) to all possible combinations for set of the genomic identifiers under question.

Functionality Exposed to the Grid

GeneConnect is a data service. It exposes functionality for searching the genomic identifiers mapping space.

Project Information

Institution: Washington University

Project team:

- Rakesh Nagarajan (rakesh@wustl.edu)
- Sachin Lale (sachin_lale@persistent.co.in)
- Mahesh Nalkande (mahesh_nalkande@persistent.co.in)
- Madhurima Bhattacharjee (madhurima_b@persistent.co.in)
- Pratibha Dhok (Pratibha_dhok@persistent.co.in)
- Srikanth Adiga (Srikanth_adiga@persistent.co.in)
- Krunal Thakkar (krunal_thakkar@persistent.co.in)

Project link: http://gforge.nci.nih.gov/projects/geneconnect/

geWorkbench

Introduction

geWorkbench is an open source bioinformatics platform written in Java that makes sophisticated tools for data management, analysis and visualization available to the community in a convenient fashion. It evolved from a project which was originally sponsored by the National Cancer Institute Center for Bioinformatics (NCICB). Some of the most fully developed capabilities of the platform include access to caArray repositories through use of the MAGEOM API, microarray data analysis, pathway analysis and reverse engineering, sequence analysis, transcription factor binding site analysis, and pattern discovery. geWorkbench will integrate into

the caGrid framework by providing a simple user interface to connect interoperable grid-enabled and client-side components into useful workflows, as well expose analytical services.

Functionality Exposed to the Grid

geWorkbench is an analytical service. It exposes functionality for gene expression analysis.

Project Information

Institution: Columbia University

Project team:

- Kiran Keshav (keshav@c2b2.columbia.edu)
- Aris Floratos (floratos@c2b2.columbia.edu)

Project link: http://gforge.nci.nih.gov/projects/geworkbench/

GridIMAGE

Introduction

GridIMAGE enables (a) users to select images from multiple geographically distributed data sources, (b) facilitate evaluation of selected images by human readers or computer-assisted detection (CAD) algorithms, (c) provide a graphical user interface to review human and CAD evaluations.

Functionality Exposed to the Grid

GridIMAGE is an analytical service. It exposes functionality for retrieving, analyzing, and storing image data.

Project Information

Institution: Ohio State University

Project team:

- Tony Pan (tpan@bmi.osu.edu)
- Ashish Sharma (ashish@bmi.osu.edu)
- Metin Gurcan (gurcan@bmi.osu.edu)

Project link: http://bmi.osu.edu/

Chapter 7 WS-Enumeration

This chapter describes the client-side APIs for enumerations.

Topics in this chapter include:

- Overview on this page
- Client API on this page

Overview

An overview of WS-enumeration is beyond the scope of this document. For more information, see the following websites:

Specification:

http://www.w3.org/Submission/WS-Enumeration/

Schema:

http://schemas.xmlsoap.org/ws/2004/09/enumeration/enumeration.xsd

WSDL:

http://schemas.xmlsoap.org/ws/2004/09/enumeration/enumeration.wsdl

Client API

There are two main client-side APIs for enumerations. The ClientEnumeration API provides basic functions for managing enumeration lifetime and retrieving its data. The ClientEnumIterator API provides java.util.Iterator abstraction for retrieving enumeration data and supports automatic data deserialization.

ClientEnumeration

The ClientEnumeration API provides basic functions for managing enumeration lifetime and retrieving its data. ClientEnumeration must be initialized with a javax.xml.rpc.Stub instance that is a Stub for the service that implements the WS-Enumeration operations and with an EnumerationContextType object returned by the enumerate operation of the service or any other operation that initiates an enumeration.

The javax.xml.rpc.Stub instance must define all of the WS-Enumeration operations except the enumerate operation. Also, the Stub instance must be properly configured with the security properties if calling a secure service.

IterationResult pull(IterationConstraints constraints)

Retrieves the next set of elements of the enumeration. The input parameter defines the

constraints for the operation such as the maximum number of elements to retrieve, the maximum number of characters that the consumer can accept, and the maximum amount of time in which the data needs to be returned. The return parameter contains the results of the iteration and an end of sequence flag to indicate if there is more data to be returned. The results of the iteration are of the *javax.xml.soap.SOAPElement* type.

This method calls the WS-Enumeration *pull* operation on the data service.

IterationResult pull()

Same as *pull(IterationConstraints)* function but uses default constraints (maximum number of elements set to 1, no maximum characters limit and no time limit).

void release()

Explicitly releases the enumeration. In general, the enumeration context is automatically released when a client finishes retrieving all the enumeration data or the enumeration expires if it was configured with an expiration time or duration. In cases where no expiration time was set for the enumeration or when not enumerating over the entire data the enumeration should be released explicitly.

This method calls the WS-Enumeration *release* operation on the data service.

EnumExpiration renew(EnumExpiration expiration)

Sets a new expiration time/duration of the enumeration. The input parameter can be null to configure the enumeration without an expiration time/duration (the enumeration will not expire). The expiration time/duration cannot be in the past (as according to the service clock). The service can choose to accept a different expiration time then specified. The return parameter can also be null to indicate that the enumeration does not have an expiration time/duration.

This method calls the WS-Enumeration *renew* operation on the data service

EnumExpiration getStatus()

Gets the current expiration time/duration of the enumeration. The return parameter can be null to indicate that the enumeration does not have an expiration time/duration.

This method calls the WS-Enumeration *getStatus* operation on the data service

ClientEnumIterator

The *ClientEnumIterator* API provide simple-to-use API for enumerating over data using the WS-Enumeration operations. The *ClientEnumIterator* class implements the *java.util.Iterator* interface but the implementation of these functions does not follow the *Iterator* contract exactly because of the WS-Enumeration specification limitations. The *ClientEnumIterator* class uses the *ClientEnumeration* API underneath and in contrast to the *ClientEnumeration* API offers automatic data deserialization.

The *ClientEnumIterator* must be initialized with a *javax.xml.rpc.Stub* instance that is a Stub for the service that implements the WS-Enumeration operations and with an

EnumerationContextType object returned by the *enumerate* operation of the service or any other operation that initiates an enumeration.

The *javax.xml.rpc.Stub* instance must define all of the WS-Enumeration operations except the *enumerate* operation. Also, the Stub instance must be property configured with the security properties if calling a secure service.

During iteration the *ClientEnumIterator* makes remote calls to the data service to retrieve the next set of items (calls the WS-Enumeration *pull* operation). The frequency of these remote calls is controlled by the *maxElements* setting of the *IterationConstraints* of the *ClientEnumIterator*. If that number is small the *ClientEnumIterator* will make a lot of remote calls but will use a small amount of memory (since it always keeps a few of the items in the memory). But if that number is big the *ClientEnumIterator* will make fewer remote calls but will use more memory (since it now keeps more items in the memory).

void setItemType(Class itemType)

Sets the type of the object returned by the *next()* operation. By default the objects returned by the *next()* operation will be of the *javax.xml.soap.SOAPElement* type. If the item type is set, the enumeration elements will be automatically deserialized into this type. This assumes the enumeration elements are all of the same type.

void setIterationConstraints (IterationConstraints constraints)

Sets iteration constraints for the iterator. By default the constraints are not set and defaults are assumed (maximum number of elements set to 1, no maximum characters limit and no time limit).

Object next()

Returns the next object in the enumeration. The returned object can be null. If item type is set (using the *setItemType* method) the current object will be automatically converted into that type and returned. Otherwise, object of the *javax.xml.soap.SOAPElement* type is returned. If the enumeration has ended (*hasNext()* returns *false*) or has been released on the client the *NoSuchElementException* is raised. Also, in certain cases the *NoSuchElementException* can also be raised even though *hasNext()* returned *true*. If deserialization is performed and it fails a *ConversionException* is raised and the index of the iteration is not advanced (so that the user can specify another item type or disable deserialization).

This method calls the WS-Enumeration *pull* operation on the data service.

boolean hasNext()

Tests if there are more elements in the enumeration to be returned. If it returns *false*, there are no more elements to be returned. If *true*, there **might** be more elements to be returned. This method can return *true* even though the *next()* operation consistently returns null. Also, this method can return *true* even though the *next()* operation will throw *NoSuchElementException*.

Object convert(SOAPElement element)

Performs object conversion on the enumeration element. This function is called by the *next()* function every time the *next()* function is called. It can be used to deserialize the enumeration element into appropriate Java object. This function is meant to be overwritten by the subclasses of the *ClientEnumIterator* to provide custom deserialization behavior. If deserialization fails a *ConversionException* is thrown.

void release()

Explicitly releases the enumeration context.

This method calls the WS-Enumeration *release* operation on the data service. *hasNext()* will return *false* and *next()* will throw *NoSuchElementException* after this method is called.

Examples

ClientEnumeration

The following example shows how to iterate over the data using ClientEnumeration API (Figure 7-1).

```
import org.globus.ws.enumeration.ClientEnumeration;
import org.globus.ws.enumeration.IterationResult;
import org.globus.ws.enumeration.IterationConstraints;
…

EnumerationServiceAddressingLocator locator =
            new EnumerationServiceAddressingLocator();

URL serviceURL =
  new URL("http://127.0.0.1:8080/wsrf/services/EnumerationService");

EnumerationPortType port =
   locator.getEnumerationPortTypePort(serviceURL);

// obtain the enumeration context from the service somehow
EnumerationContextType context = …

// create iteration constraints (return maximum of 10 elements)
```

```
IterationConstraints constraints =
   new IterationConstraints(10, -1, null);

// create client enumeration
ClientEnumeration enumeration =
   new ClientEnumeration((Stub)port, context);

// iterate over the data
IterationResult iterResult;
do {
   // retrieve the enumeration data with given constraints
   iterResult = enumeration.pull(constraints);
   Object [] items = iterResult.getItems();
   if (items != null) {
      // display the enumeration data
      for (int i=0; i < items.length; i++) {
          System.out.println(items[i]);
      }
   }
} while (!iterResult.isEndOfSequence());
```

Figure 7-1 Client enumeration example

ClientEnumIterator

This example shows how to iterate over the data using *ClientEnumIterator* API (Figure 7-2).

```
import org.globus.ws.enumeration.ClientEnumIterator;
import org.globus.ws.enumeration.IterationConstraints;

…

EnumerationServiceAddressingLocator locator =
            new EnumerationServiceAddressingLocator();
```

```
URL serviceURL =
  new URL("http://127.0.0.1:8080/wsrf/services/EnumerationService");

EnumerationPortType port =
   locator.getEnumerationPortTypePort(serviceURL);

// obtain the enumeration context from the service somehow
EnumerationContextType context = …

// create iteration constraints (return maximum of 10 elements)
IterationConstraints constraints =
   new IterationConstraints(10, -1, null);

// create the client iterator
ClientEnumIterator iterator =
   new ClientEnumIterator((Stub)port, context);

iterator.setIterationConstraints(constraints);

// iterate over the data
try {
   while(iterator.hasNext()) {
      Object obj = iterator.next();
   }
} catch (NoSuchElementException e) {
  // next() can throw this exception even though
  // hasNext() returned true
}
```

Figure 7-2 ClientEnumIterator example

Command Line Clients

ws-enumerate-start

Starts an enumeration. It calls the *enumerate* operation on the data service and prints out the enumeration context to the console. The enumeration context then can be passed to *ws-enumerate* or *ws-enumerate-end* clients.

ws-enumerate

Enumerates over the data. It calls the *pull* operation on the data service and prints out the retrieved data to the console. The client requires an argument that is a filename that contains the enumeration context (created either by *ws-enumerate-start* or other means).

The –n, --maxElements option can be used to configure the maximum number of elements to retrieve from the data service at a time. The –r, --maxCharacters option can be used to configure the maximum number of characters the client can accept at a time. The –n, --maxTime option can be used to specify the maximum amount of time in which the enumeration data has to be assembled. Any combination of these options can be specified at the same time.

ws-enumerate-end

Releases an enumeration. It calls the *release* operation on the data service. The client requires an argument that is a filename that contains the enumeration context (created either by *ws-enumerate-start* or other means).

Service

Service WSDL

The service that wishes to support enumerations must define the WS-Enumeration operations in its WSDL. All operations except the *enumerate* operation must be defined in service WSDL. The *enumerate* operation of WS-Enumeration specification is an optional operation and therefore it is up to the service designer to decide if the service should define and implement this operation or if the service will provide some other operation that will initiate an enumeration. Any operation of the service can initiate an enumeration by returning an element of the *wsen:EnumerationContextType* type to the client.

Service Implementation

The service must implement the *enumerate* operation of the WS-Enumeration specification or provide some other operation that will initiate an enumeration and return an element of the *wsen:EnumerationContextType* type to the client.

For all the other WS-Enumeration operations the service must be configured with the built-in enumeration operation provider (*EnumProvider)*. Of course, the service can choose to provide its own implementation for the WS-Enumeration operations but will need to replicate a lot of the built-in functionality.

The *EnumProvider* is configured in the same way as any other operation provider in the service deployment description (WSDD) file. All the WS-Enumeration operations should have the same security settings.

Enumeration Implementation Details

Internally, enumerations are managed and implemented just like any other WS-Resources. That is, there are enumeration resources (*EnumResource*) which are managed by the enumeration resource home (*EnumResourceHome).* The enumeration resources contain lifetime information and have a reference to the iterator (*EnumIterator*) that provides the actual data iteration functionality.

Types

There are two types of enumerations: transient and persistent. The transient enumerations live only while the container is running and are not restored after a container restart. The persistent enumerations are restored after a container restart. The type of enumeration has no impact on how the data of the enumeration is stored or retrieved. For example, a transient enumeration can query a database, retrieve data from a file, or have all the data in memory. It is entirely up to the service developers to decide how the data is retrieved, if the data is static or dynamic, etc.

In general it is not recommended to keep the entire enumeration data in memory. If the data is static, it is recommended to store the data in a database or a file, etc. and retrieve it in an efficient way.

Visibility

The enumeration resources also contain visibility properties (*VisibilityProperties*) to restrict what service and/or resource can access the particular enumeration resource. In general, an enumeration created by service S is only accessible through service S. Similarly, an enumeration created by resource R is only accessible through resource R.

Security

The service or resource through which the enumeration data is accessed can be configured with a security descriptor to further control access to the data.

Example

This example shows how to create a transient enumeration on the server-side with the help of the built-in WS-Enumeration operation provider (Figure 7-3).

```
import org.globus.ws.enumeration.EnumResourceHome;

import org.globus.ws.enumeration.EnumIterator;

import org.globus.ws.enumeration.EnumResource;
```

```
import org.globus.ws.enumeration.EnumProvider;
…

// obtain enumeration resource home
EnumResourceHome enumHome = EnumResourceHome.getEnumResourceHome();

// create iterator for the data
EnumIterator iter = …;

// create transient enumeration resource for the iterator
// with visibility properties obtained from the context
EnumResource resource = enumHome.createEnumeration(iter, false);

// get resource key for the enumeration resource
ResourceKey key = enumHome.getKey(resource);

// create EnumerationContextType to be returned to the client
EnumerationContextType enumContext =
        EnumProvider.createEnumerationContextType(key);
```

Figure 7-3 Example of creating a transient enumeration on the server-side

API

EnumIterator

This API is used by the service developers to write their own *EnumIterator* implementations in order the retrieve the enumeration data in a fast and efficient way.

A new *EnumIterator* instance must be created for each new enumeration. The implementations should assume a single thread access. Only one client is allowed to access a particular enumeration at a time. The implementation must keep track of the progress of the enumeration (for example, store the index of the last item retrieved).

For persistent enumerations, the *EnumIterator* implementation must be fully serializable using the Java serialization framework. That will enable the enumeration to be restored in case of a

container restart or in other conditions. An application should not keep references to the *EnumIterator* objects it creates. Such references will prevent efficient memory management by the *EnumResourceHome*.

IterationResult next(IterationConstraints constraints)

Retrieves the next set of items of the enumeration. The *IterationConstraints* define constraints for this operation such as the maximum number of the items that can be returned, the maximum number of characters of the items, and timeout in which the items must be returned. The constraints can change between the calls. If the timeout value constraint is specified and the data is not collected in that time, a *TimeoutException* should be raised. If there are no more elements in the enumeration a *NoSuchElementException* is raised.

The *IterationResult* contains the result of the iteration that fulfills the specified constraints. It must always be non-null. The *IterationResult* itself contains a list of enumeration items of *javax.xml.soap.SOAPElement* type and a flag that indicates if an end of sequence has been reached.

void release()

Release any resources associated with this enumeration. For example, close database connections, delete files, etc. This method is called when the enumeration is explicitly released, expires, or the user finished enumerating through all the data.

SimpleEnumIterator

The *SimpleEnumIterator* is a concrete implementation of the *EnumIterator* interface. It is a very simple implementation that can enumerate over in-memory data passed either as an array of objects or a list (*java.util.List*). The enumeration contents can be of *javax.xml.soap.SOAPElement* type, simple types such as *java.lang.Integer*, etc. or Axis generated Java beans.

The *SimpleEnumIterator* can only be used with transient type of enumerations.

IndexedObjectFileEnumIterator

The *IndexedObjectFileEnumIterator* is another concrete implementation of the *EnumIterator* interface. It is a memory efficient implementation that can enumerate over data stored in an indexed file created by *IndexedObjectFileWriter*. The indexed file format is optimized for retrieving objects in a sequential and random manner. The *IndexedObjectFileEnumIterator* uses the *IndexedObjectFileReader* to read the indexed file and quickly locate and retrieve the next set of objects of the enumeration.

The *IndexedObjectFileEnumIterator* can be used with transient and persistent types of enumerations.

IndexedObjectFileWriter

The *IndexedObjectFileWriter* is used to create an indexed file. The objects stored in the file will be serialized using the Java serialization framework, therefore, only the objects that implement the *java.io.Serializable* interface can be used.

IndexedObjectFileReader

The *IndexedObjectFileReader* is used to read an indexed file created by the *IndexedObjectFileWriter*. The objects stored in the file will be deserialized using the Java serialization framework.

IndexedObjectFileUtils

The *IndexedObjectFileUtils* is a collection of utility functions that can be used to create indexed files with the given data.

Other Implementation Details

WS-Enumeration WSDL and schema changes

The following changes have been made to the WS-Enumeration WSDL and schema files:

1. The WS-Addressing namespace used by the specification was changed to *http://schemas.xmlsoap.org/ws/2004/03/addressing* in order to work with the existing tooling.
2. The *EnumerationEndOp* operation was commented out as it violates WS-I Basic Profile 1.1 and is not supported by the tooling. Therefore, this part of WS-Enumeration functionality is not supported by the current implementation.
3. Since the *EnumerateOp* operation in is an optional operation, it was moved into a separate port type called *DataSourceStart*. All other operations remain in the *DataSource* port type.
4. The *EnumerationContextType* type was simplified to an equivalent form in order to be properly recognized by the tooling.

Other comments on the WS-Enumeration WSDL and schema files:

1. The schema file uses the *xsd:union* type which makes it hard for the tooling to figure out which value was actually serialized. The x*sd:choice* type might be better for such cases.
2. Currently there is no way to ask the data service if it has any more elements without actually retrieving some elements.
3. The *EndOfSequence* element in schema file should be defined with *xsd:boolean* type. Right not it defaults to *xsd:anyType*.

Chapter 8 Workflow Management Service

This chapter describes the architecture and APIs for interacting with caGrid workflow.

Topics in this chapter include:

- Overview on this page
- Workflow Architecture on this page
- WorkflowFactoryService API on page 123
- WorkflowManagementService API on page 125
- Security in WorkflowFactory and Context Services on page 128
- Service Selection on page 128
- Provenance Tracking on page 128
- WS-RF Resources in Workflows on page 128

Overview

caBIG aims to bring together disparate data and analytic resources into a “World Wide Web of cancer research.” This will be achieved through common standards and software frameworks for the federation of these resources into “grid” services. Many of the tasks in the collection and analysis of cancer-related data on the grid involve the use of workflow. Here, we define workflow as the connecting of services to solve a problem that each individual service could not solve. caGrid implements workflow by providing a grid service for submitting and running workflows that are composed of other grid services.

Workflow Architecture

The Workflow component leverages the same infrastructure stack as the caGrid toolkit (GT4, Tomcat, Java, Ant, and Introduce) with the addition of the ActiveBPEL workflow engine. The WorkflowFactoryService is a standard Introduce-built grid service that allows a workflow to be created from a BPEL workflow document. An EPR is returned to a WorkflowManagementService resource that can be used to start, stop, pause, resume, cancel, and destroy the created workflow. The WorkflowManagementService is layered on top of the ActiveBPEL workflow engine, which provides the primary functionality for running the BPEL-defined workflow. See Figure 8-1 for an overview of this architecture.

The following actions are performed when a user invokes start on the workflow management service:

- The input BPEL document is parsed and an exception is thrown if it is not well-formed (with respect to schema compliance).

- The input arguments to the workflow are declared as an array of xsd:any. They are parsed and cast to the types that they are meant to be.
- The service implementation invokes PDDGenerator to generate the deployment descriptor for the workflow.
- The service implementation invokes BPRGenerator to generate the BPelArchive (called bpr hereafter) that is ready to be deployed.
- The workflow management service is bootstrapped with the location of ActiveBPEL admin service location.
- The service implementation invokes deployBpr operation on the admin service, which is a vanilla Axis-based Web Service, to deploy the workflow created.
- The admin service responds with deployment summary reporting success if the workflow is deployed successfully.
- Once the workflow is deployed successfully, it is deployed as a web service inside ActiveBPEL.
- To start the workflow, a message is sent to the receiving partnerLink in the workflow.
- After the workflow successfully executes the results are returned to the client app/user by a call to getWorkflowOutput

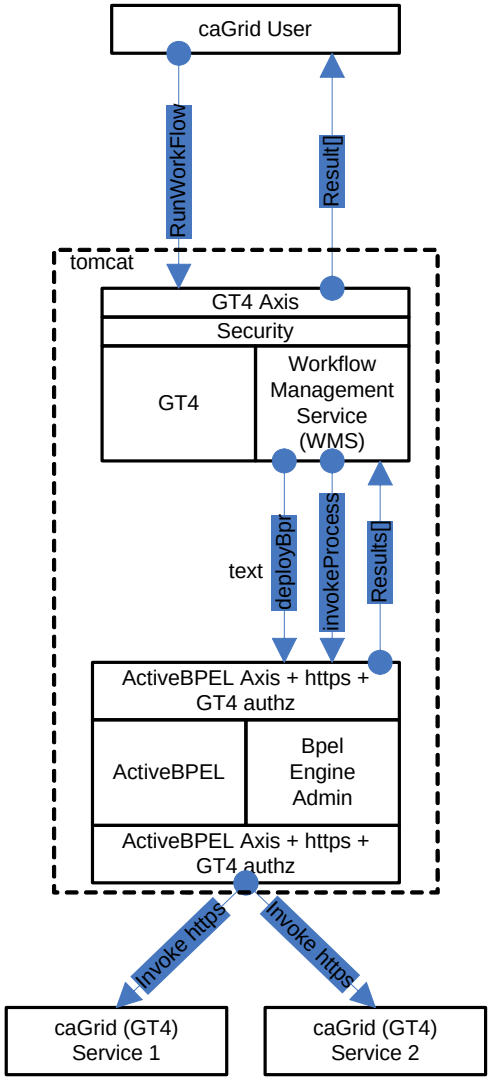


Figure 8-1 Overview of the architecture of the caGrid Workflow component

WorkflowFactoryService API

Workflows are created using the WorkflowFactoryService, which is a grid service that follows the resource pattern. The returned object holds an EPR to a WorkflowManagementService, which can be used to manipulate the create workflow.

Public WorkflowFactoryOutputType createWorkflow(WorkflowDescriptionType wmsInputType) throws WorkflowException ()

Description:

This method creates a workflow resource from the BPEL document found in wmsInputType and returns an EPR of the created resource to the client. The BPEL resource, along with the most recent state, is persisted in a MySQL database and is recovered in the event of a container

crash.

WorkflowDescriptionType:

This is the input to createWorkflow, and it consists of workflowName, a String bpelDoc, an Array of wsdlReferences, and an initial termination time for the workflow. If the termination time is not specified the service defaults to 24hrs. Termination of the workflow invalidates the WorkflowManagementService EPR and any running workflow is stopped.

```
<xsd:complexType name="WorkflowDescriptionType">
        <xsd:sequence>
      <xsd:element name="workflowName" type="xsd:string" minOccurs="1" maxOccurs="1"
/>
      <xsd:element name="bpelDoc" type="xsd:string" maxOccurs="1" />
      <xsd:element name="wsdlReferences" type="tns:WSDLReferences"
maxOccurs="unbounded" />
    <xsd:element name=”InitialTerminationTime” type=”xsd:dateTime”/>
     </xsd:sequence>
  </xsd:complexType>

<xsd:complexType name="WSDLReferences">
    <xsd:sequence>
        <xsd:element name="wsdlNamespace" type="xsd:anyURI"/>
        <xsd:element name="wsdlLocation" type="xsd:string"/>
        <xsd:element name="serviceUrl" type="xsd:anyURI"/>
    </xsd:sequence>
</xsd:complexType>
```

WorkflowFactoryOutputType:

This is the output of the createWorkflow method. An EPR is constructed by the factory and returned to the client. At this point the workflow document is deployed in the workflow engine and is also stored in a database, but it has not started. The EPR points to an instance of the WorkflowManagementService, which should be used to start the workflow.

```
<xsd:complexType name="WorkflowFactoryOutputType">
    <xsd:annotation>
      <xsd:documentation>This type represents the output from a
workflow</xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element name="WorkflowEPR" type="wsa:EndpointReferenceType" />
    </xsd:sequence>
  </xsd:complexType>
```

Faults:

UnableToDeployWorkflowFault: This fault is thrown if the workflow is unable to be deployed (e.g. the BPEL document submitted fails pre-deployment validation).

InvalidBPELFault extends UnableToDeployWorkflowFault: This fault is thrown if the BPEL document submitted fails pre-deployment validation (e.g. not valid XML).

Factory ResourceProperties:

We intend to provide aggregate resource properties on the factory service in our next iteration.

Following are some of the examples of those:

- Total number of workflows
- ListOfWorkflowsSubmitted

WorkflowManagementService API

This service is used to manage the workflow resources created by the WorkflowFactoryService (Figure 8-2). The service provides asynchronous execution of deployed workflows. The following are the operations the service provides in addition to the standard WS-RF operations such as destroy(), setTerminationTime(), etc.

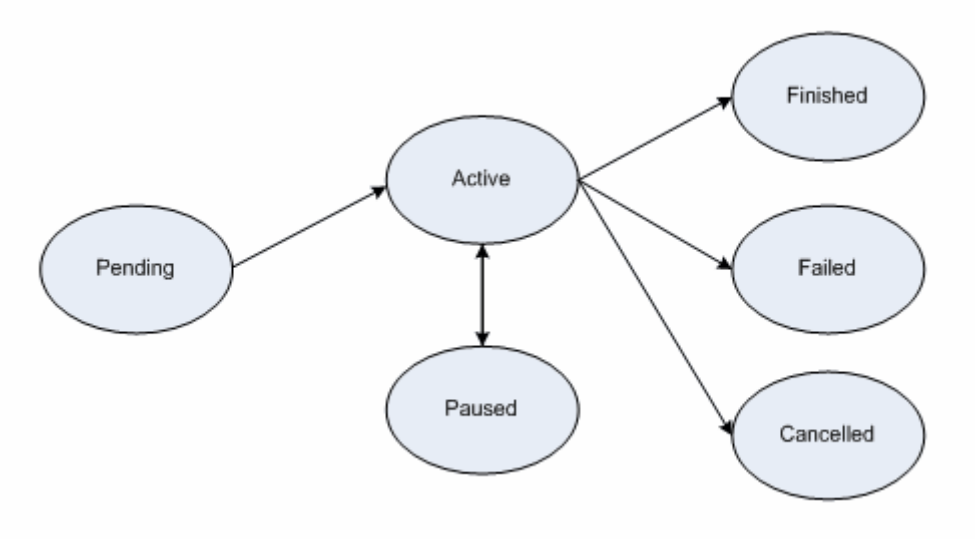


Figure 8-2 Workflow service state diagram

Public WorkflowStatusType start(StartInputType input) throws WorkflowException, StartCalledOnStartedWorkflowFault

Description:

This operation is used to start the workflow deployed using the factory with a set of input parameters. The input parameters are modeled as an array of xsd:any elements. The output is a void type.

```
<xsd:complexType name="StartInputType">
    <xsd:sequence>
      <xsd:element name="inputArgs" type="tns:WorkflowInputType" maxOccurs="1" />
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="WorkflowInputType">
    <xsd:sequence>
      <xsd:any maxOccurs="1" />
```

```
    </xsd:sequence>
  </xsd:complexType>
```

Faults:

StartCalledOnStartedWorkflowFault: This is thrown if start() is called on a workflow that is not in any one of the terminal states (i.e. Done, Failed, Cancelled).

WorkflowException: Every other fault results in the service throwing this with a message describing more details as to what went wrong.

Public WorkflowStatusType getStatus( ) throws WorkflowException

Description:

This operation is used to query for the status of the deployed workflow. WorkflowStatusType includes a fault.

```
<xsd:simpleType name="WorkflowStatusType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Pending" />
      <xsd:enumeration value="Active" />
      <xsd:enumeration value="Done" />
      <xsd:enumeration value="Failed" />
      <xsd:enumeration value="Cancelled" />
    </xsd:restriction>
  </xsd:simpleType>
```

Public WorkflowStatusType pause() throws CannotPauseFault

Description:

This operation pauses the workflow until resume() or cancel() is invoked. This operation translates to invoking an equivalent operation provided in the ActiveBPEL Admin interface. When the pause operation is invoked, ActiveBPEL stops the execution of the workflow document which means that there would no further service invocations or other activities. However, this will not affect the invocations in progress when the pause() is invoked. This operation returns the new state of the workflow resource (which always should be Active).

Public WorkflowStatusType resume() throws CannotResumeFault

Description:

This operation resumes a paused workflow. It translates to invoking an equivalent operation provided in the ActiveBPEL Admin interface.

Public WorkflowOutputType getWorkflowOutput() throws WorkflowException

Description:

This operation is used to get the final output of a completed workflow. It will return a fault if the

workflow is not yet completed. If ActiveBPEL allows for intermediate access of results, then this operation can potentially return the last result that the workflow engine has for this workflow. The output is modeled as a array of xsd:any elements.

```
<xsd:complexType name="WorkflowOutputType">
    <xsd:sequence>
      <xsd:any maxOccurs="1" />
    </xsd:sequence>
  </xsd:complexType>
```

Public void cancel() throws WorkflowException

Decription:

This operation terminates a workflow. It translates to invoking equivalent operation provided in the ActiveBPEL Admin interface.

Resource Properties:

WorkflowStatusRP:

The status of a workflow is exposed as a Resource Property so clients can subscribe to it and get notified when a state change happens. WorkflowStatusType is modeled as an Enum of Strings with the following valid values:

- Pending (Created but Start has not been called)
- Active
- Done
- Paused
- Failed

The status also includes the latest fault a workflow execution throws.

```
<xsd:simpleType name="WorkflowStatusType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Pending" />
      <xsd:enumeration value="Active" />
      <xsd:enumeration value="Done" />
      <xsd:enumeration value="Failed" />
      <xsd:enumeration value="Cancelled" />
      <xsd:enumeration value="Paused" />
    </xsd:restriction>
  </xsd:simpleType>
```

WorkflowStartTimeRP:

This property denotes the time when start() operation is called on the resource.

WorkflowEndTimeRP:

This property denotes the time when workflow status is set to Done/Failed/Cancelled.

Public void destroy()

Description:

This is a standard WS-RF operation but mentioned here to clarify the semantics and what it means to a Workflow Resource. If called, this method will delete a Workflow resource from the database along with the intermediate results and subscriptions for notifications. This operation is called by the GT4 framework when the lifetime of a resource is expired. The lifetime is set in the initial create() call in the factory. Internally, destroy() removes all the database entries for a particular workflow resource, all the subscriptions for notifications, and other temporary resources both in memory and on the disk.

Security in WorkflowFactory and Context Services

Two types of deployment patterns for Workflow are needed in regards to security. One deployment scenario would have the factory and the context service running with grid security (using Transport level security and caGrid authorization) and would require a client present grid credentials to submit and run workflows. Once a workflow resource is created by a user, programmatic GridMap authorization is used to limit access to the resource to the creator of the resource. Delegation of credentials is performed using the delegation service of the Globus Toolkit. This deployment is used to orchestrate workflows that require secure access to any services involved in the workflow. The other deployment does not have any security and is used to orchestrate workflow between unsecured grid services.

Service Selection

A Custom invoke handler is written for ActiveBPEL that queries a pre-configured GT4 index service to get the list of services. The query would be based on input and output types of the service invocation. Once a list of service handles is obtained from the index service, the dynamic endpoint for the service invocation is replaced by the first endpoint in the list.

Provenance Tracking

This is out of scope for this component during this release. No provenance tracking is exposed via the workflow component.

WS-RF Resources in Workflows

A BPEL service can involve affecting state of a WS-RF resource. Additional support needs to be added to ActiveBPEL to pass WS-A headers that contain EPRs and other relevant info to

caGrid services.

Chapter 9 caGrid Global Model Exchange

This chapter describes the caGrid Global Model Exchange.

Topics in this chapter include:

- Overview on this page
- GME Client on page 134

Overview

The caGrid Global Model Exchange (GME) is dependent on several software packages/systems that must be installed prior to installing and deploying the GME. The GME requires all of the required software packages of the caGrid core (Table 9-1).

Software	Version	Description
Java SDK	jsdk1.5 or higher	GME is written in Java therefore it requires the Java SDK. After installing you will have to set up an environmental variable pointing to the Java SDK directory and name it JAVA_HOME.
Mysql	Mysql 4.x or higher	For persistence and cache of models, GME uses the mysql database. Mysql can be downloaded from http://www.mysql.com/products/mysql/. The GME requires mysql version 4.X.
Ant	Ant 1.6.5	GME along with the Globus Toolkit in which GME is built on, uses Jakarta Ant for building and deploying.
Globus	Globus 4.0.3	GME is built on top of the Globus Toolkit. GME requires the ws-core installation of the Globus Toolkit.
Tomcat (Only required if deploying to Tomcat)	Tomcat 5.0.30	GME can be optionally deployed as a Grid Service to a Tomcat deployed Globus Toolkit.

Table 9-1 Software prerequisites for GME

Building the GME

GME is built when caGrid core is built. To build caGrid the following environment variables are required:

GLOBUS_LOCATION - The location of your globus 4.0.X installation.

CATALINA_HOME - The location of your Tomcat installation. (optional container)

If you have checked out caGrid Core from CVS, installed the required software packages, and

set the required environment variables, begin building caGrid Core by going into the caGrid Core checkout directory and entering *ant all.*

Configuring and Deploying the GME

Configuration

GME requires two configuration files for configuring the service. One configuration file is for configuring the Mobius GME and the other is for configuring the GME security preferences.

Each addresses a different GME set-up: connecting GME to other services, etc. The file *gme-globus-config.xml* contains a basic setup for a GME server to run on the local machine and provides an example of how the configuration files are structured. See the http://projectmobius.org/docs/mobiusconfig.php for information on the elements in the top resource block of the configuration files.

The `localhost config` files should never be used for more than single GME testing purposes. If you are running multiple GMEs, customize the config files by assigning unique service identifiers and making any other changes necessary to specify the service as unique. When other services connect to a GME started as localhost, the GME identifies itself as "localhost" to the connecting service. This can cause problems with service name resolution.

The GME server contains a single resource block that provides certain functionality and information to other GME components. The resources defined in the block are instantiated by the GME server at startup and are configured by the config file. The resource for the GME is: GME Configuration `<resource name="gmeConfig"... >`. Each configuration element is listed here, along with its children. Below this list, the purpose of the elements are described.

- *policies*
- *performance-caching*
- *notification-policy*
- *root-database*

`<policies>`

This element establishes parameters for how long the GME should keep old data and what other hosts it should notify of its existence.

`<performance-caching>`

The `<namespace-caching>` and `<schema-caching>` sub-elements of this element tell the GME how much namespace and schema data it should cache and for how long it should maintain the cache.

`<notification-policy>`

This element contains a `<notification-list>` sub-element that specifies which running GMEs it should notify upon instantiation.

`<root-database>`

This element configures the MySQL root database information. The children of this element configure the database. Its "id" attribute is the base name of the MySQL databases that will be created and used by the GME. If you configure multiple GMEs to

use a single MySQL installation, be sure to give a unique value to the "id" for each GME or there will be problems with name collision in MySQL.

<name>

This element specifies the name of the database. For MySQL's root database, this should be nothing.

<driver>

The driver class for accessing the MySQL database.

<urlPrefix>

The URL prefix for accessing the database.

<host>

The host the database lives on. Usually, this will be localhost.

<port>

The port from which the database can be accessed.

<username>

The username to log into the database with. This username must have privileges to create and delete databases and tables.

<password>

The password to authenticate the username.

<pool>

This element determines how many connections to the database will be made initially. When the GME needs to communicate with the database, it will get a connection, and when it's done, it releases it. Should the Database Manager run out of available connections, it will make a new one, but this causes a slight delay. Set the pool value in anticipation of how many concurrent database operations will be needed.

Deployment

The *GME_LOCATION/deploy.properties* file allows the configuration of deployment time properties of the service. The properties file contains two variables for configuring the service name and the service path. These variables are defaulted during skeleton creation time.

```
service.name=GlobalModelExchange

service.deployment.path=cagrid/GlobalModelExchange

service.deployment.host.default=localhost
```

Once GME is configured, it can be deployed to Globus running in Tomcat by entering *ant deployTomcat or ant deployGlobus* from the `GME_LOCATION` directory. Once deployed starting or restarting the container starts up the GME.

GME Backup and Restore

GME installations should run a backup script to make sure the integrity of the database can be restored upon any failures. A general purpose script for this is provided in the tools directory. There is a script for backup and one for restore. Each script has a short description inside describing the usage of the script and what variables might need to be configured. This script can be executed from a crontab and maintains five rolling backup caches of the GME databases.

Important Notes

The default GME configuration is set to connect to a MySQL database on the localhost with no password and username root. If you need to change this be sure to edit the `gme config` file in the etc directory. The GME also dynamically creates its databases and tables. Make sure that the database privileges are set correctly to enable this.

GME Client

GME Client API

The GME client API is a simple java based API that enables simple access to remote GMEs in order to publish, retrieve, and discover models as well as manage GMEs. The client API comes from the Mobius project and is included in the caGrid release. The communication factories required to use the GME in the caGrid environment are built into the caGrid release jar and deployed into the skeleton. Below is a code example of how to get a handle to a GME and request a schema, and all it referenced schemas, to be retrieved and written to a place on the file system.

```
GridServiceResolver.getInstance().setDefaultFactory(new GlobusGMEXMLDataModelServiceFactory());
List writtenNamespaces = null;
File directory = new File(CACHE_LOCATION);
try {
        XMLDataModelService handle = (XMLDataModelService) GridServiceResolver.getInstance()
        .getGridService("http://dc04.bmi.ohio-state.edu:8080/ogsa/services/cagrid/gme");
    writtenNamespaces = handle.cacheSchema(cagrid.nci.nih.gov/1/Gene,directory);
} catch (MobiusException e1) {
    e1.printStackTrace();
}
```

For more information on other methods the GME API provides, refer to http://projectmobius.org/docs/gmeapi.php or browse the Mobius GME source code, which is

freely available at www.projectmobius.org.

GME Viewer

To launch the GME Viewer, run *ant gmeViewer* from the `GME_LOCATION`. This launches the Mobius GME Viewer GUI configured to use Globus for communication. Once this tool is launched, follow the Mobius GUI Documentation for using the GUI (http://projectmobius.org/docs/gmeqs.php).

Appendix A References

Scientific Publications

[1] B. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnal, and T. S., "Data Management and Transfer in High Performance Computational Grid Environments," *Parallel Computing Journal*, vol. 28, pp. 749-771, 2002.

[2] W. E. Allcock, I. Foster, and R. Madduri, "Reliable Data Transport: A Critical Service for the Grid.," in *Proceedings of Building Service Based Grids Workshop, Global Grid Forum 11*. Honolulu, Hawaii, USA, 2004.

[3] G. Allen, T. Dramlitsch, I. Foster, T. Goodale, N. Karonis, M. Ripeanu, E. Seidel, and B. Toonen, "Cactus-G Toolkit: Supporting Efficient Execution in Heterogeneous Distributed Computing Environments," in *Proceedings of the 4th Globus Retreat*. Pittsburg, PA, 2000.

[4] H. Andrade, T. Kurc, A. Sussman, and J. Saltz, "Active Proxy-G: Optimizing the Query Execution Process in the Grid," in *Proceedings of the ACM/IEEE Supercomputing Conference (SC2002)*. Baltimore, MD: ACM Press/IEEE Computer Society Press, 2002.

[5] J. Annis, Y. Zhao, J. Voeckler, M. Wilde, S. Kent, and I. Foster, "Applying Chimera Virtual Data Concepts to Cluster Finding in the Sloan Sky Survey," in *Proceedings of the ACM/IEEE Supercomputing Conference (SC2002)*. Baltimore, MD: ACM Press/IEEE Computer Society Press, 2002.

[6] M. P. Atkinson and et.al., "Grid Database Access and Integration: Requirements and Functionalities," Technical Document, Global Grid Forum. http://www.cs.man.ac.uk/grid-db/documents.html, 2002.

[7] F. Berman, H. Casanova, J. Dongarra, I. Foster, C. Kesselman, J. Saltz, and R. Wolski, "Retooling Middleware for Grid Computing," *NPACI & SDSC enVision*, vol. 18, 2002.

[8] M. Beynon, T. Kurc, A. Sussman, and J. Saltz, "Design of a Framework for Data-Intensive Wide-Area Applications," in *Proceedings of the 2000 Heterogeneous Computing Workshop (HCW2000)*. Cancun, Mexico, 2000.

[9] H. Casanova, O. Graziano, F. Berman, and R. Wolski, "The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid," in *Proceedings of the ACM/IEEE Supercomputing Conference (SC2000)*: ACM Press/IEEE Computer Society Press, 2000.

[10] A. Chervenak, E. Deelman, I. Foster, L. Guy, W. Hoschek, A. Iamnitchi, C. Kesselman, P. Kunst, M. Ripeanu, B. Schwartzkopf, H. Stockinger, and B. Tierney, "Giggle: A Framework for Constructing Scalable Replica Location Services," in *Proceedings of the ACM/IEEE Supercomputing Conference (SC2002)*: ACM Press/IEEE Computer

Computer Society Press, 2002, pp. 1-17.

[11] A. Chervenak, E. Deelman, C. Kesselman, B. Allcock, I. Foster, V. Nefedova, J. Lee, A. Sim, A. Shoshahi, B. Drach, D. Williams, and D. Middleton, "High-performance remote access to climate simulation data: a challenge problem for data grid technologies," *Parallel Computing*, vol. 29, pp. 1335-1356, 2003.

[12] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke, "The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets," *Journal of Network and Computer Applications*, vol. 23, pp. 187-200, 2000.

[13] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh, and S. Koranda, "Mapping Abstract Complex Workflows onto Grid Environments," *Journal of Grid Computing*, vol. 1, pp. 25-39, 2003.

[14] E. Deelman, G. Singh, M. P. Atkinson, A. Chervenak, N. P. Chue Hong, C. Kesselman, S. Patil, L. Pearlman, and M. Su, "Grid-Based Metadata Services," in *Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM '04)*, 2004.

[15] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit.," *International Journal of High Performance Computing Applications*, vol. 11, pp. 115--128, 1997.

[16] I. Foster, J. Voeckler, M. Wilde, and Y. Zhao, "Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation," in *Proceedings of the 14th Conference on Scientific and Statistical Database Management (SSDBM '02)*, 2002.

[17] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke, "Condor-G: A Computational Management Agent for Multi-institutional Grids," in *Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10)*: IEEE Press, 2001.

[18] N. Furmento, W. Lee, A. Mayer, S. Newhouse, and J. Darlington, "ICENI: An Open Grid Service Architecture Implemented with JINI," in *Proceedings of the ACM/IEEE Supercomputing Conference (SC2002)*. Baltimore, MD: ACM Press/IEEE Computer Society Press, 2002.

[19] A. S. Grimshaw and W. Wulf, "The Legion: Vision of a Worldwide Virtual Computer," *Communications of the ACM*, vol. 40, pp. 39--45, 1997.

[20] S. Hastings, S. Langella, S. Oster, and J. Saltz, "Distributed Data Management and Integration: The Mobius Project," *Proceedings of the Global Grid Forum 11 (GGF11) Semantic Grid Applications Workshop, Honolulu, Hawaii, USA.*, pp. 20-38, 2004.

[21] S. Langella, S. Oster, S. Hastings, F. Siebenlist, T. Kurc, and J. Saltz, "Dorian: Grid Service Infrastructure for Identity Management and Federation," presented at The 19th IEEE Symposium on Computer-Based Medical Systems, Special Track: Grids for Biomedical Informatics, Salt Lake City, Utah., 2006.

[22] R. Oldfield and D. Kotz, "Armada: A Parallel File System for Computational Grid," in *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid*

(CCGrid2001). Brisbane, Australia: IEEE Computer Society Press, 2001.

[23] M. Sato, H. Nakada, S. Sekiguchi, S. Matsuoka, U. Nagashima, and H. Takagi, "Ninf: A Network based Information Library for a Global World-Wide Computing Infrastructure," in *Proceedings of the Conference on High Performance Computing and Networking (HPCN '97) (LNCS-1225)*, 1997, pp. 491-502.

[24] G. Singh, S. Bharathi, A. Chervenak, E. Deelman, C. Kesselman, M. Mahohar, S. Pail, and L. Pearlman, "A Metadata Catalog Service for Data Intensive Applications," in *Proceedings of the ACM/IEEE Supercomputing Conference (SC2003)*, 2003.

[25] G. Singh, E. Deelman, G. Mehta, K. Vahi, M. Su, B. Berriman, J. Good, J. Jacob, D. Katz, A. Lazzarini, K. Blackburn, and S. Koranda, "The Pegasus Portal: Web Based Grid Computing," in *Proceedings of the 20th Annual ACM Symposium on Applied Computing*. Santa Fe, New Mexico, 2005.

[26] J. Smith, A. Gounaris, P. Watson, N. W. Paton, A. A. Fernandes, and R. Sakellariou, "Distributed Query Processing on the Grid.," presented at Proceedings of the Third Workshop on Grid Computing (GRID2002), Baltimore, MD, 2003.

[27] D. Thain, J. Basney, S. Son, and M. Livny, "Kangaroo Approach to Data Movement on the Grid," in *Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC-10)*, 2001.

[28] L. Weng, G. Agrawal, U. Catalyurek, T. Kurc, S. Narayanan, and J. Saltz, "An Approach for Automatic Data Virtualization," in *Proceedings of the 13th IEEE International Symposium on High-Performance Distributed Computing (HPDC-13)*. Honolulu, Hawaii, 2004, pp. 24-33.

[29] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," Open Grid Service Infrastructure Working Group Technical Report, Global Grid Forum. http://www.globus.org/alliance/publications/papers/ogsa.pdf 2002.

[30] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations.," *International Journal of Supercomputer Applications*, vol. 15, pp. 200-222, 2001.

[31] E. Cerami, *Web Services Essentials*: O'Reilly & Associates Inc., 2002.

[32] S. Graham, S. Simeonov, T. Boubez, D. Davis, G. Daniels, Y. Nakamura, and R. Neyama, *Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI*: SAMS Publishing, 2002.

[33] K. Czajkowski, D. F. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke, and W. Vambenepe, "The WS-Resource Framework version 1.0," vol. 2004, 2004.

[34] J. Saltz, S. Oster, S. Hastings, T. Kurc, W. Sanchez, M. Kher, A. Manisundaram, K. Shanbhag, and P. Covitz, "caGrid: Design and Implementation of the Core Architecture of the Cancer Biomedical Informatics Grid," *Bioinformatics. (in press).* 2006.

[35] S. Langella, S. Hastings, S. Oster, T. Kurc, U. Catalyurek, and J. Saltz, "A Distributed Data Management Middleware for Data-Driven Application Systems," in *Proceedings of the 2004 IEEE International Conference on Cluster Computing (Cluster 2004)*, 2004.

[36] K. Bhatia, S. Chandra, and K. Mueller, "GAMA: Grid Account Management Architecture," San Diego Supercomputer Center (SDSC), UCSD Technical Report. #TR-2005-3, 2005.

[37] I. Foster, C. Kesselman, S. Tuecke, V. Volmer, V. Welch, R. Butler, and D. Engert, "A National Scale Authentication Infrastructure," *IEEE Computer*, vol. 33, pp. 60-66, 2000.

[38] V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, and S. Tuecke, "Security for Grid Services," presented at 12th International Symposium on High Performance Distributed Computing (HPDC-12), 2003.

[39] H. Morohoshi and R. Huang, "A User-friendly Platform for Developing Grid Services over Globus Toolkit 3," presented at The 2005 11th International Conference on Parallel and Distributed Systems (ICPADS'05), 2005.

[40] S. Mizuta and R. Huang, "Automation of Grid Service Code Generation with AndroMDA for GT3," presented at The 19th International Conference on Advanced Information Networking and Applications (AINA'05), 2005.

[41] G. von Laszewski, I. Foster, J. Gawor, and P. Lane, "A Java Commodity Grid Kit," *Concurrency and Computation: Practice and Experience*, vol. 13, pp. 643-662, 2001.

[42] G. von Laszewski, I. Foster, J. Gawor, W. Smith, and S. Tuecke, "CoG Kits: A Bridge Between Commodity Distributed Computing and High Performance Grids," presented at ACM Java Grande 2000 Conference, 2000.

[43] R. Buyya and S. Venugopal, "The Gridbus Toolkit for Service Oriented Grid and Utility Computing: An Overview and Status Report," presented at the First IEEE International Workshop on Grid Economics and Business Models (GECON 2004), New Jersey, USA, 2004.

[44] M. Humphrey and G. Wasson, "Architectural Foundations of WSRF.NET," *International Journal of Web Services Research*, vol. 2, pp. 83-97, 2005.

[45] M. Smith, T. Friese, and B. Freisleben, "Model Driven Development of Service Oriented Grid Applications," presented at Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT-ICIW '06), 2006.

Technical Manuals/Articles

National Cancer Institute. "caCORE 3.1 Technical Guide", ftp://ftp1.nci.nih.gov/pub/cacore/caCORE3.1_Tech_Guide.pdf

Java Bean Specification: http://java.sun.com/products/javabeans/docs/spec.html

Foundations of Object-Relational Mapping: http://www.chimu.com/publications/objectRelational/

Object-Relational Mapping articles and products:

http://www.service-architecture.com/object-relational-mapping/

Hibernate Reference Documentation: http://www.hibernate.org/hib_docs/reference/en/html/

Basic O/R Mapping: http://www.hibernate.org/hib_docs/reference/en/html/mapping.html

Java Programming: http://java.sun.com/learning/new2java/index.html

Javadoc tool: http://java.sun.com/j2se/javadoc/

JUnit: http://junit.sourceforge.net/

Extensible Markup Language: http://www.w3.org/TR/REC-xml/

XML Metadata Interchange: http://www.omg.org/technology/documents/formal/xmi.htm

Global Grid Forum: http://www.gridforum.org

Globus: http://www.globus.org

Mobius: http://www.projectmobius.org

W3C: http://www.w3c.org

OGSA-DAI: http://www.ogsadai.org

Apache: http://www.apache.org

Globus Toolkit 3 Programmer's Tutorial:
http://gdp.globus.org/gt3-tutorial/singlehtml/progtutorial_0.4.3.html

XPath tutorial: http://www.w3schools.com/xpath/xpath_syntax.asp

Globus Security Overview:
http://www.ogsadai.org.uk/docs/OtherDocs/SECURITY-FOR-DUMMIES.pdf

High level Overview of Grid:
http://gridcafe.web.cern.ch/gridcafe/index.html

Overview of Globus Toolkit 3 and the OGSI architecture :
http://www-128.ibm.com/developerworks/grid/library/gr-gt3/

caBIG Material

caBIG: http://cabig.nci.nih.gov/

caBIG Compatibility Guidelines: http://cabig.nci.nih.gov/guidelines_documentation

caCORE Material

caCORE: http://ncicb.nci.nih.gov/NCICB/infrastructure

caBIO: http://ncicb.nci.nih.gov/NCICB/infrastructure/cacore_overview/caBIO

caDSR: http://ncicb.nci.nih.gov/NCICB/infrastructure/cacore_overview/cadsr

EVS: http://ncicb.nci.nih.gov/NCICB/infrastructure/cacore_overview/vocabulary

CSM: http://ncicb.nci.nih.gov/NCICB/infrastructure/cacore_overview/csm

Glossary

Term	Definition
{jboss-home}	The base directory where JBoss is installed on the server
API	Application Programming Interface
caArray	cancer Array Informatics
caBIG	cancer Biomedical Informatics Grid
caBIO	Cancer Bioinformatics Infrastructure Objects
caCORE	cancer Common Ontologic Representation Environment
caDSR	Cancer Data Standards Repository
caMOD	Cancer Models Database
cardinality	Cardinality describes the minimum and maximum number of associated objects within a set
CDE	Common Data Element
CGAP	Cancer Genome Anatomy Project
CMAP	Cancer Molecular Analysis Project
CN	Common Name
CS	Classification Scheme
CSI	Classification Scheme Item
CSM	Common Security Module
CTEP	Cancer Therapy Evaluation Program
CUI	Concept Unique Identifier
CVS	Concurrent Versions System
DAIS	Data Access and Integration Services
DAML	DARPA Agent Markup Language
DAO	Data Access Objects
DARPA	Defense Advanced Research Projects Agency
DAS	Distributed Annotation System
DL	Description Logic
EA	Enterprise Architect
EBI	European Bioinformatics Institute
EVS	Enterprise Vocabulary Services
GAI	CGAP Genetic Annotation Initiative

Term	Definition
GEDP	Gene Expression Data Portal
GGF	Global Grid Forum
GME	Mobius Global Model Exchange - DNS-like service for the universal creation, versioning, and sharing of data descriptions
Grid Service	Basically a Web Services with improved characteristics and standard services like stateful and potentially transient services, Service Data, Notifications, Service Groups, portType extension, and Lifecycle management.
GSH	Grid Service Handle
GSI	Grid Security Infrastructure - represents the latest evolution of the Grid Security Infrastructure. GSI in GT3 builds off of the functionality present in early GT2 toolkit releases - X.509 certificates, TLS/SSL for authentication and message protection, X.509 Proxy Certificates for delegation and single sign-on.
HTTP	Hypertext Transfer Protocol
ISO	International Organization for Standardization
JAAS	Java Authentication and Authorization Service
JAR	Java Archive
Javadoc	Tool for generating API documentation in HTML format from doc comments in source code (http://java.sun.com/j2se/javadoc/)
JDBC	Java Database Connectivity
JET	Java Emitter Templates
JMI	Java Metadata Interface
JSP	JavaServer Pages
JUnit	A simple framework to write repeatable tests (http://junit.sourceforge.net/)
LDAP	Lightweight Directory Access Protocol
LLT	Lowest Level Term
LOINC	Logical Observation Identifier Names and Codes
MAGE	MicroArray and Gene Expression
MAGE-OM	MicroArray Gene Expression - Object Model
MDA	Model Driven Architecture
MedDRA	Medical Dictionary for Regulatory Activities
metadata	Definitional data that provides information about or documentation of other data.
MGED	Microarray Gene Expression Data

Term	***Definition***
Mobius	An array of tools and middleware components to coherently share and manage data and metadata in a Grid and/or distributed computing environment.
multiplicity	Multiplicity of an association end indicates the number of objects of the class on that end may be associated with a single object of the class on the other end
NCI	National Cancer Institute
NCICB	National Cancer Institute Center for Bioinformatics
OGSA	Open Grid Services Architecture - developed by the Global Grid Forum, aims to define a common, standard, and open architecture for grid-based applications.
OGSI	Open Grid Services Infrastructure -gives a formal and technical specification of what a Grid Service is. In other words, for a high-level architectural view of what Grid Services are, and how they fit into the next generation of grid applications
OIL	Ontology Inference Layer
OilEd	Ontology editor allowing you to build ontologies using DAML+OIL
OLLT	Obsolete Lower Level Terms
OMG	Object Management Group
ORM	Object Relational Mapping
PT	Preferred Term
RDBMS	Relational Database Management System
SDE	Service Data Element
SDK	Software Development Kit
Semantic connector	A development kit to link model elements to NCICB EVS concepts.
SOA	Service Oriented Architecture: A discipline for building reliable distributed systems that deliver application functionality as services with the additional emphasis on loose coupling between interacting services.
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SOC	System Organ Class
SPORE	Specialized Programs of Research
SQL	Structured Query Language

Term	***Definition***
SSC	Special Search Categories
UI	User Interface
UID	User Identification
UML	Unified Modeling Language
UML	Unified Modeling Language
UMLS	Unified Medical Language System
UPT	User Provisioning Tool
URL	Uniform Resource Locators
VD	Value Domain
Virtualization	Make a computational or data resource available to caBIG community - some people call "Gridification"
VO	Virtual Organization
WAR	Web Application Archive
Web Service	Application to application communication using web based service interfaces as describe by the Web Services 1.0 or 2.0 specification.
WSDD	Web Service Deployment Descriptor
WSDL	Web Services Description Language
WSDL	Web Services Description Language
WSRF	Web Services Resource Framework
XMI	XML Metadata Interchange (http://www.omg.org/technology/documents/formal/xmi.htm) - The main purpose of XMI is to enable easy interchange of metadata between modeling tools (based on the OMG-UML) and metadata repositories (OMG-MOF) in distributed heterogeneous environments
XML	Extensible Markup Language (http://www.w3.org/TR/REC-xml/) - XML is a subset of Standard Generalized Markup Language (SGML). Its goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML
XML	Extensible Markup Language
XPath	XML query/traversal language adhering to the XPath specification set forth by the W3C.
XQuery	XML query/transformation language adhering to the XQuery specification set forth by the W3C.

Index

addNonNullPredicate
  gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 39
annotateServiceMetadata, 46
API
  caDSR grid service usage, 45
  caDSR Grid Service usage examples, 53
  caDSR Grid Services details, 49
  Discovery API details, 26
  Discovery usage examples, 41
  Discovery usage overview, 24
  EVS usage overview, 57
  federated query processor details, 88
  federated query processor examples, 92
  GME client, 134
  metadata, 15
  metadata details, 18
  metadata usage examples, 21
  WS-enumeration client, 109
  WS-enumeration usage examples, 112
BPEL, 121
buildDataUMLClassPredicate
  gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 38
buildPOCPredicate
  gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 38
buildUMLClassPredicate
  gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 38
caArray
  exposure to Grid, 102
caBIG. *See* caGrid
  references, 141
caBioconductor, 102
caCORE
  references, 141
caDSR Grid Service
  overview, 45
  security considerations, 49
caDSR Grid Services
  API details, 49
  API usage examples, 53
  examining project information model, 53
  generating data service metadata, 54
caDSRServiceClient, 49
caGrid
  infrastructure description, 8
  overview, 5
  reference implementation, 99
  reference implementation test bed, 101
  security infrastructure, 10
  security overview, 65
caGrid document, 2
castor mapping, 78
caTRIP, 103
client, 75
CQL syntax validation, 81
`CQLQueryResults`, 76
  creating, 78
createConceptPredicatedUMLClass
  gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 34
createPermissibleValuePredicatedUMLClass
  gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 34
Data Services
  CQL query processors, 83
  CQL query results, 76
  CQL query syntax, 81
  domain model conformance, 82
  federated query processor overview, 83
  overview, 75
  results validation, 82
  utility classes, 78
`DataServiceConstants`, 76
deserializeDomainModel, 23
  gov::nih::nci::cagrid::metadata::MetadataUtils, 20
deserializeServiceMetadata, 23
  gov::nih::nci::cagrid::metadata::MetadataUtils, 20
discoverByFilter
  gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 39
discoverDataServicesByAssociationsWithClass
  gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 37
discoverDataServicesByDomainModel, 43
  gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 35
discoverDataServicesByExposedClass
  gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 36
discoverDataServicesByModelConceptCode
  gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 35
discoverDataServicesByPermissibleValue
  gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 36
discoverServiceByOperationInput, 45
discoverServicesByConceptCode, 44

gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 29
discoverServicesByDataConceptCode
gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 33
discoverServicesByName
gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 29
discoverServicesByOperationClass
gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 31
discoverServicesByOperationConceptCode, 44
gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 32
discoverServicesByOperationInput
gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 30
discoverServicesByOperationName, 43
gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 30
discoverServicesByOperationOutput
gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 31
discoverServicesByPermissibleValue
gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 33
discoverServicesByPointOfContact
gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 28
discoverServicesByResearchCenter, 43
gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 28
discoverServicesBySearchString, 43
gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 27
Discovery API
configuring index service, 41
details, 26
discovering services, 42
usage examples, 41
DiscoveryClient
definition, 24
gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 26
Disovery API
overview, 24
Document conventions, technical guide, 2
domain model, 79
Domain Model, 82
`enumerationQuery()` method, 76
EVS API
getHistoryRecords, 60
getMetaSources, 57
getVocabularyNames, 58
searchDescLogicConcept, 58
searchMetaThesaurus, 61
searchSourceByCode, 62
usage overview, 57
execute
gov::nih::nci::cagrid::fqp::processor::FederatedQueryEngine, 91
executeAndAggregateResults
gov::nih::nci::cagrid::fqp::processor::FederatedQueryEngine, 92
FederatedQueryEngine
gov::nih::nci::cagrid::fqp::processor::FederatedQueryEngine, 91
FileReader, 23
FileWriter, 23
GAARDS, 10
GeneConnect, 105
GenePattern, 104
generateDomainModelForClasses, 46
generateDomainModelForClassesWithExcludes, 46
generateDomainModelForPackage, 46
generateDomainModelForProject, 46, 55
getAllDataServices
gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 34
getAllServices
gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 26
getDomainModel
gov::nih::nci::cagrid::metadata::MetadataUtils, 19
getIndexEPR
gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 40
getServiceMetadata
gov::nih::nci::cagrid::metadata::MetadataUtils, 18
Getting help, 1
geWorkbench, 106
Global Model Exchange. *See* GME
GME
building, 131
client API, 134
configuring, 131
deploying, 133
overview, 7
software prerequisites, 131
viewer, 135
GME XE "Global Model Exchange" \t "*See* GME" XE "GME:software prerequisites"
overview, 131
gov::nih::nci::cagrid::discovery::client::DiscoveryClient
addNonNullPredicate, 39
buildDataUMLClassPredicate, 38
buildPOCPredicate, 38
buildUMLClassPredicate, 38

createConceptPredicatedUMLClass, 34
createPermissibleValuePredicatedUMLClass, 34
discoverByFilter, 39
discoverDataServicesByAssociationsWithClass, 37
discoverDataServicesByDomainModel, 35
discoverDataServicesByExposedClass, 36
discoverDataServicesByModelConceptCode, 35
discoverDataServicesByPermissibleValue, 36
discoverServicesByConceptCode, 29
discoverServicesByDataConceptCode, 33
discoverServicesByName, 29
discoverServicesByOperationClass, 31
discoverServicesByOperationConceptCode, 32
discoverServicesByOperationInput, 30
discoverServicesByOperationName, 30
discoverServicesByOperationOutput, 31
discoverServicesByPermissibleValue, 33
discoverServicesByPointOfContact, 28
discoverServicesByResearchCenter, 28
discoverServicesBySearchString, 27
DiscoveryClient, 26
getAllDataServices, 34
getAllServices, 26
getIndexEPR, 40
main, 40
setIndexEPR, 40
translateXPath, 39
gov::nih::nci::cagrid::discovery::XPathUtils
translateXPath, 41
gov::nih::nci::cagrid::fqp::processor::FederatedQueryEngine
execute, 91
executeAndAggregateResults, 92
FederatedQueryEngine, 91
gov::nih::nci::cagrid::metadata::MetadataUtils
deserializeDomainModel, 20
deserializeServiceMetadata, 20
getDomainModel, 19
getServiceMetadata, 18
serializeDomainModel, 20
serializeServiceMetadata, 19
Grid Grouper
overview, 69
GridIMAGE, 107
InternalRuntimeException, 18, 24
InvalidProjectException, 46
InvalidResourcePropertyException, 18, 25
ISO/IEC 11179, 6
main
gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 40
Malformed query, 84
Metadata
accessing from service, 21
API details, 18
API usage examples, 21
API usage overview, 15
processing as XML, 22
PointOfContact, 44
`processQuery()` method, 84
properties, 83
optional, 83
QName, 76
`query()` method, 75
QueryInvalidException, 18, 25
Reader, 23
Reference Implementation
overview, 99
References
caBIG, 141
caBIG materials, 141
caCORE, 141
caCORE material, 141
scientific publications, 137
technical manuals, guides, 140
RemoteResourcePropertyRetrievalException, 18, 25
ResourcePropertyHelper, 17
ResourcePropertyRetrievalException, 18, 24, 25
serializeDomainModel, 23
gov::nih::nci::cagrid::metadata::MetadataUtils, 20
serializeServiceMetadata, 23
gov::nih::nci::cagrid::metadata::MetadataUtils, 19
ServiceMetadata, 23, 46
setIndexEPR
gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 40
sourceClassName, 46
sourceRoleName, 46
target data types, 82
targetClassName, 46
targetRoleName, 46
Text conventions, technical guide, 2
thread, 76
translateXPath
gov::nih::nci::cagrid::discovery::client::DiscoveryClient, 39
gov::nih::nci::cagrid::discovery::XPathUtils, 41
UMLAssocation, 48
UMLAssociationExclude, 48
UMLAssociationExcludes, 46
UMLAssociationMetadata, 48

UMLClass, 44
URL, 76
Workflow
  architecture description, 121
  overview, 121
  security contexts, 128
  WorkflowFactoryService API, 123
  WorkflowManagementService API, 125
Writer, 23
wsdd parameters, 81
WS-Enumeration
  client API, 109
  command line clients, 115
  overview, 109
WS-Enumeration feature, 76