caBIG™ cancer Biomedical Informatics Grid™

caGrid 1.1 Data Services

Technical Guide

Updated July 25,

caBIG™ is an initiative of the National Cancer Institute, NIH, DHHS

Copyright and License page

Insert date here (optional)

Insert Names of Development Team(s) here		
Development	***Programmer's guide***	***Program Management***
Names of developers	Names of technical writers and reviewers	Names of program managers
David W. Ervin	David W. Ervin	
[1] Footnote contributing companies/ contractors/ divisions		[3] company
[2] department or division		

Contacts and Support	
Enter training contact	name (email address)
Enter customer support contact	Web hyperlink Telephone: 111-222-3456 Toll free: 888-234-1234

LISTSERV Facilities Pertinent to software teams		
LISTSERV	***URL***	***Name***
caGrid_Developers	https://list.nih.gov/archives/cagrid_developers.html	caGrid Developers Discussion

Insert date here (optional)

Table of Contents

List of Figures

Chapter 1 Introduction to the Technical Guide

Overview of the Guide

Brief overview of this developer's guide.

Document Text Conventions

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

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 methods, variables, and classes.	`URL_definition ::= url_string`
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 other documents, sections, figures, and tables.	*caCORE Software Development Kit 1.0 Programmer's Guide*
Italic boldface type	Text that you type	In the New Subset text box,

Convention	Description	Example
		enter ***Proprietary Proteins.***
Note:	Highlights a concept of particular interest	**Note:** This concept is used throughout the installation manual.
Warning!	Highlights information of which you should be particularly aware.	**Warning!** Deleting an object will permanently delete it from the database.
{}	Curly brackets are used for replaceable items.	Replace {`root directory}` with its proper value such as `c:\cabio`

Table 1 Document Conventions

Chapter 2 caGrid Data Services Overview

Introduction

caGrid Data Services provide an object view of a data resource across the grid. The data resource is exposed through a well defined query method, which also relies on well defined query language objects to perform queries and return results as a strongly typed set. caGrid Data Services are designed to expose objects whose XML schemas are registered in the GME, and also expose metadata about those data objects derived from the caDSR.

The caGrid Data Services infrastructure is composed of many individual parts, which work in conjunction to streamline the processes of service creation, configuration, and querying. The design is such that many components are pluggable, allowing new capabilities and functionality to be added to Data Services without requiring sweeping changes throughout the large system that composes the Data Services infrastructure.

Data Services also provide support for integration with alternate results delivery mechanisms such as WS-Enumeration and caGrid's Bulk Data Transport framework. Enabling these features adds new query methods to the Grid-facing API.

Chapter 3 Service Side Components

Introduction

This chapter describes the various components, shown below in Figure 1, that make up the business logic a caGrid Data Service. Some components are reusable outside of the data services context (e.g. CQL Validation Utilities).

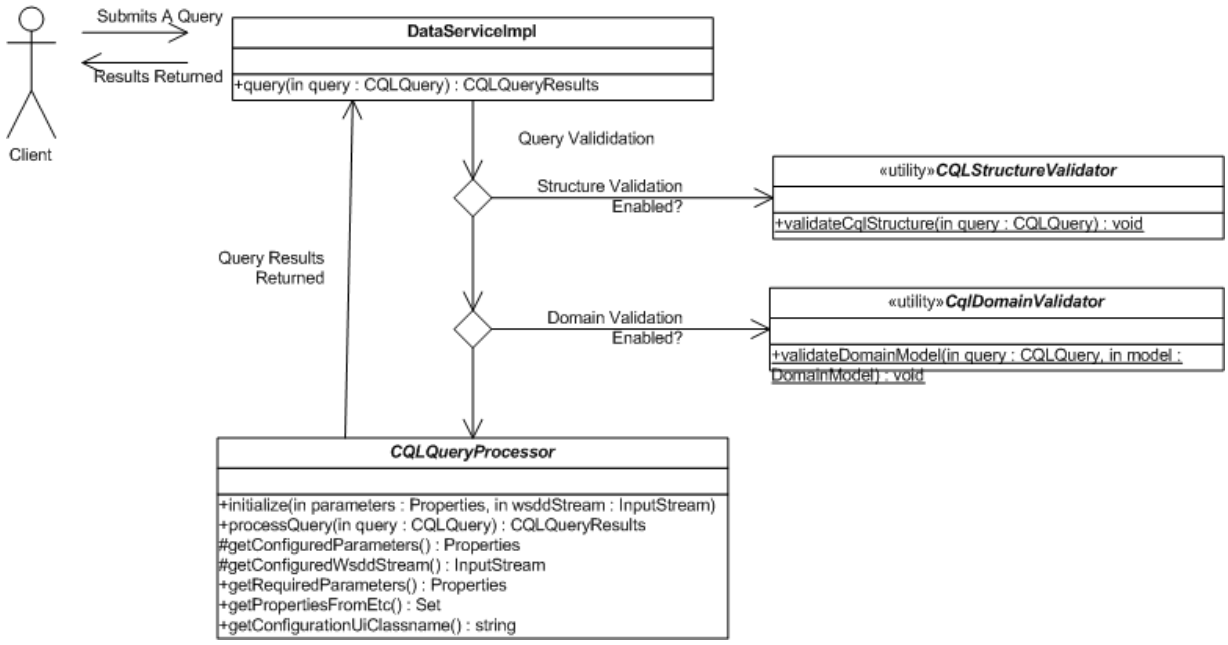

Figure 1 Data Services service side components

Service Side Component Details

Data Service Impl

The Data Services core server implementation resides in the *DataServiceImpl* class. This class has a single public facing API method for querying the data service (*query(CQLQuery)*). Two additional implementations exist, which provide support for integration with WS-Enumeration and the caGrid Bulk Data Transfer infrastructure. Bulk Data Transfer integration is achieved by using a template to write custom code into the BDT generated resource class. This generated code in turn calls into a resource helper which maintains the result set and WS-Enumeration implementation.

Invoking the query method of a data service uses values stored in the service properties to determine if validation of the query should be performed, and if so, to what extent.

Query Validation

A CQL query instance may be validated for both structure and conformance to the service's exposed Domain Model. If structure validation is enabled, the query will pass through the CQL Structure Validator first. This utility ensures that the query conforms to the CQL schema. If domain validation is enabled, the query is parsed to verify that at each step of the query, only objects exposed in the domain model and valid attributes and associations of each are used. Failing either of these validation steps will raise a *MalformedQueryException*, which contains information pertaining to the cause of the failure, and no further processing takes place.

CQL Query Processor

The CQL Query Processor is a pluggable implementation which handles the details of processing CQL against some backend data source and producing a CQLQueryResults instance. The particular implementation used is determined by a value in the service's deployment properties, and an instance of the processor is loaded at runtime via reflection. The query processor may optionally supply a set of properties via the *getRequiredParameters().* These properties may be configured prior to deployment of the service, and are passed in to the query processor when it is first instantiated via the *initialize()* method.

When a query is issued to the data service, the query will be passed along to the CQL Query Processor instance's *processQuery()* method. This method may throw both a *QueryProcessingException* in the case of an error in handling the query and a *MalformedQueryException* in cases where the query was found to be invalid for any reason.

Chapter 4 Client Side API and Utilities

Introduction

This chapter describes the software components which make up the client API and client side utilities of the caGrid data services infrastructure. These tools are designed to enable application developers to leverage remote caGrid Data Services in ways that hide most of the complexity of invoking a remote service and iterating the resulting data.

Client side Components and Details

Generic Data Service Clients

The caGrid Data Services infrastructure supplies three basic client classes which can be used to invoke any arbitrary caGrid Data Service. This capability is due to the query methods of all data services being defined in a common, well known WSDL which each unique service instance imports.

The basic data service client, which is capable of invoking any caGrid Data Service, is the class *gov.nih.nci.cagrid.data.client.DataServiceClient*. The class defines the *query()* method, which takes a CQL Query as single parameter and returns a CQL Query Results instance.

Additionally, the caGrid Data Services infrastructure provides clients which can connect to data services which support WS-Enumeration and the caGrid Bulk Data Transfer infrastructure. Respectively, these clients are *gov.nih.nci.cagrid.data.enumeration.client.EnumerationDataServiceClient* and *gov.nih.nci.cagrid.data.bdt.client.BDTDataServiceClient*. These clients provide public APIs which return an *EnumerationContext* instance or a *BulkDataHandlerReference* respectively. These return types may be used to start up an instance of the Globus provided *ClientEnumIter* class, or make use of the caGrid Bulk Data Transfer Client directly.

Client Side Utilities

The caGrid Data Services infrastructure provides a number of utility classes to make invocation of remote data services a simpler process for the application developer. The package *gov.nih.nci.cagrid.data.utilities* contains utilities which can invoke a standard, enumeration, and BDT data service, as well as tools for handling domain models and working with wsdd and castor mapping files.

The interface *DataServiceIterator* specifies a single *query()* method, which takes a CQL Query and returns an instance of *java.util.Iterator*. The iterator can be used to walk through the results of a query issued to a data service. Three concrete implementations of the *DataServiceIterator* interface are provided, each for communicating with a different type of data service. The *DataServiceHandle* class can be used to invoke a standard caGrid Data Service, while the *EnumerationDataServiceHandle* and *BdtDataServiceHandle* classes are designed for WS-Enumeration and Bulk Data Transfer supporting data services, respectively.

Additionally, this package contains an Iterator utility for handling *CQLQueryResults* instances. The class *CQLQueryResultsIterator* implements *java.util.Iterator,* and has three constructors. The choice of constructor affects the behavior of calling the *next()* method.

- *CQLQueryResultsIterator(CQLQueryResults)*
  - Creates an Iterator over the results which will return materialized objects deserialized using the default type mappings.
- *CQLQueryResultsIterator(CQLQueryResults, Boolean)*
  - Creates an Iterator over the results, and the value of the Boolean parameter indicates if XML strings should be returned from the *next()* method. If the Boolean value is true, XML text of each item is returned, otherwise the results will be deserialized using the default type mappings.
- *CQLQueryResultsIterator(CQLQueryResults, InputStream)*
  - Creates an Iterator over the results, and expects the InputStream will point to a client or server side wsdd. The contents of this wsdd file will be used to configure deserialization of the objects contained in the results.

Most Current Information and Examples

The most current information regarding the caGrid Data Services client side APIs and utilities may be found on the caGrid.org wiki page:
(http://www.cagrid.org/mwiki/index.php?title=Data_Services:Client_API)

Chapter 5 Introduce Extension for Data Services

Introduction

caGrid Data Services can be built with a set of extensions to the Introduce Toolkit. This provides grid service developers with a simple and well defined starting point to create caBIG gold compliant Data Services. When the extension has been selected in creating a new service, the user is presented with the both standard service modification interface, and a new tab is placed on the interface containing options specific to configuring data services.

User Interface Components and Details

Creation Interface

When selected from the caBIG creation dialog in Introduce, the service developer is presented with a dialog allowing selection of a data service style. This dialog also allows the developer to enable the WS-Enumeration or Bulk Data Transfer features of the data service. When the initial creation process is complete, the user is presented with the service modification interface.

Service Modification

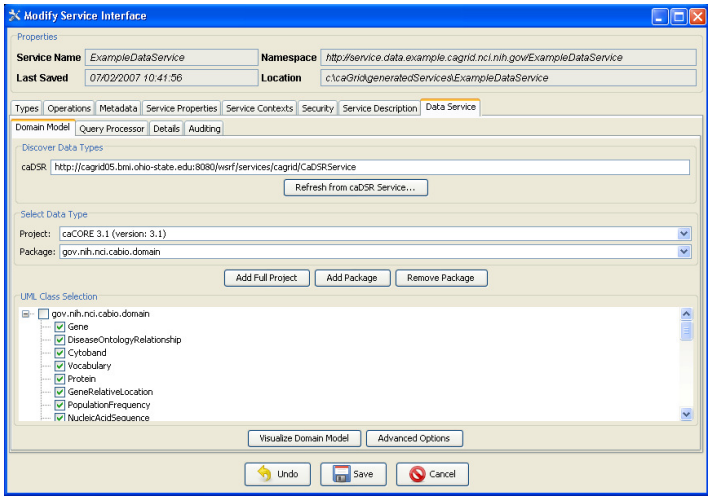

Figure 2 The Data Services configuration UI

The Data Service extension adds a new graphical component to the Introduce Toolkit's service modification view. This tab contains several sub tabs, each containing major points of configuration for the data service:

- Domain Model
  - Provides facilities for selecting the domain of data types available to be queried by the data service
- Query Processor
  - Allows the user to select an implementation of CQL to use in the data service and provides for configuration of its parameters
- Details
  - Serialization of data types, mapping classes to XML Schema element names, and enabling / disabling query validation
- Auditing
  - Selection and configuration of auditors for various processes of the data service infrastructure

- Enumeration (Optional)
    - If the WS-Enumeration or BDT feature is enabled for data services, this tab will be available to choose a server side implementation of the WS-Enumeration spec. Generally this will not need to be changed from the default implementation

Domain Model

On the Domain Model tab, the user may select the types exposed by this data service. Data types are derived from information stored in the caDSR.

The top field labeled ‘caDSR’ indicates the URL of the caDSR grid service to access. The button ‘Refresh from caDSR Service’ queries the grid service to retrieve a list of Projects and UML Packages available in the caDSR. Projects and packages can be selected with the two dropdown menus provided. The ‘Add Full Project’ button will insert all packages and classes from the project into the domain model. ‘Add Package’ will insert a single selected package, and ‘Remove Package’ will take a package out of the domain model. When a package is added, its contents will appear in the tree below the buttons. This tree contains check boxes next to the package and the individual classes. Placing a check next to a class will add it to the domain model as an object which can be queried, and by default, used as a type which can be targeted with a query. Placing a check by a package will automatically add all the classes in that package in the same way.

Each package selected to participate in the domain model must be mapped to an XML schema for serialization across the grid. If a package is selected for which no schema can be found, the data service extension will prompt the service developer to locate it.

The domain model may be displayed graphically by clicking the 'Visualize Domain Model' button. Note that this will cause the domain model information to be retrieved from the caDSR service before it can be displayed, which can be a time consuming process. A dialog with a progress bar keeps the developer apprised of the progress of this process.

The ‘Advanced Options’ button shows a dialog enabling the service developer to control advanced options for domain model selection.

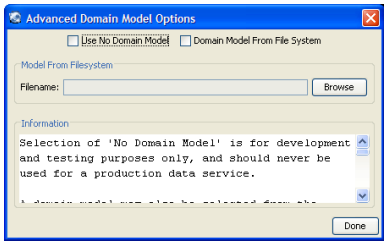

Figure 3Advanced Domain Model Options

The ‘No Domain Model’ check box disables use of a domain model in the data service. Selection of this option is intended for use when testing other aspects of a service. The option to use a domain model from the file system is also intended for testing purposes. Since building a domain model from the caDSR can be a time consuming process, the domain model may be generated from the caDSR elsewhere and specified here. Selecting either of these options will disable selection of the domain model from the caDSR on the Domain Model configuration tab.

Query Processor

The Query Processor tab shows a drop down of all currently available CQL Query Processor implementations. If a query processor other than the one displayed is required, the jar file

containing it may be added to the service by clicking the Add Jar button. When a new jar file is added to the service, both the list of jars and the query processor drop down will update to reflect the changes. The added jar files will be copied into the service's library directory, and deployed with the service. Selecting a class from the drop down will make it the query processor implementation which is invoked by the data service to handle queries at runtime. Its configuration properties (if any) will be shown in the table at the bottom of the screen. This table shows the name of every parameter, its default value, and an editable field where a custom value may be entered. These parameters are stored as service properties in the generated service and are made available at runtime through JNDI. Query processors may optionally supply their own configuration user interface, which can be displayed by clicking the 'Launch Query Processor Configurator' button.

Custom CQL Query processors may be implemented to support querying over a specialized data source by extending the provided base class *gov.nih.nci.cagrid.data.cql.CQLQueryProcessor.*

Details

The Details page allows configuration of lower level functionality in the data service. This tab contains a table allowing configuration of the way each type in the domain model will be serialized by the data service, as well as a flag to indicate if each type may be targeted and returned by a CQL query. Types can be selected either individually or in groups and their serialization is configured by a popup menu. This menu allows selection of either the default serialization, SDK serialization for caCORE SDK generated objects, or a custom serialization. Selecting SDK Serialization assigns the SDK serializer and deserializer factories to the types. Custom serialization presents the user with a dialog box in which to enter the serializer and deserializer classes which will be assigned to the schema type. This information is recorded in the generated client-config.wsdd file, as well as the server-config.wsdd. These configuration files may be used later in the data service clients and utilities to properly deserialize results of queries. This tab also allows the service developer to select what type of query validation they desire to be performed. Selecting to 'validate CQL syntax' will cause the Data Service to ensure that the submitted CQL query conforms to the CQL schema. Selecting the 'validate domain model' option will cause the data service to parse the CQL query against the domain model, ensuring that all aspects of the query remain within the confines of the exposed domain model. Using these options in conjunction ensures that every query that reaches the CQL query processor implementation is both syntactically correct and conforms to the domain model, which can substantially simplify checking for potential errors in the query processor.

Auditing

The Auditing page allows auditing settings to be configured for the data service. Multiple auditors may be added to the service, each of which may handle auditing events in its own way. Multiple instances of the same auditor type may be added as well, allowing configuration options to dictate their behavior and handling of auditing events.

Four points of the data service query process may be audited:

- Query Begins
    - This event is fired when a query is first submitted to the data service before any

validation or processing has been done.

- Validation Failure
  - This event is fired when a query fails the validation process. If validation is not enabled, this event will never be fired
- Query Processing Failure
  - If a query should fail to process correctly, this event is fired. The reason (exception) causing the failure is included in this event.
- Query Results
  - When a query completes successfully, this event is fired. The results of the query are available at this point as well.

Enumeration

The Enumeration page appears only when a data service's enumeration or BDT support features are enabled. This tab allows the service developer to select a server side implementation to support the WS-Enumeration specification. In caGrid 1.1, there are five implementations which may be selected. Two are supplied with the Globus WS-Enumeration support, and three are part of caGrid itself. The default selection is the caGrid implementation which uses Java 5's concurrent package to fully support the specification. The other implementations support fewer features, but may be desirable for duplicating functionality of other WS-Enumeration enabled services.

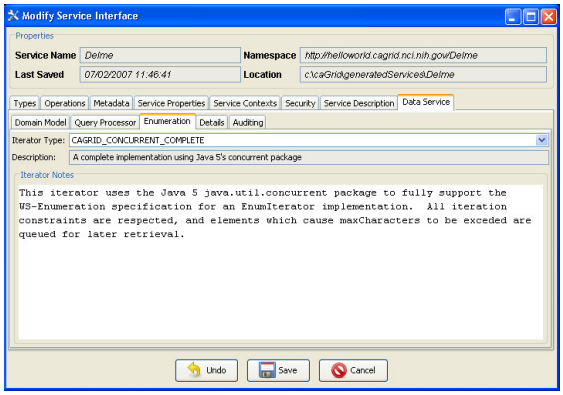

Figure 4 Enumeration Implementation Selection

Functionality of the Extension

The Introduce Toolkit allows for extensions to be included in the service development process which add functionality to almost any step of the service build process. The Data Service extension makes use of three of these extension points. The first extension point used is immediately following service creation. This post-creation operation makes the following changes to the generated service:

- The data service WSDL file is copied into the service.
  - Optionally, if WS-Enumeration and / or BDT support is enabled, the WSDLs for these will be included as well.
- The data service schemas for CQL and Domain Models are copied into the generated service.
- The data service libraries are copied in to the service.
- The base data service query method is added, and defined to be implemented in the copied libraries and provided in the copied WSDL.
  - If WS-Enumeration and / or BDT support is enabled, specialized query methods for each of these are added.
- Data Service specific service properties are added.

The next step in the build process added by the Data Service extension is invoked when modifications to the service are saved. This operation happens before the standard operations provided by Introduce are executed which generate code for user defined methods, edit WSDL files, and copy schemas. This pre code generation operation makes the following changes to the service:

- Service properties are modified.
  - The property defining the query processor class implementation is populated with the user's defined value.
  - Properties required by the query processor implementation for initialization are created and added to the service.
- Adds domain model metadata.
  - The caDSR grid service is contacted to generate a domain model. This process can be time consuming depending on the network connection to the caDSR and the specific package and project combination selected.
  - If the service developer has defined an XML file containing the domain model definition, it will be copied into the service.

Following the execution of the pre code generation extension, the standard Introduce build operations are invoked. When this is complete, the final extension operation is invoked on the new service. This operation modifies the generated Eclipse files to add the new jar library files to the project's classpath.

Most Current Information

For the most up to date information regarding the Introduce extension for caGrid Data Services, please see the caGrid.org wiki page: *(http://www.cagrid.org/mwiki/index.php?title=Data_Services:Introduce_Extension:1.1)*.

Chapter 6 Data Service Styles

Introduction

This section describes the extensible and pluggable system for constructing data services on a repeatable basis known as Data Service Styles. Service styles are intended to allow service developers to repeatedly create data services which require a specific set of functionality to be added or have a well defined configuration and setup process.

Service Styles Architecture

Data service styles may be added to the data service extension to provide additional functionality to the service creation and configuration processes, and are selected by the service developer when a service is first created. Styles may be installed at any time after the primary caGrid Data Services extension has been installed by adding to the *styles* directory found in the installed data service extension directory. Each style must provide its own directory in which files it uses will be placed, but no restriction is made on the naming of these directories. At the top level of each style directory, a *style.xml* file must exist, describing the style. This document describes the style's name, which caGrid and Data Service versions it is compatible with, and information on which classes are to be loaded for each component of the style. If the service developer selects no style at service creation time, the service is created with only the standard data services components and query method, and ready to have a domain model, query processor, and other data service requirements selected and configured.

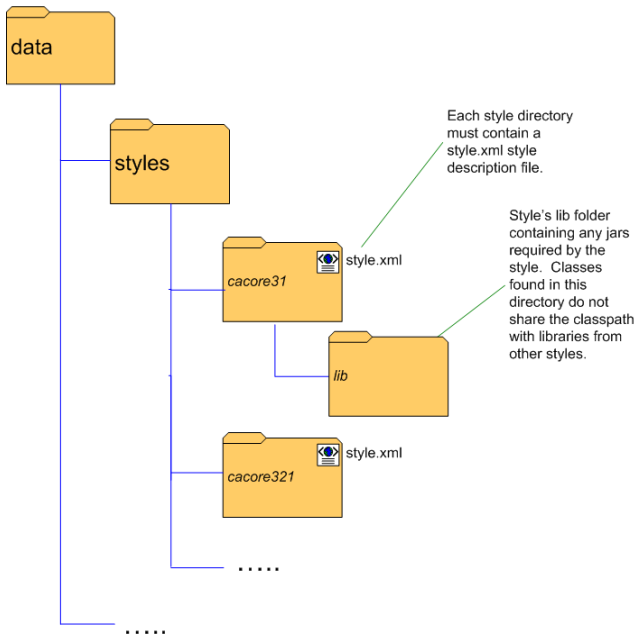

Figure 5 Data Service Styles Directory Structure

Functionality Extended by Styles

Data Service styles may add functionality to any or all of the following areas of service

development with the Introduce toolkit:

- Creation Wizard
    - The service style may supply a list of wizard panels to be displayed and chained together in a wizard-like fashion to break the setup process for the service style into a series of steps. These panels will be shown in a wizard dialog when a service style is selected at service creation time.
- Post-creation processing
    - Just as Introduce extensions may add functionality to the service creation process, data service styles may add processing capabilities to this step.
- Modification User Interface
    - The style may supply a graphical panel which will be added to the data service tab in the Introduce service modification viewer. This tab can be used to configure any style-specific options in the service.
- Post-code generation processing
    - The style may add functionality to the code generation process of service modification. This processing will be invoked each time the service is modified and saved in Introduce.

Chapter 7 caCORE Support

Overview

caGrid Data Services provide support for exposing a caCORE data source via the data services grid interface. This allows for rapid migration of existing silver systems to the gold level of the grid. Support for caCORE generated systems is provided by specialized data service styles. These styles add functionality to the service creation and code generation steps of building a service with the Introduce toolkit which make this migration a straightforward process.

At the time of this writing, caGrid 1.1 provides support for caCORE versions 3.1, 3.2, and 3.2.1. Much of the support for these versions overlaps, so differences are noted in this document only where relevant.

caCORE Service Styles

The caCORE data service styles may be selected when the service is first created by the service developer, a dialog is presented allowing selection of the style, as well as optional features of the data service infrastructure, such as WS-Enumeration and Bulk Data Transfer support. Selecting a service style in the drop down at the top of the dialog populates a brief description of the style in the text box near the center of the dialog. Both of the caCORE SDK styles supply a set of panels to be presented in a service creation wizard, so selecting either of them will bring up the wizard with the appropriate set of panels already populated.

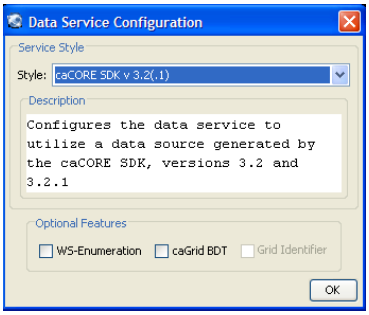

Figure 6 Data Service Creation

The first panel of this wizard contains a brief description of the SDK data service creation process. Behind the scenes, this panel also copies schemas and libraries required for the caCORE query processor in to the service.

The next panel requires the service developer to select the directory in which the client libraries for the caCORE application service reside. The directory is first verified to determine if a valid client.jar file exists in it, and then the required libraries are copied from the directory and into the service. In the case of caCORE versions 3.2 and 3.2.1, this panel also presents the option to use the caCORE's local access API. Selecting this option requires the additional specification of the directory in which the configuration files for the local application service reside. The files contained here are packaged into a jar file which is then placed in the service's

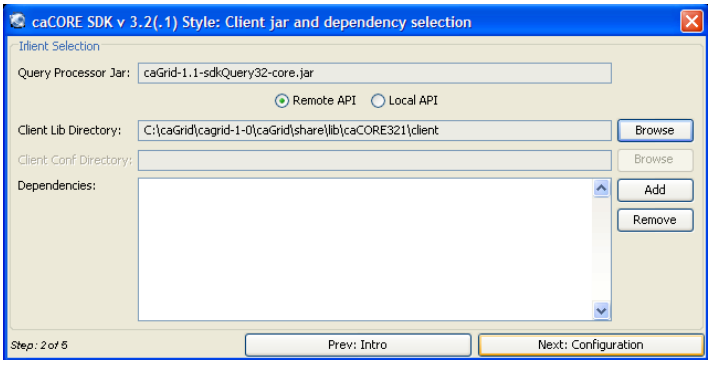

Figure 7 Application Service Configuration

library directory. In this way, the local caCORE APIs will be able to locate them at runtime.

The next panel allows the service developer to select the URL which will be used for communication with a remote application service. If the local API is selected, this option will be disabled. Additionally, this panel provides an option to enable CSM security with the application service, and then requires specification of the CSM context name and a configuration file.

The fourth panel gives the service developer the opportunity to select the domain model which the data service is to expose. Generally, this domain model should be derived from the caDSR. A project name, and one or many packages from the project may be selected to participate in the domain model on this panel. Behind the scenes, the serialization settings of all objects found in this domain model are automatically configured to correctly translate objects generated by the caCORE SDK to and from XML.

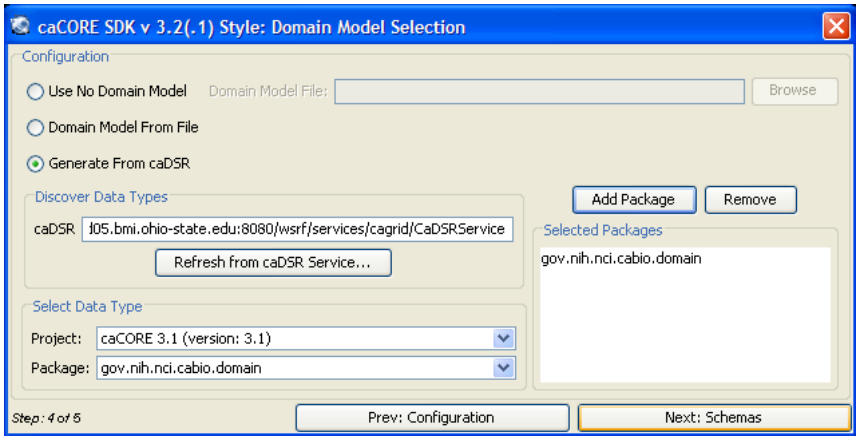

Figure 8 Selection of Domain Model

The last panel of the wizard requires the service developer to select the schema or schemas used to serialize the objects in the domain model. Each package is displayed, along with the wizard's best guess for an XML namespace to which the package should be mapped. Using the resolve button to the right of these, the developer can select the XML schema from a variety of sources. Once a schema has been specified, the wizard panel attempts to map the data types in the domain model's package to elements in the XML schema. If this process is successful for each type, the status of the package to schema mapping will change to 'Found'. Once all packages have been mapped to XML schema, the wizard can be completed by clicking the 'Done' button.

Once the service is generated, the standard Introduce service modification interface will appear, containing an additional tab for editing data service specific parameters. The caCORE service styles will have already set this information appropriately, and no additional configuration is needed to have a complete data service backed by the caCORE SDK. As always, additional methods and types may be added to the service to support domain specific logic, and security options may be changed using the usual means for doing so via the service modification interface.

Chapter 8 References

This appendix could include lists and hypertext links, where appropriate, to technical manuals, articles, scientific publications, etc. Examples are shown:

Technical Manuals/Articles

- National Cancer Institute. "caCORE 2.0 Technical Guide", ftp://ftp1.nci.nih.gov/pub/cacore/caCORE2.0_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

Scientific Publications

caBIG Material

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

caCORE Material

1. **caCORE:** http://ncicb.nci.nih.gov/core
2. **caBIO:** http://ncicb.nci.nih.gov/core/caBIO
3. **caDSR:** http://ncicb.nci.nih.gov/core/caDSR
4. **EVS:** http://ncicb.nci.nih.gov/core/EVS

5. **CSM:** http://ncicb.nci.nih.gov/core/CSM

Appendix A Glossary

Following is an *example* list of terms and their definitions.

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
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
HIPPA	Health Insurance Portability and Accountability Act
HLGT	High Level Group Term
HLT	High Level Term
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
MedDRA	Medical Dictionary for Regulatory Activities
metadata	Definitional data that provides information about or documentation of other data.
MGED	Microarray Gene Expression Data
MMHCC	Mouse Models of Human Cancers Consortium
MO	MGED Ontology
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
OIL	Ontology Inference Layer
OilEd	Ontology editor allowing you to build ontologies using DAML+OIL

Term	***Definition***
OLLT	Obsolete Lower Level Terms
OMG	Object Management Group
ORM	Object Relational Mapping
PT	Preferred Term
RDBMS	Relational Database Management System
SDK	Software Development Kit
Semantic connector	A development kit to link model elements to NCICB EVS concepts.
SOC	System Organ Class
SPORE	Specialized Programs of Research
SQL	Structured Query Language
SSC	Special Search Categories
UI	User Interface
UID	User Identification
UML	Unified Modeling Language
UMLS	Unified Medical Language System
UPT	User Provisioning Tool
URL	Uniform Resource Locators
VD	Value Domain
WAR	Web Application Archive
WSDL	Web Services Description Language
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

Index

It is very easy to generate an index and extremely helpful to the manual user. See the *Technical Style Guide* located at: https://cabig.nci.nih.gov/working_groups/Training_SLWG/Documents/Technical_Pubs_Style_Guide_021405_jbh.pdf for directions for inserting index markers and generating an index in MS Word (PC).