caCORE
3.2 Technical Guide

National Cancer Institute®

Center for Bioinformatics

Revised December 22, 2006

Credits and Resources

caCORE 3.2 Contributors			
Development		***Technical Guide***	***Program Management***
Prerna Aggarwal[3]	Art Lian[4]	Steve Alred[3]	Tara Akhavan[1]
Gavin Brennan[7]	Ying Long[1]	Michael Connolly[1]	Steve Alred[3]
Vanessa Caldwell[7]	Wei Lu[7]	Dan Dimitru[4]	Bill Britton[7]
Ben Chapman[7]	Christophe Ludet[3]	Wendy Erickson-Hirons[6]	Peter Covitz[2]
Michael Connolly[1]	Harsh Marwaha[3]	Jill Hadfield [2]	Charles Griffin[4]
Eric Copen[4]	Doug Mason[1]	Jane Jiang[3]	Frank Hartel[2]
Kim Diercksen[7]	Kunal Modi[4]	Shan Jiang[1]	George Komatsoulis[2]
Dan Dumitru[4]	Shaziya Muhsin[1]	George Komatsoulis[2]	Sichen Liu[2]
Craig Fee[7]	Satish Patel[4]	Christophe Ludet[3]	Krishnakant Shanbhag[2]
Gilberto Fragoso[2]	Kim Ong[5]	Doug Mason[1]	Denise Warzel[2]
Steven Hunter [4]	Konrad Rokicki [1]	Kunal Modi[4]	
Shan Jiang[1]	Ralph Rutherford[7]	Shaziya Muhsin[1]	
Sriram Kalyanasundaram[7]	Claire Wolfe[7]	Satish Patel[4]	
Doug Kanoza[7]	Ye Wu[1]	Tracy Safran[1]	
Alan Klink[7]	Rob Wynne[8]	Krishnakant Shanbhag[2]	
Vinay Kumar[4]	Bob Wysong[7]	Denise Warzel [2]	
Norval Johnson[7]	Nafis Zebarjadi[1]	Nafis Zebarjadi[1]	
Thai Le[1]	Jennifer Zeng[1]		
[1] Science Applications International Corporation (SAIC)	[3] Oracle Corporation	[5] Northrop-Grumman	[7] Terpsys
[2] National Cancer Institute Center for Bioinformatics (NCICB)	[4] Ekagra	[6] Northern Taiga Ventures, Inc.	[8] Management System Designers, Inc.

The following NCICB listserv facilities are pertinent to this Technical Guide.

LISTSERV	***URL***	***Name***
caBIO_Users	https://list.nih.gov/archives/cabio_users.html	caBIO Users Discussion Forum
caBIO_Developers	https://list.nih.gov/archives/cabio_developers.html	caBIO Developers Discussion Forum
caCORE_SDK_Users	https://list.nih.gov/archives/cacore_sdk_users-l.html	caCORE SDK Users Discussion Forum
caCORE_SDK_Developers	https://list.nih.gov/archives/cacore_sdk_dev-l.html	caCORE SDK Developers Discussion Forum
caDSR_Users	https://list.nih.gov/archives/cadsr_users.html	Cancer Data Standards Repository
NCI EVS Listserv	https://list.nih.gov/archives/ncievs-l.html	NCI Vocabulary Services Information

GForge is a cross project collaboration site for NCICB caCORE developers located at http://gforge.nci.nih.gov/projects/cacore/.

caCORE Area/ Tool	***URL***	***Description***
cacoretoolkit	http://gforge.nci.nih.gov/projects/cacoresdk/	The NCICB caCORE Software Development Kit is a set of tools designed to aid in the design and creation of a 'caCORE-like' software system.
caBIO DB	http://gforge.nci.nih.gov/projects/cabiodb/	Schema and related data for the caBIO database.
caCORE Perl API	http://gforge.nci.nih.gov/projects/cabioperl/	The purpose of this project is to develop a Perl-based interface to caCORE.
Common Security Module (CSM)	http://gforge.nci.nih.gov/projects/security/	The CSM provides application developers with powerful security tools to allow application developers to integrate security with minimal coding effort.
Common Logging Module (CLM)	http://gforge.nci.nih.gov/projects/logging/	The CLM is a powerful set of auditing and logging tools implemented in a flexible and comprehensive solution.

TABLE OF CONTENTS

Chapter 4

PREFACE

Purpose

The *caCORE 3.2 Technical Guide* describes a core infrastructure called Cancer Common Ontologic Representation Environment (caCORE), an open-source standards-based semantics computing environment and tool set created by the National Cancer Institute Center for Bioinformatics (NCICB).

This guide describes:

- the purpose, architecture and components of caCORE including Enterprise Vocabulary Services (EVS), Cancer Data Standards Repository (caDSR), Cancer Bioinformatics Infrastructure Objects (caBIO), Cancer Models Database (caMOD), Common Security Module (CSM), and Common Logging Module (CLM).
- the APIs for accessing the caCORE system including Java, Perl, Web services, and XML-HTTP.
- the data sources accessible through the caCORE APIs.
- an overview of the Unified Modeling Language (UML).

Release Schedule

This guide is updated for each caCORE release. It may be updated between releases if errors or omissions are found. The current document refers to the 3.2 version of caCORE, which was released in December 2006 by NCICB.

Audience

The primary audience of this guide is the application developer who wants to learn about the architecture and use of caCORE-like systems and/or requires access to one or more caCORE APIs. A ‘caCORE-like’ system can be generated using the caCORE Software Development Kit (SDK). For more information, see the caCORE SDK 3.2 Programmer’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 caCORE resources in software applications, it assumes that you have experience with building and using complex data systems.

Neither caCORE nor this documentation are intended for "end" users, such as individual health professionals or members of the general public - unless they are also software developers.

How To Use This Guide

This brief overview explains what you will find in each section of this guide.

Chapter 1 provides an overview of caCORE.

Chapter 2 describes the architecture of the caCORE system.

Chapter 3 provides installation instructions and API use instructions.

Chapter 4 describes the Enterprise Vocabulary Services (EVS) and the EVS API.

Chapter 5 describes the Cancer Data Standards Repository (caDSR) and the caDSR API.

Chapter 6 describes the Cancer Bioinformatics Infrastructure Objects (caBIO) and the caBIO API.

Chapter 7 describes the Common Package.

Chapter 8 describes the Common Security Module (CSM).

Chapter 9 describes the Common Logging Module (CLM).

Appendix A provides a general background and notation of the Unified Modeling Language (UML).

Appendix B provides a list of references used to produce this guide or referred to within the text.

Additional caCORE Documentation

The caCORE 3.2 Release Notes contain a description of the end user tool enhancements and bug fixes included in this release.

The caCORE 3.2 JavaDocs contain the current caCORE API specification

The caCORE SDK 3.2 Programmer’s Guide contains detailed instruction on the use of the SDK and how it helps create a caCORE-like software system.

CHAPTER 1

OVERVIEW TO caCORE

This chapter provides an overview of the NCICB caCORE infrastructure.

Topics in this chapter include:

- *Architecture Overview* on this page
- *Domain Models in caCORE* on page 4

Architecture Overview

The NCI Center for Bioinformatics (NCICB) provides biomedical informatics support and integration capabilities to the cancer research community. NCICB has created a core infrastructure called Cancer Common Ontologic Representation Environment (caCORE), a data management framework designed for researchers who need to be able to navigate through a large number of data sources.

By providing a common data management framework, caCORE helps streamline the informatics development throughout academic, government and private research labs and clinics. The components of caCORE support the semantic consistency, clarity and comparability of biomedical research data and information.

caCORE is an open-source enterprise architecture for NCI-supported research information systems, built using formal techniques from the software engineering and computer science communities. The four characteristics of caCORE include:

- Model Driven Architecture (MDA)
- *n*-tier architecture with open Application Programming Interfaces (APIs)
- Use of controlled vocabularies, wherever possible
- Registered metadata

The use of MDA and *n*-tier architecture, both standard software engineering practices, allows for easy access to data, particularly by other applications. The use of controlled vocabularies and registered metadata, less common in conventional software practices, requires specialized tools, generally unavailable.

As a result, the NCICB (in cooperation with the NCI Office of Communications) has developed the Enterprise Vocabulary Services (EVS) system to supply controlled vocabularies, and the Cancer Data Standards Repository (caDSR) to provide a dynamic metadata registry.

When all four development principles are addressed, the resulting system has several desirable properties. Systems with these properties are said to be “caCORE-like”.

1. The *n*-tier architecture with its open APIs frees the end user (whether human or machine) from needing to understand the implementation details of the underlying data system to retrieve information.
2. The maintainer of the resource can move the data or change implementation details (Relational Database Management System, and so forth) without affecting the ability of remote systems to access the data.
3. Most importantly, the system is ‘semantically interoperable’; that is, there exists runtime-retrievable information that can provide an explicit definition and complete data characteristics for each object and attribute that can be supplied by the data system.

Domain Models in caCORE

The components that comprise caCORE are EVS, caDSR, caBIO, CSM, and CLM. Each is described briefly below and is described in detail in its own chapter.

Enterprise Vocabulary Services (EVS)

EVS provides controlled vocabulary resources that support the life sciences domain, implemented in a description logics framework. EVS vocabularies provide the semantic 'raw material' from which data elements, classes, and objects are constructed.

Cancer Data Standards Repository (caDSR)

The caDSR is a metadata registry, based upon the ISO/IEC 11179 standard, used to register the descriptive information needed to render cancer research data reusable and interoperable. The caBIO, EVS and caDSR data classes are registered in the caDSR, as are the data elements on NCI-sponsored clinical trials case report forms.

Cancer Bioinformatics Infrastructure Objects (caBIO)

The caBIO model and architecture is the primary programmatic interface to caCORE. Each of the caBIO domain objects represents an entity found in biomedical research. Unified Modeling Language™ (UML) models of biomedical objects are implemented in Java as middleware connected to various cancer research databases to facilitate data integration and consistent representation. Examining the relationships between these objects can reveal biomedical knowledge that was previously buried in the various primary data sources.

Common Security Model (CSM)

CSM provides a flexible solution for application security and access control with three main functions:

- Authentication to validate and verify a user's credentials
- Authorization to grant or deny access to data, methods, and objects
- User Authorization Provisioning to allow an administrator to create and assign authorization roles and privileges.

Common Logging Module (CLM)

CLM provides a separate service under caCORE for Audit and Logging Capabilities. It also comes with a web based locator tool. It can be used by a client application directly, without the application using any other components like CSM.

CHAPTER 2

CACORE ARCHITECTURE

This chapter describes the architecture of the caCORE system. It includes information about the major components, such as security, logging, database object-relational mappings (ORM), client-server communication, and how the system connects to non-ORM systems, such as EVS. It also describes the layout of the caCORE system packages.

Topics in this chapter include:

caCORE System Architecture

The caCORE infrastructure exhibits an n-tiered architecture with client interfaces, server components, backend objects, data sources, and additional backend systems (*Figure 2.1*). This n-tiered system divides tasks or requests among different servers and data stores. This isolates the client from the details of where and how data is retrieved from different data stores. The system also performs common tasks such as logging and provides a level of security.

Clients (browsers, applications) receive information from backend objects. Java applications also communicate with backend objects via domain objects packaged within the client.jar. Non-Java applications can communicate via SOAP (Simple Object Access Protocol). Back-end objects communicate directly with data sources, either relational databases (using Hibernate) or non-relational systems (using, for example, the Java RMI API).

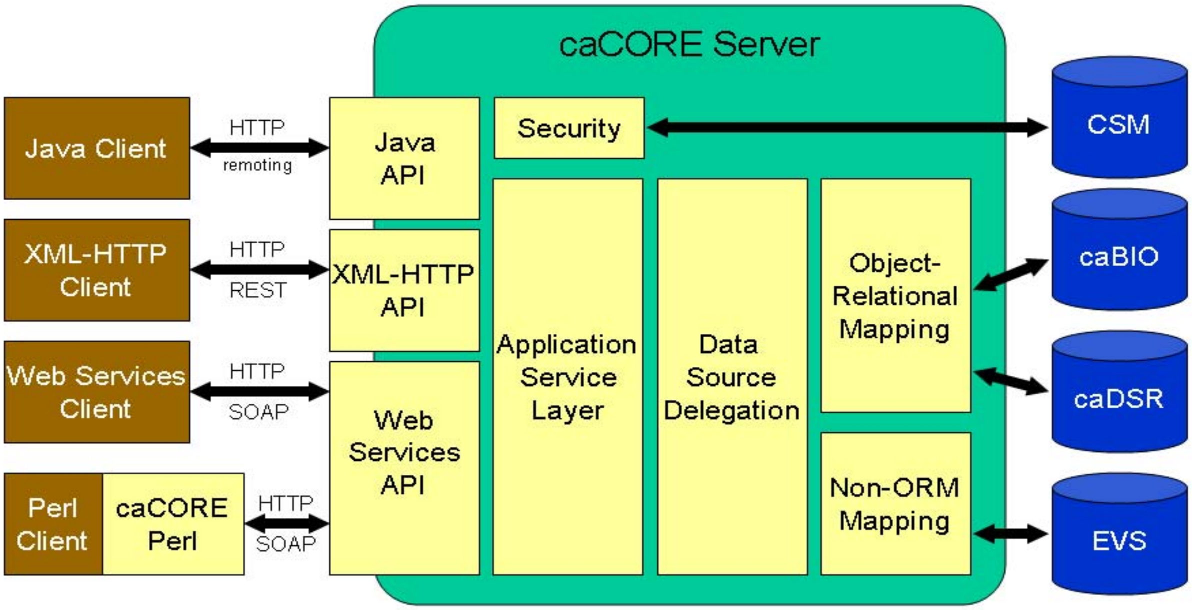


Figure 2.1 caCORE Architecture

Most of the caCORE infrastructure is written in the Java programming language and leverages reusable, third-party components.

The infrastructure is composed of the following layers:

The Application Service layer—consolidates incoming requests from the various interfaces and translates them to native query requests that are then passed to the data layers. This layer is also responsible for handling client authentication and access control using the Java API. (This feature is currently disabled for the caCORE system running at NCICB; all interfaces provide full, anonymous read-only access to all data.)

The Data Source Delegation layer—is responsible for conveying each query that it receives to the respective data source that can perform the query. The presence of this layer enables multiple data sources to be exposed by a single running instance of a caCORE server.

Object-Relational Mapping (ORM)—is implemented using Hibernate. Hibernate is a high performance object/relational persistence and query service for Java. Hibernate provides the ability to develop persistent classes following common object-oriented (OO) design methodologies such as association, inheritance, polymorphism, and composition.

The *Hibernate Query Language* (*hql*), designed as a "minimal" object-oriented extension to SQL, provides a bridge between the object and relational databases. Hibernate allows for real world modeling of biological entities without creating complete SQL-based queries to represent them.

Access to non-relational (non-ORM data sources), such as Enterprise Vocabulary Services (EVS), is performed by objects that follow the façade design pattern. These objects make the task of accessing a large number of modules/functions much simpler by providing an additional interface layer that allows it to interact with the rest of the caCORE system.

Security is provided by the Common Security Module (CSM). The CSM provides highly granular access control and authorization schemes. For more information, see *Chapter 8*.

Enterprise logging is provided by the Common Logging Module (CLM). The CLM provides a separate service under caCORE for audit and logging capabilities. This is similar to the output generated by Apache log4j, but includes information for auditing. For more information, see *Chapter 9*.

Client Technologies

Applications using the Java programming language can access caBIO, caDSR, and, EVS directly through the domain objects provided by the client.jar (see *Chapter 3 Interacting with caCORE*.) The network details of the communication to the caCORE servers are abstracted away from the developer. Hence developers need not deal with issues such as network and database communication, but can instead concentrate on the biological problem domain.

Perl application programmers can access the same information by using the caCOREperl API, which is a Perl application programming interface to caCORE's hosted vocabulary, metadata, and biomedical data.

The caCORE system also allows non-Java and Perl Applications to use SOAP clients to interface with caCORE Web services. SOAP is a lightweight XML-based protocol for the exchange of information in a decentralized, distributed environment. It consists of an envelope that describes the message and a framework for message transport. caCORE uses the open source Apache Axis package to provide SOAP-based web services to users. This allows other languages, such as Python or Perl to communicate with caCORE objects in a straightforward manner.

The caCORE architecture includes a presentation layer that uses a J2SE application server (such as Tomcat or JBoss). The JSPs (Java Server Pages) are web pages with Java embedded in the HTML to incorporate dynamic content in the page. caCORE also employs Java Servlets, which are server-side Java programs, that web servers can run to generate content in response to client requests. All logic implemented by the presentation layer uses Java Beans, which are reusable software components that work with Java. All caCORE objects can be transformed into XML, the eXtensible Markup Language, as a universal format for structured data on the Web.

Communication between the client interfaces and the server components occurs over the Internet using the HTTP protocol. The server components are deployed in a web application container as a .war (Web archive) file which communicates with the backend relational database management system that contains the actual data. In some cases, such as EVS, the caCORE server communicates to external backend systems.

By using XSL/XSLT, the extensible stylesheet language for expressing stylesheets and XSL Transformations (XSLT), as a language for transforming XML documents, nonprogrammers can transform the information in the caBIO objects for use in documents or other systems.

Major caCORE Domain Packages

Table *2.1* shows the major caCORE domain packages from which you can access the Java interfaces and classes, including caBIO, EVS, caDSR, and Common. All of the objects in the domain package implement the java.io.serializable interface. To view the JavaDocs page for each package, go to http://ncicb.nci.nih.gov/NCICB/content/ncicblfs/caCORE3-2_JavaDocs.

Package	***Description***

EVS	Contains the domain-specific classes in the EVS object package such as DescLogicConcept, MetaThesaurusConcept, etc. For a list of the domain objects, see the *caCORE EVS API* on page 62.
	`gov.nih.nci.evs.domain`
	`gov.nih.nci.evs.query`

caDSR	Contains the domain-specific classes in the caDSR object package such as DataElement, Administered-Component, etc. For a list of the domain objects, see the *caDSR API* on page 90. `gov.nih.nci.cadsr.domain`
caBIO	Contains the domain-specific classes in the caBIO object package such as Gene, Chromosome, etc. For a list of the domain objects, see the *caBIO API* on page 109. `gov.nih.nci.cabio.domain`
Common	Contains one class, DatabaseCrossReference, which is used by the caBIO domain package objects to provide links to related data hosted by other sources. See the *Common Package API* on page 121. `gov.nih.nci.common.domain`

Table 2.1 caCORE packages and descriptions

Figure 2.2 displays a package for each application area in caCORE 3.2, described below.

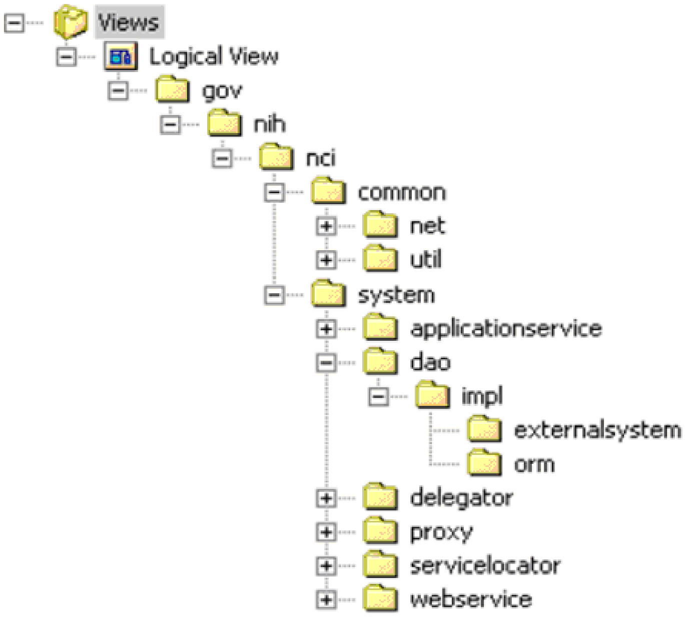


Figure 2.2 caCORE packages and descriptions

In addition to domain packages, the caCORE API specification includes the following framework packages:

Common

The common package (`gov.nih.nci.system.common`) contains utility classes and network classes.

System

The system package has subpackages of application service package (described in the *Client Technologies* on page 9), data access package, delegate/service locator package, proxy package, and web service package.

Data Access

The data access package (`gov.nih.nci.system.dao`) is the layer at which the query is parsed from objects to the native query, the query is executed, and the result sets are converted back to domain objects results. This layer has implementation for internal ORM and external data access layer for querying other subsystems.

Delegate/Service Locator

The delegate/service locator package (`gov.nih.nci.system.delegator`) uses the resource locator (`gov.nih.nci.system.servicelocator`) to identify the data source where a query would be performed. For instance, if a query is for a Gene object with the criterion Taxon, this layer identifies the data source information as an internal RDBMS data base and uses the Object Relational Mapping (ORM) data access layer instance.

Proxy Interface

The proxy interface package (`gov.nih.nci.system.proxy`) is the gateway for the requests from Java and platform independent webservice clients.

Web Service

The Web service package (`gov.nih.nci.system.webservice`) contains the Web service wrapper class that uses Apache's Axis.

CHAPTER

3

INTERACTING WITH CACORE

This chapter describes the service interface layer provided by the caCORE architecture and gives examples of how to use each of the interface APIs.

Topics in this chapter include:

caCORE Service Interface Paradigm

The caCORE architecture includes a service layer that provides a single, common access paradigm to clients using any of the provided interfaces. As an object-oriented middleware layer designed for flexible data access, caCORE relies heavily on strongly-typed objects and an object-in/object-out mechanism. The methodology used for obtaining data from caCORE systems is often referred to as query-by-example, meaning that the inputs to the query methods are themselves domain objects that provide the criteria for the returned data. The major benefit of this approach is that it allows for run-time semantic interoperability when used as part of the caCORE infrastructure, which provides shared vocabularies and a metadata registry.

The basic order of operations required to access and use a caCORE system is as follows:

1. Ensure that the client application has knowledge of the objects in the domain space.
2. Formulate the query criteria using the domain objects.

3. Establish a connection to the server.
4. Submit the query objects and specify the desired class of objects to be returned.
5. Use and manipulate the result set as desired.

There are four primary application programming interfaces (APIs) native to caCORE systems. Each of the available interfaces described below uses this same paradigm to provide access to the caCORE domain model, but with minor changes relating primarily to the syntax and structure of the clients. The following sections describe each API, identify installation and configuration requirements, and provide code examples.

Java API

The Java API provides direct access to domain objects and all service methods. Because caCORE is natively built in Java, this API provides the fullest set of features and capabilities.

Installation and Configuration

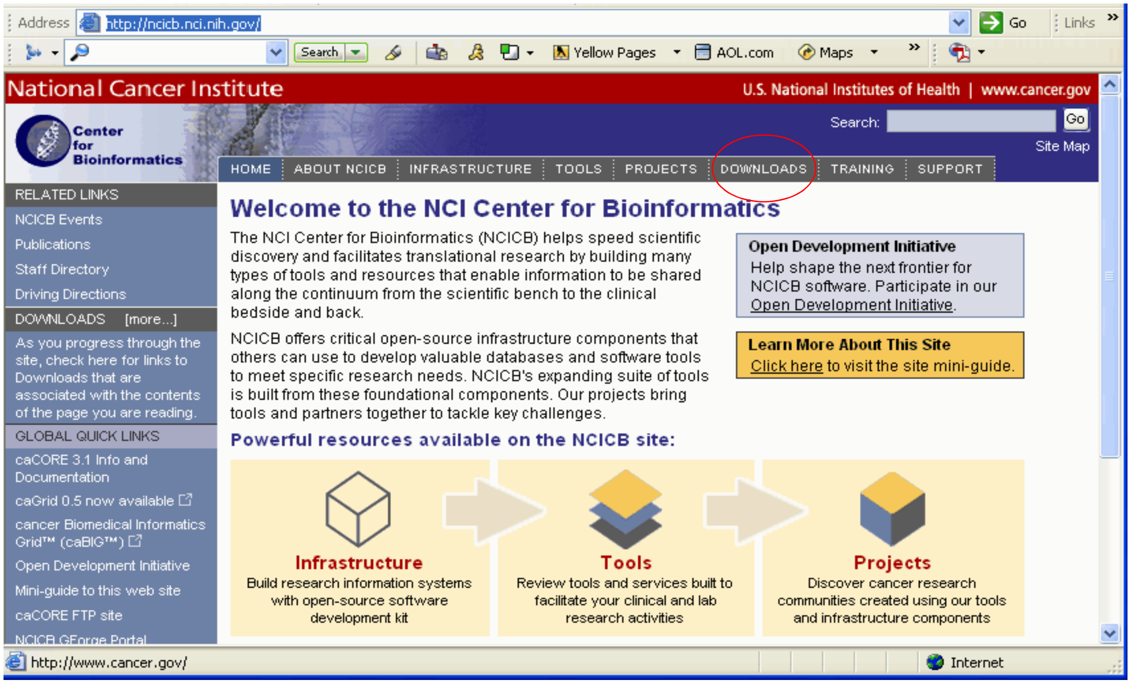
The caCORE Java API uses the following on the client machine (Table *3.1*).

Software	***Version***	***Required?***
Java 2 Platform Standard Edition Software 5.0 Development Kit (JDK 5.0)	1.5.0 or higher	Yes
Apache Ant	1.6.5 or higher	Yes

Table 3.1 caCORE Java API Client software

Accessing the caCORE system also requires an Internet connection.

To use the Java API, download the client package provided on the NCICB web site (*Figure 3.1*).



As you progress through the site, check here for links to Downloads that are associated with the contents of the page you are reading.

Welcome to the NCI Center for Bioinformatics

The NCI Center for Bioinformatics (NCICB) helps speed scientific discovery and facilitates translational research by building many types of tools and resources that enable information to be shared along the continuum from the scientific bench to the clinical bedside and back.

NCICB offers critical open-source infrastructure components that others can use to develop valuable databases and software tools to meet specific research needs. NCICB's expanding suite of tools is built from these foundational components. Our projects bring tools and partners together to tackle key challenges.

Open Development Initiative
Help shape the next frontier for NCICB software. Participate in our Open Development Initiative.

Learn More About This Site
Click here to visit the site mini-guide.

Powerful resources available on the NCICB site:

Infrastructure: Build research information systems with open-source software development kit

Tools: Review tools and services built to facilitate your clinical and lab research activities

Projects: Discover cancer research communities created using our tools and infrastructure components

Figure 3.1 Downloads section on the NCICB website

1. Open your browser and go to http://ncicb.nci.nih.gov.
2. Click the Download link on the menu bar.
3. Scroll down to the section titled caBIO and click on the Download link.
4. In the provided form, enter your name, email address and institution name and click to Enter the Download Area.
5. Accept the license agreement.
6. On the caCORE downloads page, download the caCORE Zip file from the Primary Distribution section.
7. Extract the contents of the downloadable archive to a directory on your hard drive (for example, `c:\cacore` on Windows or `/usr/local/cacore` on Linux). The extracted directories and files include the following:

Directories and Files	***Description***	***Component***
build.xml	Ant build file	Build file

Table 3.2 Extracted directories and files in caCORE client package

Directories and Files	***Description***	***Component***
TestClient.java	Java API client sample	Sample code
TestDSR.java	Java API caDSR client sample	
TestEVS.java	Java API EVS client sample	
TestXML.java	XML utility sample	
WSTestClient.java	Web services Java sample	
EVSWSClient.java	Web services EVS sample	
lib directory	contains required jar files	
*-client.jar	domain objects	
spring.jar	Spring framework	HTTP remoting
hibernate3.jar	Hibernate	ORM layer
axis.jar	Apache Axis	web services (Java implementation)
saaj.jar	SOAP API for Java	
jaxrpc.jar	Java API for XML-based RPC	
wsdl4j-1.5.1.jar	WSDL for Java	
log4j-1.2.13.jar	logging utilities	logging
commons-logging.jar		
commons-discovery-0.2.jar		
castor-1.0.2.jar	Castor serializer/deserializer	XML conversion
xercesImpl.jar	Apache Xerces XML parser	
**.xsd*	XML schemas for objects	
activation.jar		
cglib-2.1.3.jar		
xml.properties		
xml-mapping.xml		
conf directory		
remoteService.xml		
deploy.wsdd		
log4j.properties	Logging utilities configuration properties	

Table 3.2 Extracted directories and files in caCORE client package (Continued)

All of the jar files provided in the lib directory of the caCORE client package in addition to the files in the conf directory are required to use the Java API. These should be included in the Java classpath when building applications. The `build.xml` file that is included demonstrates how to do this when using Ant for command-line builds. If you are using an integrated development environment (IDE) such as Eclipse, refer to the tool's documentation for information on how to set the classpath.

A Simple Example

To run the simple example program after installing the caCORE client, open a command prompt or terminal window from the directory where you extracted the downloaded archive and enter `ant rundemo`. This will compile and run the TestClient class; successfully running this example indicates that you have properly installed and configured the caCORE client. The following is a short segment of code from the TestClient class along with an explanation of its functioning.

```
ApplicationService appService = ApplicationServiceProvider.getApplicationService();
Gene gene = new Gene();
gene.setSymbol("brca*");  // searching for all genes whose symbols start with brca
try {
   List resultList = appService.search(Gene.class, gene);
   for (Iterator resultsIterator = resultList.iterator(); resultsIterator.hasNext();)
   {
      Gene returnedGene = (Gene) resultsIterator.next();
      System.out.println("Symbol: " + returnedGene.getSymbol() +
         "\tName " + returnedGene.getFullName()) +
         "\tTaxon:" + returnedGene.getTaxon().getScientificName();
      }
} catch (Exception e) {
     e.printStackTrace();
}
```

This code snippet creates an instance of a class that implements the ApplicationService interface. This interface defines the service methods used to access data objects. A criterion object is then created that defines the attribute values for which to search. The search method of the ApplicationService implementation is called with parameters that indicate the type of objects to return, `Gene.class`, and the criteria that returned objects must meet, defined by the gene object. The search method returns objects in a List collection, which is iterated through to print some basic information about the objects.

Although this is a fairly simple example of the use of the Java API, a similar sequence can be followed with more complex criteria to perform sophisticated manipulation of the data provided by caCORE. Additional information and examples are provided in the sections that follow.

Service Methods

The methods that provide programmatic access to running the caCORE server are located in the `gov.nih.nci.system.applicationservice` package. The `ApplicationServiceProvider` class uses the factory design pattern to return an implementation of the ApplicationService interface. The provider class determines whether there is a locally running instance of the caCORE system or whether it should use a remote instance. The returned ApplicationService implementation exposes the service methods that enable read/write operations on the domain objects. (Note that, although the infrastructure is capable of write operations, this functionality has been disabled for caCORE because it is primarily meant as a read-only data system.)

The separation of the service methods from the domain classes is an important architectural decision that insulates the domain object space from the underlying service framework. As a result, new business methods can be added without needing to update any of the domain model or the associated metadata information from the object model. (This is critical for ensuring semantic interoperability over multiple iterations of architectural changes.)

Within the ApplicationService implementation, a variety of methods are provided allowing users to query data based on the specific needs and types of queries to be performed. In general, there are five types of searches:

- **Simple searches** are those that take one or more objects from the domain models as inputs and return a collection of objects from the data repositories that meet the criteria specified by the input objects.
- **Nested searches** also take domain objects as inputs but determine the type of objects in the result set by traversing a known path of associations from the domain model.
- **Detached criteria searches** use Hibernate detached criteria objects to provide a greater level of control over the results of a search (such as boolean operations, ranges of values, etc.)
- **HQL searches** provide the ability to use the Hibernate Query Language for the greatest flexibility in forming search criteria.
- **SDK Query Object criteria searches** were modeled similar to the Object representation of caBIG Query Language (CQL). The SDK Query Object criteria searches use a syntax similar to the query by example (QBE) to specify the way results are to be retrieved. The system formulates the query based on the navigation path specified in the query search criteria. The query mechanism allows the user to search for the objects in platform independent query syntax.

Method Signature	`List search(`
	`Class targetClass,`
	`Object obj)`

Search Type	Simple (One criteria object)
Description	Returns a List collection containing objects of type targetClass that conform to the criteria defined by obj
Example	`search(Gene.class, gene)`

Method Signature	`List search(`
	`Class targetClass,`
	`List objList)`

Search Type	Simple (Criteria object collection)
Description	Returns a List collection containing objects of type targetClass that conform to the criteria defined by a collection of objects in objList. The returned objects must meet ANY criteria in objList (i.e. a logical OR is performed).
Example	`search(Gene.class, geneCollection)`

Method Signature	`List search(`
	`String path,`
	`Object obj)`

Search Type	Nested
Description	Returns a List collection containing objects conforming to the criteria defined by obj and whose resulting objects are of the type reached by traversing the node graph specified by path
Example	`search("gov.nih.nci.cabio.domain.Protein, gov.nih.nci.cabio.domain.Gene", nucleicAcidSequence)`

Method Signature	`List search(`
	`String path,`
	`List objList)`

Search Type	Nested
Description	Returns a List collection containing objects conforming to the criteria defined by the objects in objList and whose resulting objects are of the type reached by traversing the node graph specified by path
Example	`search("gov.nih.nci.cabio.domain.Protein, gov.nih.nci.cabio.domain.Gene", sequenceList)`

Method Signature	`List query(`
	`DetachedCriteria detachedCriteria,`
	`String targetClassName)`

Search Type	Detached criteria
Description	Returns a List collection conforming to the criteria specified by detachedCriteria and whose resulting objects are of the type specified by targetClassName
Example	`query(criteria, "gov.nih.nci.cabio.domain.Gene")`

Method Signature	`List query(`
	`Object criteria,`
	`int firstRow,`
	`int resultsPerQuery,`
	`String targetClassName)`

Search Type	Detached criteria
Description	Identical to the previous query method, but allows for control over the size of the result set by specifying the row number of the first row and the maximum number of objects to return
Example	`query(criteria, 101, 100, targetClassName)`

Method Signature	`List query(`
	`HQLCriteria hqlCriteria,`
	`String targetClassName)`
	
	`Search Type HQL`

Description	Returns a List collection of objects of the type specified by targetClassName that conform to the query in HQL syntax contained in hqlCriteria

Example	`query(hqlCriteria,`
	`"gov.nih.nci.cabio.domain.Gene")`

Method Signature	`List query(`
	`CQLQuery cqlQuery,`
	`String targetClassName)`
	
	`Search Type CQL`

Description	Returns a List collection of objects of the type specified by targetClassName that conform to the query in SDK Query Object syntax contained in cqlQuery
Example	`query(cqlQuery, "gov.nih.nci.cabio.domain.Gene")`

In addition to the data access methods, the following helper methods are available via the ApplicationService class that provide flexibility in controlling queries and result sets.

Method Signature	`int getQueryRowCount(`
	`   Object criteria,`
	`   String targetClassName`

Description	Gets the number of objects in a given query. This is useful in determining the size of the data before querying for data

Example	`getQueryRowCount(criteria,`
	`"gov.nih.nci.cabio.domain.Gene")`

Method Signature	`void setRecordsCount(int recordCount)`
Description	Allows users to set the maximum number of records returned by the search or query methods
Example	`setRecordsCount(250)`

Method Signature	`void setSearchCaseSensitivity(`
	`     boolean caseSensitivity)`

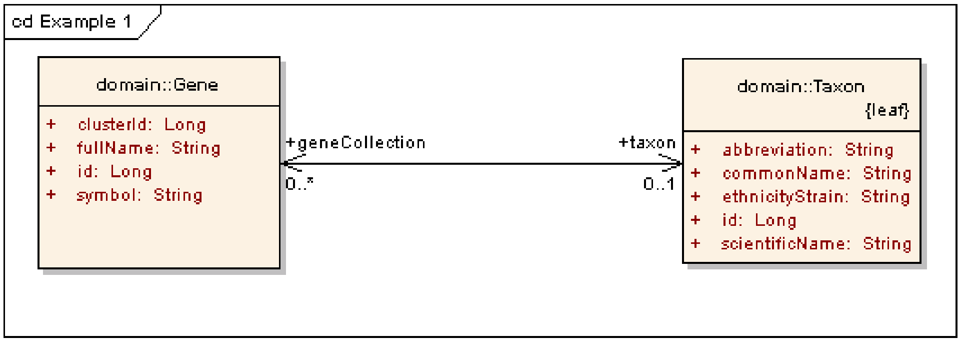
Description	Allows users to enable/disable case sensitivity for search criteria containing strings
Example	`setSearchCaseSensitivity(true)`

Examples of Use

This section includes a number of examples that demonstrate the use of the caCORE APIs. Included with each example is a brief description of the type of search being performed, a UML diagram depicting the domain objects used, and the example code accompanied by explanatory text.

Example One: Simple Search (Single Criteria Object)

In this example, a search is performed for all genes whose symbols start with 'brca'. The code iterates through the returned objects and prints out the symbol and name of each object along with the name of an associated object of type Taxon. The fetch of the associated Taxon object is done in the background and is completely transparent to the user.



```
 1  ApplicationService appService = ApplicationServiceProvider.getApplicationService();
 2  Gene gene = new Gene();
 3  gene.setSymbol("brca*");  // searching for all genes whose symbols start with ‘brca’
 4  try
 5  {
 6     List resultList = appService.search(Gene.class, gene);
 7     for (Iterator resultsIterator = resultList.iterator(); resultsIterator.hasNext();)
 8     {
 9        Gene returnedGene = (Gene) resultsIterator.next();
10        System.out.println("Symbol: " + returnedGene.getSymbol() +
11           "\tName " + returnedGene.getFullName()) +
12           "\tTaxon:" + returnedGene.getTaxon().getScientificName();
13        }
14  } catch (Exception e) {
15       e.printStackTrace();
16  }
```

Lines	***Description***
1	Creates an instance of a class that implements the ApplicationService interface; this interface defines the service methods used to access data objects.
2-3	Creates a criterion object that defines the attribute values for which to query.
6	Calls the search method of the ApplicationService implementation and passes it the type of objects to return, Gene.class, and the criteria that returned objects must meet, defined by the gene object; the search method returns objects in a List collection.
9	Casts an object from the result List and creates a variable reference to it of type Gene.
10	Prints the symbol attribute.
11	Prints the fullName attribute.

Lines	Description
12	Fetches an associated object of type Taxon and prints its scientificName attribute.

Example Two: Simple Search (Criteria Object Collection)

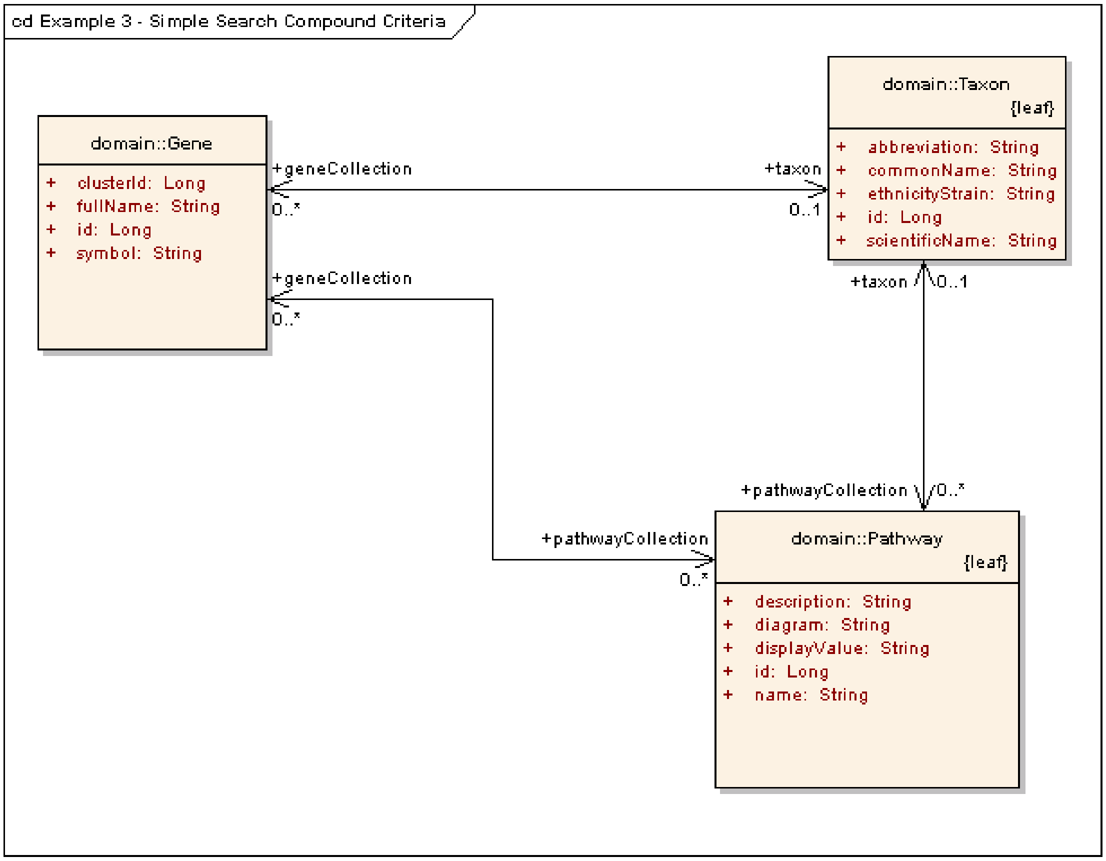
This example demonstrates a search with multiple criteria objects that are passed to the search() method. The result set will include all objects of the specified type that match ANY of the criteria objects. In this case, the search will return all objects of type Gene that are associated with Taxon objects whose abbreviation attribute is either "hs" or "ms". In biological terms, this search will return all human and mouse genes.

```
 1  Taxon taxon1 = new Taxon();
 2  taxon1.setAbbreviation("hs");       // Homo sapiens
 3  Taxon taxon2 = new Taxon();
 4  taxon2.setAbbreviation("m");       // Mus musculus
 5  List taxonList = new ArrayList();
 6  taxonList.add(taxon1);
 7  taxonList.add(taxon2);
 8  try
 9  {
10     List resultList = appService.search(Gene.class, taxonList);
11     for (Iterator resultsIterator = resultList.iterator(); resultsIterator.hasNext();) 12
13        Gene returnedGene = (Gene)resultsIterator.next();
14        System.out.println(“Symbol: “ + returnedGene.getSymbol() +
15           “\tName: ” + returnedGene.getFullName());
16     }
17  } catch (Exception e) {
18     e.printStackTrace();
19  }
```

Lines	Description
1-4	Creates two Taxon objects describing the search criteria.
5-7	Creates a List collection containing the two Taxon objects.
10	Searches for all Gene objects where the associated Taxon object matches ANY of the objects found in the taxonList collection.

Example Three: Simple Search (Compound Criteria Object)

In this example, the object that is passed to the search() method contains criteria values that are found in associated objects and collections of objects. This query will return those objects that match all of the criteria in the compound object. Note the distinction between this type of search and the previous example in which a collection of objects is passed into the search method. In the last example, the results will match ANY of the criteria objects. In this example, however, where a single compound object is used, ALL criteria are matched. In biological terms, this search will return all pathways associated with the human Interleukin 5 gene.



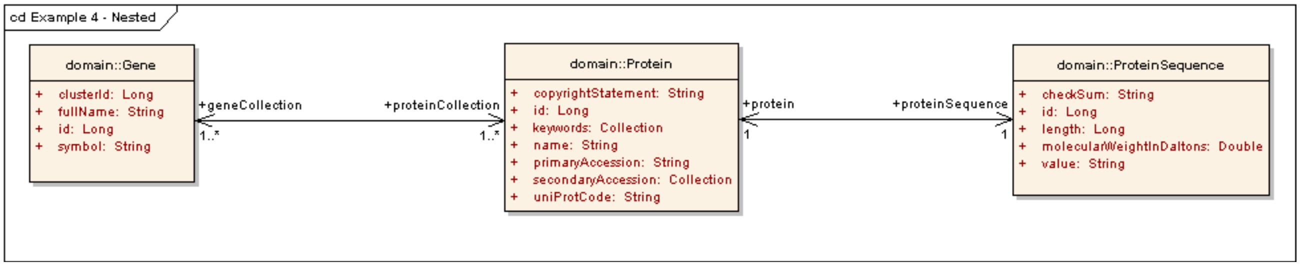
```
1   Taxon taxon = new Taxon();
2   taxon.setAbbreviation("hs");         // Homo sapiens
3   Gene gene = new Gene();
4   gene.setTaxon(taxon);
5   gene.setSymbol("IL5");               // Interleukin 5
6   List geneList = new ArrayList();
7   geneList.add(gene);
8   Pathway pathway = new Pathway();
9   pathway.setGeneCollection(geneList);
10  try
11  {
12     List resultList = appService.search("gov.nih.nci.cabio.domain.Pathway", pathway);
13     for (Iterator resultsIterator = resultList.iterator(); resultsIterator.hasNext();) 14
15        Pathway returnedPathway = (Pathway)resultsIterator.next();
16        System.out.println(returnedPathway.getDisplayValue());
17     }
18  } catch (Exception e) {
19     e.printStackTrace();
20  }
```

Lines	Description
1-5	Creates a Gene object and sets the symbol to "IL5" and the associated Taxon to an object whose abbreviation is set to "hs".
6-7	Because the Pathway and Gene classes are related by a many-to-many association, it is necessary to create a collection to contain the Gene object that will act as part of the compound criteria; multiple Gene objects could be added to this collection as needed.

Lines	Description
8-9	Creates a Pathway object and sets the value of its geneCollection to the geneList object just created.
12	Searches for all Pathway objects whose geneCollection contains objects that match the set criteria (i.e. the symbol is "IL5" and the associated Taxon objects' abbreviations are set to "hs").

Example Four: Nested Search

A nested search is one where a traversal of more than one class-class association is required to obtain a set of result objects given the criteria object. This example demonstrates one such search in which the criteria object passed to the search method is of type *Gene*, and the desired objects are of type *ProteinSequence*. Because there is no direct association between these two classes, the path of the traversal is passed to the search method enabling the query to be performed.



```
 1  Gene gene = new Gene();
 2  gene.setSymbol("TP53");             // Tumor protein p53 (Li-Fraumeni syndrome)
 3  try
 4  {
 5     List resultList = appService.search(
 6        "gov.nih.nci.cabio.domain.ProteinSequence,gov.nih.nci.cabio.domain.Protein”,
 7        gene);
 8     for (Iterator resultsIterator = resultList.iterator(); resultsIterator.hasNext();)
 9     {
10        ProteinSequence returnedProtSeq = (ProteinSequence)resultsIterator.next();
11        System.out.println(“ID: “ + returnedProtSeq.getId() +
12                           “Length: “ + returnedProtSeq.getLength() );
13     }
14  } catch (Exception e) {
15     e.printStackTrace();
16  }
```

Lines	Description
1-2	Creates a Gene object and sets the symbol to "TP53".
6	Defines search path as traversing from the criteria object of type Gene through Protein to ProteinSequence; note that the first element in the path is the desired class of objects to be returned, and that subsequent elements traverse back to the criteria object.
7	Sets the criteria object to the previously-created Gene.

Example Five: Detached Criteria Search

This example demonstrates the use of Hibernate detached criteria objects to formulate and perform more sophisticated searches (*Figure 3.2*). A detailed description of detached criteria is beyond the scope of this document; for more information, please consult the Hibernate documentation at http://www.hibernate.org/hib_docs/v3/api/org/hibernate/criterion/DetachedCriteria.html

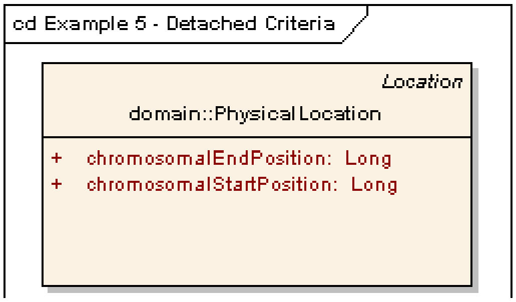

cd Example 5 - Detached Criteria
Location
domain::PhysicalLocation
+ chromosomalEndPosition: Long
+ chromosomalStartPosition: Long

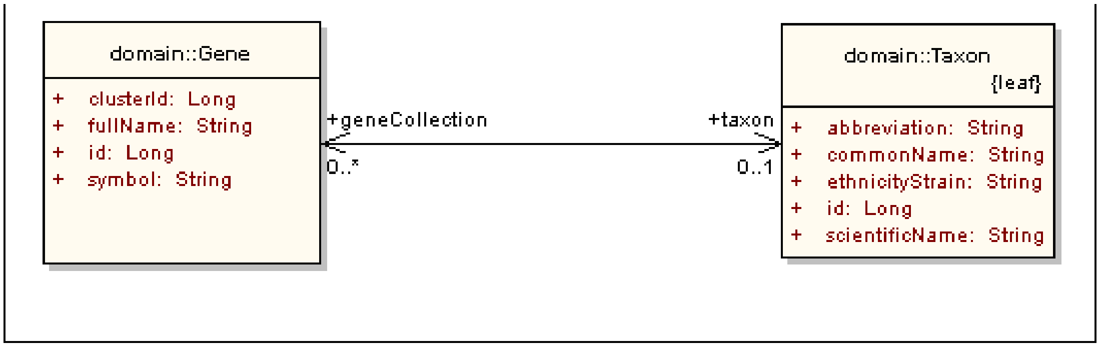
Figure 3.2 Example Five: Detached Criteria Search

```
1  DetachedCriteria criteria = DetachedCriteria.forClass(PhysicalLocation.class);
2  criteria.add(Restrictions.gt("chromosomalStartPosition", new Long(86851632)));
3  criteria.add(Restrictions.lt("chromosomalEndPosition",   new Long(86861632)));
4  List resultList = appService.query(criteria,PhysicalLocation.class.getName());
```

Lines	*Description*
1	Creates an DetachedCriteria object and sets the class on which the criteria will operate to PhysicalLocation.
2	Sets a restriction on the objects that states that the attribute chromosomalStartPosition must be greater than ("gt") the value 86851632.
3	Sets a restriction on the objects that states that the attribute chromosomalEndPosition must be less than ("lt") the value 86861632.
4	Calls the query method of the ApplicationService implementation, specifying the desired object type to return, PhysicalLocation, and passing the detached criteria object.

Example Six: HQL Search

In this example, a search is performed for all genes whose symbols start with 'brca'. This is identical to Example One but uses a Hibernate Query Language (HQL) search string to form the query. For more information on HQL syntax, consult the Hibernate documentation at http://www.hibernate.org/hib_docs/v3/reference/en/html/query-hql.html.



```
1  String hqlString = “FROM Gene g WHERE g.symbol LIKE ‘BRCA%’”;
2  HQLCriteria hqlC = new HQLCriteria(hqlString);
3  List resultList = appService.query(hqlC, Gene.class.getName());
```

Lines	***Description***
1	Creates a string that contains the query in HQL syntax.
2	Instantiates an HQLCriteria object and sets the query string.
3	Calls the query method of the ApplicationService implementation and passes it the HQLCriteria object and the type of objects to return.

Example Seven: SDK Query Object Search

SDK Query Object Example One

```
1     ApplicationService appService =
                  ApplicationServiceProvider.getApplicationService();

2     CQLQuery cqlQuery = new CQLQuery();
3     CQLObject target = new CQLObject();  //Create Gene Object
4     target.setName("gov.nih.nci.cabio.domain.Gene");
5     CQLAttribute attribute = new CQLAttribute();
6     attribute.setName("symbol");
7     attribute.setValue("%il%");
8     attribute.setPredicate(CQLPredicate.LIKE);
9     target.setAttribute(attribute);

10    cqlQuery.setTarget(target);

11    try {
12          List resultList = appService.query(cqlQuery,
13                                  "gov.nih.nci.cabio.domain.Gene");
14          //Iterate through retrieved list of objects
15          for (Iterator resultsIterator = resultList.iterator();
16                                        resultsIterator.hasNext();) {
17                Gene returnedGene = (Gene) resultsIterator.next();
18                System.out.println(
19                      "Symbol: " + returnedGene.getSymbol() + "\n" +
20                      "\tTaxon:" +
21                      returnedGene.getTaxon().getScientificName() +
22                      "\n" +"\tName " + returnedGene.getFullName() +
23                      "\n");
24          }
25    } catch (Exception e) {
26          e.printStackTrace();
27    }
```

Lines	***Description***
2	Creates a new CQLQuery Object.
3-4	Creates a new search Target object (Gene).
5-8	Creates a new attribute for the search target object whose value should be compared with the value in the database for given predicate.
10	Sets the search target object in SDK Query Object Query.
12-13	Sets the search criteria object to the previously-created CQLQuery.

SDK Query Object Example Two

```
1     ApplicationService appService =
                  ApplicationServiceProvider.getApplicationService();

2     CQLQuery cqlQuery = new CQLQuery();
3     CQLObject target = new CQLObject();
```

```
4     target.setName("gov.nih.nci.cabio.domain.Gene");

5     //Create Taxon Object Query
6     CQLAssociation association1 = new CQLAssociation();
7     association1.setName("gov.nih.nci.cabio.domain.Taxon");
8     CQLAttribute attribute1 = new CQLAttribute();
9     attribute1.setName("abbreviation");
10    attribute1.setValue("%hs%");
11    attribute1.setPredicate(CQLPredicate.LIKE);
12    association1.setAttribute(attribute1);

13    //Create Taxon Object Query
14    CQLAssociation association2 = new CQLAssociation();
15    association2.setName("gov.nih.nci.cabio.domain.Taxon");
16    CQLAttribute attribute2 = new CQLAttribute();
17    attribute2.setName("abbreviation");
18    attribute2.setValue("%m%");
19    attribute2.setPredicate(CQLPredicate.LIKE);
20    association2.setAttribute(attribute2);

21    //Create Group(Collection) of Taxon Object Query
22    CQLGroup group = new CQLGroup();
23    group.addAssociation(association1);
24    group.addAssociation(association2);
25    group.setLogicOperator(CQLLogicalOperator.OR);
26    target.setGroup(group);

27    cqlQuery.setTarget(target);

28    try {
29          List resultList = appService.query(cqlQuery,
30                                      "gov.nih.nci.cabio.domain.Gene");
31          //Iterate through retrieved list of objects
32          for (Iterator resultsIterator = resultList.iterator();
33                                            resultsIterator.hasNext();)
34                Gene returnedGene = (Gene) resultsIterator.next();
35                System.out.println(
36                      "Symbol: " + returnedGene.getSymbol() + "\n" +
37                      "\tTaxon:" +
38                      returnedGene.getTaxon().getScientificName() +
39                      "\n" +"\tName " + returnedGene.getFullName() +
40                      "\n");
41          }
42    } catch (Exception e) {
43          e.printStackTrace();
44    }
```

Lines	***Description***
2	Creates a new CQLQuery Object.
3-4	Creates a new search Target object (Gene).
5-12, 13-20	Creates a new associated object (Taxon) whose attribute value should be compared with the value in the database for a given predicate.
21-26	Creates a group (collection) of objects and associates it to the search target object.

Lines	Description
27	Sets the search target object in SDK Query Object Query.
29-30	Sets the search criteria object to the previously-created CQLQuery.

SDK Query Object Example Three

```
1     ApplicationService appService =
                  ApplicationServiceProvider.getApplicationService();

2     CQLQuery cqlQuery = new CQLQuery();
3     CQLObject target = new CQLObject();
4     target.setName("gov.nih.nci.cabio.domain.Pathway");

5     //Create Gene Object Query
6     CQLAssociation association1 = new CQLAssociation();
7     association1.setName("gov.nih.nci.cabio.domain.Gene");
8     CQLAttribute attribute1 = new CQLAttribute();
9     attribute1.setName("symbol");
10    attribute1.setValue("IL5");
11    attribute1.setPredicate(CQLPredicate.EQUAL_TO);
12    association1.setAttribute(attribute1);

13    //Create Taxon Object Query
14    CQLAssociation association2 = new CQLAssociation();
15    association2.setName("gov.nih.nci.cabio.domain.Taxon");
16    CQLAttribute attribute2 = new CQLAttribute();
17    attribute2.setName("abbreviation");
18    attribute2.setValue("%hs%");
19    attribute2.setPredicate(CQLPredicate.LIKE);
20    association2.setAttribute(attribute2);

21    //Set Relatonship between Gene and Taxon
22    association1.setAssociation(association2);

23    //Set Relatonship between Pathway and Gene
24    //Role name is required as it can not be determined using
25    //reflection
26    association1.setTargetRoleName("geneCollection");
27    target.setAssociation(association1);

28    cqlQuery.setTarget(target);

29    try
30    {
31          List resultList = appService.query(cqlQuery,
32                                  "gov.nih.nci.cabio.domain.Pathway");
33          //Iterate through retrieved list of objects
34          for (Iterator resultsIterator = resultList.iterator();
35                                              resultsIterator.hasNext();)
36          {
37                Pathway returnedPathway =
38                                  (Pathway)resultsIterator.next();
39                System.out.println(
40                            returnedPathway.getDisplayValue());
41          }
42    } catch (Exception e) {
43          e.printStackTrace();
44    }
```

Lines	Description
2	Creates a new CQLQuery Object.
3-4	Creates a new search Target object (Pathway).
5-12	Creates a new associated object (Gene) whose attribute value should be compared with the value in the database for a given predicate.
13-20	Creates a new associated object (Taxon) whose attribute value should be compared with the value in the database for a given predicate.
22	Creates a relationship between two objects (Gene and Taxon).
26-27	Creates a relationship between a target object (Pathway) and its associated object (Gene). The role name is the name of the association used to retrieve the associated object. It is required in case it can not be determined by reflection.
27	Sets the search target object in a SDK Query Object Query.
31-32	Sets the search criteria object to the previously-created CQLQuery.

SDK Query Object Example Four

```
1     ApplicationService appService =
                  ApplicationServiceProvider.getApplicationService();

2     CQLQuery cqlQuery = new CQLQuery();
3     CQLObject target = new CQLObject();
4     target.setName("gov.nih.nci.cabio.domain.ProteinSequence");

5     //Create Gene Object Query
6     CQLAssociation association1 = new CQLAssociation();
7     association1.setName("gov.nih.nci.cabio.domain.Gene");
8     CQLAttribute attribute1 = new CQLAttribute();
9     attribute1.setName("symbol");
10    attribute1.setValue("TP53");
11    attribute1.setPredicate(CQLPredicate.EQUAL_TO);
12    association1.setAttribute(attribute1);

13    //Create Protein Object Query
14    CQLAssociation association2 = new CQLAssociation();
15    association2.setName("gov.nih.nci.cabio.domain.Protein");

16    //Set Relatonship between Gene and Protein
17    //Role name is required as it can not be determined using
18    //reflection
19    association1.setTargetRoleName("geneCollection");
20    association2.setAssociation(association1);

21    //Set Relatonship between Protein and ProteinSequence
22    //Example of using relationship from target to source
```

```
24    association1.setSourceRoleName("proteinSequence");
25    target.setAssociation(association2);

26    cqlQuery.setTarget(target);


27    try
28    {
29          List resultList = appService.query(cqlQuery,
30                      "gov.nih.nci.cabio.domain.ProteinSequence");
31          //Iterate through retrieved list of objects
32          for (Iterator resultsIterator = resultList.iterator();
33                                        resultsIterator.hasNext();)
34          {
35                ProteinSequence returnedProtSeq =
36                            (ProteinSequence)resultsIterator.next();
37                System.out.println("ID: " + returnedProtSeq.getId()
38                      "Length: " + returnedProtSeq.getLength());
                }
40    } catch (Exception e) {
41          e.printStackTrace();
42    }
```

Lines	***Description***
2	Creates a new CQLQuery Object.
3-4	Creates a new search Target object (ProteinSequence).
5-12	Creates a new associated object (Gene) whose attribute value should be compared with the value in the database for a given predicate.
14-15	Creates a new associated object (Protein).
19-20	Creates a relationship between a source object (Protein) and its associated object (Gene). The role name is the name of the association used to retrieve the associated object. It is required in case it can not be determined by reflection.
24-25	Creates a relationship between a target object (ProteinSequence) and its associated object (Protein).
26	Sets the search target object in a SDK Query Object Query.
29-30	Sets the search criteria object to the previously-created CQLQuery.

EVS Service Methods

The service methods described above apply to all object models built using the caCORE infrastructure. In addition to these, the caCORE system hosted at the NCICB provides a special data access method designed to simplify searching the terminologies available from the Enterprise Vocabulary Services (EVS).

Method Signature	`List evsSearch(EVSQuery evsCriterion)`
Search Type	EVS

Description	Returns a List collection containing objects conforming to the criteria defined by evsCriterion
Example	`evsSearch(evsCriterion1)`

The EVSQuery class provides a number of methods that can be used to define the search parameters. For more information about this class, see the *caCORE EVS API* on page 62.

The following code demonstrates use of the EVS service method.

```
 1  ApplicationService appService = ApplicationServiceProvider.getApplicationService();
 2  try
 3  {
 4      String vocabularyName = "NCI_Thesaurus";
 5      EVSQuery query = new EVSQuery();
 6      evsQuery.getHistoryRecords(vocabularyName, "C16612");
 7
 8      List evsResults = new ArrayList();
 9      evsResults = (List)appService.evsSearch(evsQuery);
10
11      System.out.println(evsResults.size() + " records found");
12      for (Iterator resultsIterator = evsResults.iterator(); resultsIterator.hasNext();)
13      {
14             HistoryRecord historyRecord = (HistoryRecord)iter.next();
15             // Additional processing logic
16      }
17  } catch (Exception ex) {
18      ex.printStaceTrace();
19  }
```

Lines	***Description***
1	The EVS service method, like all other Java service methods, uses an implementation of the ApplicationService interface.
4	EVS provides several vocabularies; most searches require that the name of the desired vocabulary is specified.
5-6	Creates a new EVSQuery object and specifies that the desired search is for HistoryRecord objects relating to the concept with code "C16612" from the "NCI_Thesaurus" vocabulary.
9	Calls the evsSearch method of the ApplicationService implementation passing the EVSQuery object.
13	The type of object that is returned (and consequently can be expected to populate the resulting List collection) depends on the search parameters set in the EVSQuery object; in this case, because the getHistoryRecords() method was invoked, the resulting objects are of type HistoryRecord.

Utility Methods

XML Utility

caCORE provides a utility (*XMLUtility* class) in the `gov.nih.nci.common.util` package that provides the capability of converting caCORE domain objects between native Java objects and XML serializations based on standard XML schemas. The XML schemas for all domain objects in caCORE, directly generated from the UML model, are included in the downloadable archive (in the lib directory). Currently, the XML generated using the *XMLUtility* class includes only the object attributes; associated objects are not included.

Properties used by the XML utility are included in two files. The first, `xml.properties`, defines some basic information needed by the class and also contains a property defining the filename of the second. This second file, called `xml-mapping.xml` by default, defines the binding between class and attribute names and the corresponding XML element and attribute names.

A default marshaller and unmarshaller are provided with the caCORE client; developers wishing to use their own should provide the fully-qualified name of the two classes in the `xml.properties` file.

In the following code, the XML utility is used to serialize an object and save it to a file stream. A new object is then instantiated from the file using the utility.

```
1  // Assume an object of type Gene called myGene
2  File myFile = new File("myGene.xml");
3  FileWriter myWriter = new FileWriter(myFile);
4  XMLUtility myUtil = new XMLUtility();
5  myUtil.toXML(myGene,myWriter);
6  Gene myNewGene = (Gene)myUtil.fromXML(myFile);
7  bool isSameGene = myNewGene.equals(myGene);      // true
```

Lines	***Description***
2-3	Creates a new file stream where the XML serialization will be saved.
4	Creates a new XMLUtility object; in this case, the default marshaller and unmarshaller will be used.
5	Serializes the myGene object to XML using the mapping file and writes the output to the file stream myGene.xml.
6	Creates a new object called myNewGene by invoking the fromXML() method of the XMLUtility class and casting it to the proper type.
7	The newly created Gene object is equivalent to the old one.

For additional details, consult the caCORE JavaDocs at http://ncicb.nci.nih.gov/NCICB/content/ncicblfs/caCORE3-2_JavaDocs.

SVG Manipulation Utility

caCORE includes a class (*SVGManipulator*) in the `gov.nih.nci.common.util` package that provides useful services to manipulate Scalable Vector Graphics (SVG) diagrams retrieved from the caBIO Pathway domain object. For more information on how to use this utility, see *Manipulating SVG Diagrams* on page 117.

Web Services API

The caCORE Web services API allows access to caCORE data from development environments where the Java API cannot be used, or where use of XML Web services is more desirable. This includes non-Java platforms and languages such as Perl, C/C++,.NET Framework (C#, VB.Net), Python, etc.

The Web services interface can be used in any language-specific application that provides a mechanism for consuming XML Web services based on the Simple Object Access Protocol (SOAP). In these environments, connecting to caCORE can be as simple as providing the endpoint URL. Some platforms and languages require additional client-side code to handle the implementation of the SOAP envelope and the resolution of SOAP types. A list of packages catering to different programming languages is available at http://www.w3.org/TR/SOAP/ and at http://www.soapware.org/.

To maximize standards-based interoperability, the caCORE Web service conforms to the Web Services Interoperability Organization (WS-I) Basic Profile. The WS-I Basic Profile provides a set of non-proprietary specifications and implementation guidelines enabling interoperability between diverse systems. More information about WS-I compliance is available at http://www.ws-i.org.

On the server side, Apache Axis is used to provide SOAP-based inter-application communication. Axis provides the appropriate serialization and deserialization methods for the Java beans to achieve an application-independent interface. For more information about Axis, visit http://ws.apache.org/axis/.

Configuration

The caCORE WSDL file is located at http://cabio.nci.nih.gov/cacore32/ws/caCOREService?wsdl.

In addition to describing the protocols, ports and operations exposed by the caCORE Web service, this file can be used by a number of IDEs and tools to generate stubs for caCORE objects. This allows code on different platforms to instantiate objects native to each for use as parameters and return values for the Web service methods. Consult the specific documentation for your platform for more information on how to use the WSDL file to generate class stubs.

The caCORE Web services interface has a single endpoint called caCOREService, which is located at http://cabio.nci.nih.gov/cacore32/ws/caCOREService. Client applications should use this URL to invoke Web service methods.

Operations

Through the caCOREService endpoint, developers have access to three operations:

Operation	`getVersion`
Input Schema	`none`

Output Schema	`<complexType>`
	`    <sequence>`
	`        <element`
	`type="xsd:string"/>`
	`     </sequence>`
	`</complexType>`

Description	Returns an xsd:string containing the version of the running caCORE system (e.g., "caCORE 3.2")

Operation	queryObject

Input Schema	<complexType>
	    <sequence>
	         <element name="in0"
	type="xsd:string"/>
	     <element name="in1"
	type="xsd:anyType"/>
	      </sequence>
	</complexType>

Output Schema	<sequence> <element name="queryReturn"type= "ArrayOf_xsd_anyType"/> </sequence>
Description	Performs a search for objects conforming to the criteria defined by input parameter in1 and whose resulting objects are of the type reached by traversing the node graph specified by parameter in0; the result is a set of serialized objects (the type ArrayOf_xsd_anyType resolves to a sequence of xsd:anyType elements)

Operation	query

Input Schema	<complexType>
	   <sequence>
	      <element name="in0" type="xsd:string"/>
	      <element name="in1" type="xsd:anyType"/>
	      <element name="in2" type="xsd:int"/>
	      <element name="in3" type="xsd:int"/>
	   </sequence>
	</complexType>

Output Schema	<sequence>
	  <element name="queryReturn"
	type="ArrayOf_xsd_anyType"/>
	</sequence>

Description	Identical to the previous queryObject method, but allows for control over the result set by specifying the row number of the first row (in2) and the maximum number of objects to return (in3)

Developers should be aware of a significant behavioral decision that has been made regarding the Web services interface. When a query is performed with this interface, returned objects do not contain or refer to their associated objects (a notable exception is with the EVS domain model-see below). This means that a separate query invocation must be performed for each set of associated objects that need to be retrieved. One of the examples below demonstrates this functionality.

EVS Considerations

This section briefly describes specific aspects of the caCORE Web services behavior that relate to the EVS domain model. For a detailed description of EVS classes and the underlying data model, see the *caCORE EVS API* on page 62.

EVS consists of two main vocabularies, the NCI Thesaurus and the NCI MetaThesaurus. Web service queries can be performed on these vocabularies to get objects of type

DescLogicConcept from the NCI Thesaurus or of type *MetaThesaurusConcept* from the NCI MetaThesaurus. (A third EVS object class, *HistoryRecord*, can also be returned by the Web services interface.) Most other objects in the EVS model are accessible because they are properties of these two main classes.

The EVS domain objects are unique in this way when using the Web services interface. Whereas objects in other domains do not return associations, the three EVS classes that can be queried from Web services always provide associations to their related objects. This enables access to the objects that are not of type *DescLogicConcept*, *MetaThesaurusConcept* or *HistoryRecord*.

Because of the unique behavior and properties of the EVS domain model, queries using the Web services interface can be performed only on the selected attribute values listed in Table *3.3*.

Class	***Available search attributes***
DescLogicConcept	name
	code
	Property name and value
	Role name and value
MetaThesaurus Concept	name
	cui (concept unique identifier)
	Atom code and Source abbreviation
HistoryRecord	DescLogicConcept name or code (HistoryRecord is the targetObject and the DescLogicConcept is the criteriaObject)

Table 3.3 Allowable attributes for searching the EVS domain model

Examples of Use

Example One: Simple Search

The following code demonstrates a simple query written in the Java language that uses the Web services API. This example uses Apache Axis on the client side to handle the type mapping, SOAP encoding, and operation invocation.

```
 1  Service  service = new Service();
 2  Call call = (Call) service.createCall();
 3
 4  QName qnGene = new QName("urn:ws.domain.cabio.nci.nih.gov", "Gene");
 5  call.registerTypeMapping(
 6     Gene.class,
 7     qnGene,
 8     new org.apache.axis.encoding.ser.BeanSerializerFactory(Gene.class, qnGene),
 9     new org.apache.axis.encoding.ser.BeanDeserializerFactory(Gene.class, qnGene)
10     );
11
12  String url = "http://cabio.nci.nih.gov/cacore32/ws/caCOREService";
13
14  call.setTargetEndpointAddress(new java.net.URL(url));
15  call.setOperationName(new QName("caCOREService", "queryObject"));
16  call.addParameter("arg1", org.apache.axis.encoding.XMLType.XSD_STRING, ParameterMode.IN);
17  call.addParameter("arg2", org.apache.axis.encoding.XMLType.XSD_ANYTYPE, ParameterMode.IN);
18  call.setReturnType(org.apache.axis.encoding.XMLType.SOAP_ARRAY);
19
20  gov.nih.nci.cabio.domain.ws.Gene gene = new gov.nih.nci.cabio.domain.ws.Gene();
21  gene.setSymbol("IL*");
22
23  try
24  {
25     Object[] resultList = (Object[])call.invoke(
26        new Object[]{"gov.nih.nci.cabio.domain.ws.Gene", gene });
27     System.out.println("Total objects found: " + resultList.length);
28     if (resultList.length > 0)
29     {
30        for(int resultIndex = 0; resultIndex < resultList.length; resultIndex++)
31        {
32           Gene returnedGene = (Gene)resultList[resultIndex];
33           System.out.println(
34              "Symbol: " + returnedGene.getSymbol() + "\n" +
35              "\tName " + returnedGene.getFullName() + "\n" +
36              "");
37        }
38     }
39  } catch (Exception e) {
40     e.printStackTrace();
41  }
```

Lines	*Description*
1-2	Defines a new Web service call.
4-10	Maps a serialized object to its Java equivalent using the qualified name of the object from the WSDL file; in this case, the XML element Gene in the `urn:ws.domain.cabio.nci.nih.gov` namespace is mapped to the Java *Gene* class.
12	Defines the service endpoint.
14-18	Sets the call properties including the name of the operation to invoke, the input parameters that will be sent and the return type to expect.
20-21	Creates a Gene criteria object and sets its symbol attribute to `"IL*"`; note that the `*.ws.*` package is used.

Lines	Description
25-26	Invokes the Web service operation using an array of two objects (target class name and criteria object) as input parameters and expecting an object array as its result.
32	Casts each object in the result array to type Gene.

Example Two: Searching Associations

This example is similar to the previous one but demonstrates how to search for associated elements by performing additional invocations of the query or queryObject operation.

```
 1  try
 2  {
 3     Object[] resultList = (Object[])call.invoke(
 4        new Object[]{"gov.nih.nci.cabio.domain.ws.Gene", gene });
 5     System.out.println("Total objects found: " + resultList.length);
 6     if (resultList.length > 0)
 7     {
 8        for(int resultIndex = 0; resultIndex < resultList.length; resultIndex++)
 9        {
10           Gene returnedGene = (Gene)resultList[resultIndex];
11           System.out.println(
12              "Symbol: " + returnedGene.getSymbol() + "\n" +
13              "\tName: " + returnedGene.getFullName() +
14              "");
15           Object[] nestedResultList = (Object[])call.invoke(
16              new Object[]{“gov.nih.nci.cabio.domain.ws.Taxon”, gene });
17           if (nestedResultList.length > 0)
18           {
19              Taxon returnedTaxon = (Taxon)nestedResultList[0];
20              System.out.println(“\tTaxon: “ + returnedTaxon.getScientificName());
21           }
22        }
23     }
24  } catch (Exception e) {
25     e.printStackTrace();
26  }
```

Lines	Description
15-16	A second operation invocation requests objects of type Taxon based on the same gene criteria used for the original query.
19	Casts the objects resulting from the nested query as Taxon objects.

Example Three: EVS Domain Search (NCI MetaThesaurus)

The code below demonstrates use of the Web services interface to query data from the NCI MetaThesaurus using EVS domain objects. (Certain repetitive sections have been removed for brevity.)

```
Service  service = new Service();
Call call = (Call) service.createCall();

QName qnMTC = new QName("urn:ws.domain.evs.nci.nih.gov", "MetaThesaurusConcept");
call.registerTypeMapping(
   MetaThesaurusConcept.class,
   qnMTC,
   new BeanSerializerFactory(MetaThesaurusConcept.class, qnMTC),
   new BeanDeserializerFactory(MetaThesaurusConcept.class, qnMTC)
   );
QName qnMTCArray = new QName("urn:ws.domain.evs.nci.nih.gov",
   "ArrayOf_tns1_MetaThesaurusConcept");
call.registerTypeMapping(
   MetaThesaurusConcept[].class,
   qnMTCArray,
   new org.apache.axis.encoding.ser.ArraySerializerFactory(),
   new org.apache.axis.encoding.ser.ArrayDeserializerFactory()
   );
 // Similarly, define Qnames and register type mappings for all EVS objects
 // The exact method of generating these statements depends on your platform and language

String url = "http://cabio.nci.nih.gov/cacore32/ws/caCOREService";

call.setTargetEndpointAddress(new java.net.URL(url));
call.setOperationName(new QName("caCOREService", "queryObject"));
call.addParameter("arg1", org.apache.axis.encoding.XMLType.XSD_STRING, ParameterMode.IN);
call.addParameter("arg2", org.apache.axis.encoding.XMLType.XSD_ANYTYPE, ParameterMode.IN);
call.setReturnType(qnMTCArray);

MetaThesaurusConcept mtc = new MetaThesaurusConcept();
mtc.setName("blood*");

Object[] metaParams = new Object[]{"MetaThesaurusConcept",mtc};
MetaThesaurusConcept[] meta = (MetaThesaurusConcept[])call.invoke(metaParams);

if(meta.length>0){
    for(int m=0; m < meta.length; m++)
    {
           MetaThesaurusConcept concept = (MetaThesaurusConcept)meta[m];
           System.out.println("\nConcept code: " + concept.getCui() + "\n” +
                  “\t" + concept.getName());
           List sList =  concept.getSourceCollection();
           System.out.println("\tSource-->" + sList.size());
           for(int y=0; y<sList.size(); y++)
           {
                  Source s = (Source)sList.get(y);
                  System.out.println("\t - "+s.getAbbreviation());
           }
           List semanticList = concept.getSemanticTypeCollection();
           System.out.println("\tSemanticType---> count ="+ semanticList.size());
           for(int z=0; z<semanticList.size(); z++)
           {
                  SemanticType sType = (SemanticType) semanticList.get(z);
                  System.out.println("\t- Id: " + sType.getId() + "\n” +
                         “\t- Name : " + sType.getName());
           }
           List atomList = concept.getAtomCollection();
           System.out.println("\tAtoms -----> count = "+ atomList.size());
           for(int i=0;i<atomList.size(); i++)
           {
                  Atom at = (Atom)atomList.get(i);
                  System.out.println("\t -Code: "+ at.getCode() +
                         " -Name: " + at.getName() +
                         " -LUI: " + at.getLui() +
                         " -Source: " + at.getSource().getAbbreviation()
                         );
67              }
68              List synList = concept.getSynonymCollection();
69              System.out.println("\tSynonyms -----> count = "+ synList.size());
70              for(int i=0; i< synList.size(); i++){
71                     System.out.println("\t - "+ (String) synList.get(i));
72              }
73       }
74  }
```

Lines	Description
4-20	Maps the serialized EVS objects to their Java equivalents using the qualified names of the objects from the WSDL file; each EVS class should have a qualified name (QName) declaration and accompanying type mapping.
28	Declares the return type of the invocation as an array of MetaThesaurusConcept objects.
30-31	Creates a MetaThesaurusConcept criteria object and sets its name attribute to "blood*".
33-34	Invokes the Web service operation using an array of two objects (target class name and criteria object) as input parameters and expecting an object array as its result.
39	Casts each object in the result array to type MetaThesaurusConcept.
42-72	EVS objects retrieved from the Web services interface include associated objects, therefore it is possible to call the getAssociatedObject and getAssociatedObjectCollection methods; no additional Web services invocations are required.

Example Four: EVS Domain Search (NCI Thesaurus)

This example is similar to the previous one, except here the search is performed for data in the NCI Thesaurus.

```
 // As in previous example, define Qnames and register type mappings for all EVS objects

call.setReturnType(qnDLCArray);

DescLogicConcept dlc = new DescLogicConcept();
dlc.setName("blood*");

Object[] thesaurusParams = new Object[]{"DescLogicConcept",dlc};
DescLogicConcept[] dlcs = (DescLogicConcept[])call.invoke(thesaurusParams);

for(int i=0; i < dlcs.length; i++)
{
    DescLogicConcept concept = dlcs[i];
    System.out.println("\nConcept: "+ concept.getName()+"\t"+ concept.getCode());
    List pList = new ArrayList();
    pList = concept.getPropertyCollection();
    for(int x=0; x<pList.size(); x++)
    {
           Property prop = (Property)pList.get(x);
           System.out.println("\tProperty :"+ prop.getName()+"\t"+ prop.getValue());
           List qList = prop.getQualifierCollection();
           for(int q=0; q< qList.size(); q++)
           {
                  Qualifier qual = (Qualifier)qList.get(q);
                  System.out.println("\t\tQualifer " + qual.getName() +
                         "\t" + qual.getValue());
           }
    }
}
```

Limitations

- By default, the queryObject operation limits the result set to 1000 objects, even if the size of the result set is larger. To retrieve the objects past the 1000th record, you must use the query operation and specify the first object index (parameter in2) to be greater than 1000.
- Result sets serialized and returned by the Web services interface do not currently include associations to related objects. A consequence of this behavior is that nested criteria objects with one-to-many associations that are passed to the query or queryObject operations will result in an exception being thrown.

  The following code demonstrates a Web services invocation that would fail:

```
Gene gene = new Gene();
gene.setSymbol("IL*");
Pathway pathway = new Pathway();
pathway.setId(new Long(120));
List pathwayList = new ArrayList()
pathwayList.add(pathway);
gene.setPathwaycollection(pathwayList);
try
{
   Object[] resultList = (Object[])call.invoke(
      new Object[]{"gov.nih.nci.cabio.domain.ws.Gene", gene });
} catch (Exception e) {
   // Web Services Exception will be caught
}
```

- Because the Web services invocation has an inherent timeout behavior, queries which take a long time to execute may not complete. If this is the case, use the query method to specify a smaller result count.
- Access to the EVS domain model is limited by the Web services interface, as shown in the following table:

Typical Behavior	***EVS Model Behavior***
Can query for any object in the object model	Can query only for a DescLogicConcept, HistoryRecord or a MetaThesaurusConcept
The association values of the caCORE domain objects are not populated; need to run a second query to get associated values	All attributes of the result object are populated
Can perform queries on any attribute value	Queries can be performed only on selected attribute values (see Table *3.3*)

XML-HTTP API

The caCORE XML-HTTP API, based on the REST (Representational State Transfer) architectural style, provides a simple interface using the HTTP protocol. In addition to its ability to be invoked from most internet browsers, developers can use this interface to build applications that do not require any programming overhead other than an HTTP client. This is particularly useful for developing web applications using AJAX (asynchronous JavaScript and XML).

Service Location and Syntax

The caCORE XML-HTTP interface uses the following URL syntax (Table *3.4*):

```
http://{server}/{servlet}?query={returnClass}&{criteria}&
   resultCounter={counter}&startIndex={index}&
   pageSize={pageSize}&pageNumber={pageNumber}
```

Element	***Meaning***	***Required***	***Example***
server	Name of the web server on which caCORE 3.2 web application is deployed.	Yes	cabio.nci.nih.gov
servlet	URI and the name of the servlet that will accept the HTTP GET requests	Yes	cacore32/server/GetXML cacore32/server/GetHTML
returnClass	Class name indicating the type of objects that this query should return	Yes	query=Gene
criteria	Search request criteria describing the requested objects	Yes	Gene[@id=2]

Table 3.4 URL syntax used by the caCORE XML-HTTP interface

Element	Meaning	Required	Example
counter	Number of top level objects returned by the search	No	resultCounter=500
index	Start index of the result set	No	startIndex=25
pageSize	Number of records to display on each "page	No	pageSize=50
pageNumber	The number of the "page" for which to display results	No	pageNumber=3

Table 3.4 URL syntax used by the caCORE XML-HTTP interface (Continued)

The caCORE architecture currently provides two servlets that accept incoming requests:

- **GetXML** returns results in an XML format that can be parsed and consumed by most programming languages and many document authoring and management tools
- **GetHTML** presents result using a simple HTML interface that can be viewed by most modern Internet browsers

Within the request string of the URL, the criteria element specifies the search criteria using XQuery-like syntax. Within this syntax, square brackets ("[" and "]") represent attributes and associated roles of a class, the "at" symbol ("@") signals an attribute name/value pair, and a forward slash character ("/") specifies nested criteria. Criteria statements within XML-HTTP queries are generally of the following forms (although more complex statements can also be formed):

```
{ClassName}[@{attributeName}={value}]
[@{attributeName}={value}]…
{ClassName}[@{attributeName}={value}]/
{ClassName}[@{attributeName}={value}]/…
```

Parameter	Meaning	Example
ClassName	The name of a class	Gene
attributeName	The name of an attribute of the return class or an associated class	symbol
value	The value of an attribute	brca*

Figure 3.3 Criteria statements within XML-HTTP queries

Examples of Use

The following examples demonstrate use of the XML-HTTP interface. In actual use, the queries shown here would either be submitted by a block of code or entered in the address bar of an Internet browser. Also note that the servlet name *GetXML* in each of the examples can be replaced with *GetHTML* to view with layout and markup in a browser.

Query	http://server/servlet/GetXML?query=Gene&Gene[@symbol=brca*]

Syntactic Meaning	Find all objects of type Gene whose symbol starts with 'brca'.
Biological Meaning	Find all BRCA genes.

Query	http://server/servlet/GetXML?query=Gene&Gene[@symbol=brca*]/Taxon[@scientificName=homo sapiens]
Syntactic Meaning	Find all objects of type Gene whose symbol starts with 'brca' and which have an associated Taxon object whose scientificName is equal to 'homo sapiens'.
Biological Meaning	Find all human BRCA genes.

Query	http://server/servlet/GetXML? query=Tissue&Tissue[@organ=eye][@histology=neoplasia]
Syntactic Meaning	Find all objects of type Tissue associated with attribute organ equal to 'eye' and histology equal to 'neoplasia'.
Biological Meaning	Find all tissues representing neoplasms of the eye.

Query	http://server/servlet/GetXML? query=Gene&Chromosome[@number=2]/Taxon[@scientificName=homo sapiens]
Syntactic Meaning	Find all objects of type Gene associated with Chromosome objects with number equal to 2 which themselves are related to Taxon objects with scientificName equal to 'homo sapiens'.
Biological Meaning	Find all human genes on chromosome number 2

Query	http://server/servlet/GetXML? query=Gene&Chromosome[@number=2]/Taxon[@scientificName=homo sapiens]
Syntactic Meaning	Find all objects of type Gene associated with Chromosome objects with number equal to 2 which themselves are related to Taxon objects with scientificName equal to 'homo sapiens' in biological terms
Biological Meaning	Find all human genes on chromosome number 2

Working With Result Sets

Because HTTP is a stateless protocol, the caCORE server has no knowledge of the context of any incoming request. Consequently, each invocation of GetXML or GetHTML must contain all of the information necessary to retrieve the request, regardless of previous requests. Developers should consider this when working with the XML-HTTP interface.

Retrieving Related Results using XLinks

When using the GetXML servlet to retrieve results as XML, associations between objects are converted to XLinks within the XML. The link notation, shown below, allows the client to make a subsequent request to retrieve the associated objects.

```
<class name="gov.nih.nci.cabio.domain.Gene" recordNumber="1">
   …
   <field name="taxon"
      xlink:type="simple"
      xlink:href="http://cabio.nci.nih.gov/cacore31/
GetXML?query=Taxon&Gene[@id=5]">
         getTaxon
      </field>
   …
</class>
```

Controlling the Number of Items Returned

The GetXML servlet provides a throttling mechanism to allow developers to define the number of results returned on any single request and where in the result set to start. For example, if a search request yields 500 results, specifying resultCounter=450 will return only the last 50 records. Similarly, specifying startIndex=50 will return only the first 50 records.

Paging Results

In addition to controlling the number of results to display, the GetXML servlet also provides a mechanism to support "paging". This concept, common to many web sites, allows results to be displayed over a number of pages, so that, for example, a request that yields 500 objects could be displayed over 10 pages of 50 objects each. When the paging feature is used, the GetXML servlet will include XLinks to each of the result pages in an XML <page/> element. The element data of the <page/> element is the number of the page, suitable for output as text or HTML when using an XSL stylesheet:

```
<page number="1"
        xlink:type="simple"
        xlink:href="http://cabio.nci.nih.gov/cacore31/GetXML?query={query}&
                pageNumber=4&resultCounter=1000&startIndex=0"> 4 </page>
```

Limitations

When specifying attribute values in the query string, use of the following characters generates an error: [ ] / \ # & %.

Perl API

Programmatic access to the caCORE system for Perl users is provided through a Perl package called caCOREperl. This package implements the caCORE object model and integrates with the caCORE server via the caCORE web services client built using SOAP::Lite and XML parsing. Developers using caCOREperl can take advantage of the capability to deal with caCORE data in the form of native Perl objects.

The following diagram illustrates the overall architecture of caCOREperl and its relationship with the caCORE server.

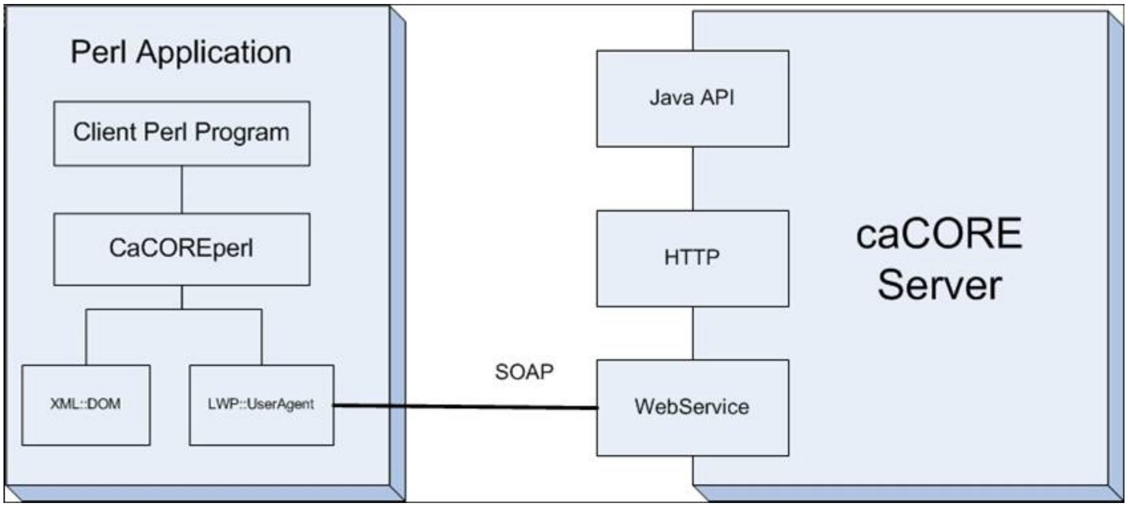


Figure 3.4 caCORE Perl API architecture

Language-Specific Considerations

All caCORE domain objects are represented as Perl objects in caCOREperl and follow a standard pattern:

- A domain object can be created with a constructor method
- All attributes of the domain object are accessible via getter and setter methods
- All associated domain objects are accessible via getter and setter methods

In order to conform to Perl conventions, the package naming structure (described for each domain object model in this guide) has been slightly modified, as follows:

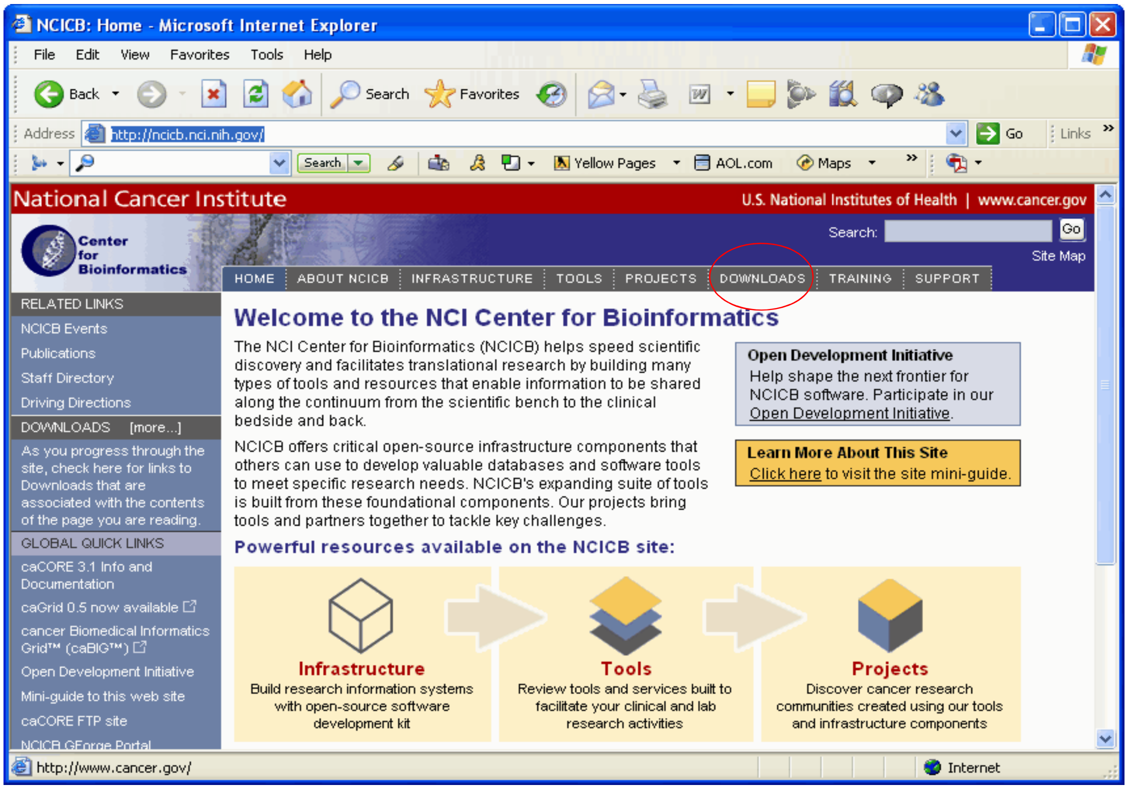
Fully-Qualified Package Name	***caCOREperl Package***
`gov.nih.nci.cabio.domain`	`CaCORE::CaBIO`
`gov.nih.nci.cadsr.domain`	`CaCORE::CaDSR`
`gov.nih.nci.common.domain`	`CaCORE::Common`
`gov.nih.nci.evs.domain`	`CaCORE::EVS`

Installation and Configuration

caCOREperl will work on any operating system (Windows, Solaris, Linux, etc.) that supports Perl version 5.6.0 or higher. Most UNIX and Linux platforms come with Perl already installed. Windows users can download Perl from http://www.activestate.com.

caCOREperl is dependent on two Perl modules, XML::DOM and LWP::UserAgent. These must be present on the client machine in order for caCOREperl to work. If you do not already have these installed, use one of the methods below to obtain them and install caCOREperl.

First, download the caCOREperl package from the Downloads section of the NCICB web site (*Figure 3.5*):


NCICB: Home - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://ncicb.nci.nih.gov/ Go Links
Search Yellow Pages AOL.com Maps
National Cancer Institute
U.S. National Institutes of Health | www.cancer.gov
Center for Bioinformatics
Search: Go
Site Map
HOME ABOUT NCICB INFRASTRUCTURE TOOLS PROJECTS DOWNLOADS TRAINING SUPPORT
RELATED LINKS
NCICB Events
Publications
Staff Directory
Driving Directions
DOWNLOADS [more...]
GLOBAL QUICK LINKS
caCORE 3.1 Info and Documentation
caGrid 0.5 now available
cancer Biomedical Informatics Grid™ (caBIG™)
Open Development Initiative
Mini-guide to this web site
caCORE FTP site
NCICB GForge Portal
Infrastructure
Tools
Projects
http://www.cancer.gov/
Internet

As you progress through the site, check here for links to Downloads that are associated with the contents of the page you are reading.

Welcome to the NCI Center for Bioinformatics

The NCI Center for Bioinformatics (NCICB) helps speed scientific discovery and facilitates translational research by building many types of tools and resources that enable information to be shared along the continuum from the scientific bench to the clinical bedside and back.

NCICB offers critical open-source infrastructure components that others can use to develop valuable databases and software tools to meet specific research needs. NCICB's expanding suite of tools is built from these foundational components. Our projects bring tools and partners together to tackle key challenges.

Open Development Initiative
Help shape the next frontier for NCICB software. Participate in our Open Development Initiative.

Learn More About This Site
Click here to visit the site mini-guide.

Powerful resources available on the NCICB site:

Infrastructure: Build research information systems with open-source software development kit

Tools: Review tools and services built to facilitate your clinical and lab research activities

Projects: Discover cancer research communities created using our tools and infrastructure components

Figure 3.5 Downloads section on the NCICB website

1. Open your browser and go to http://ncicb.nci.nih.gov
2. Click the Download link on the menu bar
3. Scroll down to the section titled caBIO and click on the Download link
4. In the provided form, enter your name, email address and institution name and click to Enter the Download Area
5. Accept the license agreement
6. On the caCORE downloads page, download the caCOREperl Zip file from the Primary Distribution section
7. Extract the contents of the downloadable archive to a temporary directory on your hard drive (for example, c:\temp\caCOREperl on Windows or /tmp/cacoreperl on Linux).

Alternatively, you can also download caCOREperl from the CPAN archive (www.cpan.org).

Installation Option One: Using make

Open a command window or terminal prompt and go to the directory where you extracted the downloadable archive (for example, enter `cd c:/temp/caCOREperl`). Then enter:

```
perl Makefile.PL
```

Alternatively, if you plan to install caCOREperl somewhere other than your system's Perl library directory, you can specify the location by entering:

```
perl Makefile.PL PREFIX=/home/me/perl INSTALLDIRS=perl
```

Then build caCOREperl by entering

```
make
```

To test whether the module has been properly build, enter:

```
make test
```

If you have write access to the Perl library directories, you may then install caCOREperl by entering

```
make install
```

Installation Option Two: Using Perl Package Manager (PPM)

Perl Package Manager (PPM) is a tool that is installed with ActiveState Perl and is used to manage Perl packages. After installing ActiveState Perl, start PPM (the exact method depends on the version, but is typically done through the Start menu by going to Programs | ActiveState Perl | PPM, or from the command prompt by running PPM in the directory that contains the Perl executable files).

Once you have started PPM, a window containing the PPM prompt will appear. At the prompt, enter:

```
install XML-DOM
```

This will install the XML::DOM module on your PC.

Installation Option Three: Using CPAN

This will work on any version of Perl and on all platforms. From a command prompt, enter:

```
perl -MCPAN -e shell
```

Refer to the CPAN documentation for more details on how to use this command.

Installation Option Four: Manual Installation

This is only recommended if none of the above methods works. After extracting the caCOREperl distribution, copy the entire `lib/caCORE` folder to the `lib/site` folder in your Perl installation folder. For example, if you extracted caCOREperl to `c:/temp` and your Perl installation is in `c:/perl`, at the command prompt enter

```
copy c:/temp/caCORE/lib/CaCORE c:/perl/lib/site
```

Service Methods

The methods that provide programmatic access to running the caCORE server are exposed by the ApplicationService class located in the caCORE package. This class encapsulates the calls to the Web services API required for the functioning of caCOREperl.

The ApplicationService object follows the singleton pattern, in that each program will ONLY contain one instance of such class. It can be constructed using the instance(url) method, where "url" is the URL of the service endpoint of the caCORE webservice. If no URL is provided, it will default to the caCORE production server, http://cabio.nci.nih.gov/cacore32/ws/caCOREService.

```
my $appsvc = CaCORE::ApplicationService->
     instance("http://cabio.nci.nih.gov/cacore32/ws/
caCOREService");
```

The ApplicationService class provides two query methods that allow users to search for data based on the specific needs and types of queries to be performed:

Method Signature	`queryObject(targetPath, sourceObject)`
Description	Returns an array of objects of the type specified by targetPath and that conform to the criteria specified by sourceObject
Example	`queryObject("CaCORE::CaBIO::Gene", $gene)`

Method Signature	`query(targetPath, sourceObject,`
	`       startIndex, requestSize)`

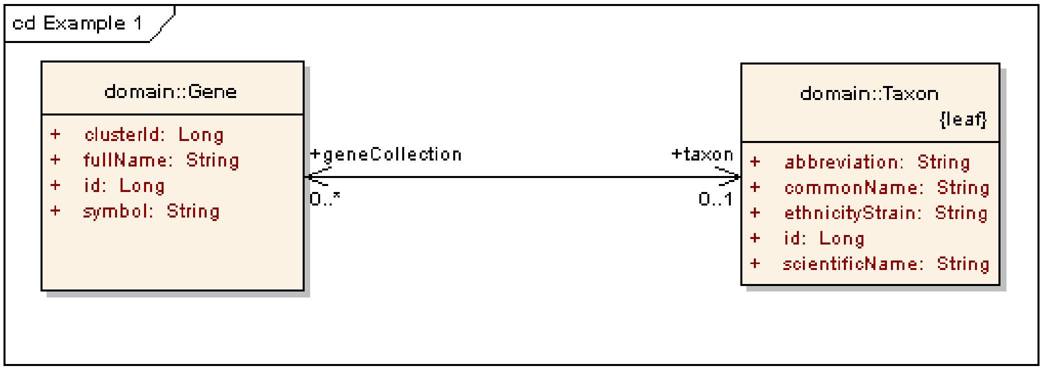
Description	Identical to the previous queryObject method, but allows for control over the size of the result set by specifying the index of the first result to return and the maximum size of the result set
Example	`query("CaCORE::CaBIO::Gene", $gene, 301, 100)`

Examples of Use

This section includes a number of examples that demonstrate the use of the caCOREperl package. Included with each example is a brief description of the type of search being performed, a UML diagram depicting the domain objects used, and the example code accompanied by explanatory text.

Example One: Simple Search (Single Criteria Object)

In this example, a search is performed for all genes whose symbols start with 'brca'. The code iterates through the returned objects and prints out the symbol and name of each object along with the name of an associated object of type Taxon. The fetch of the associated Taxon object is done in the background and is completely transparent to the user.

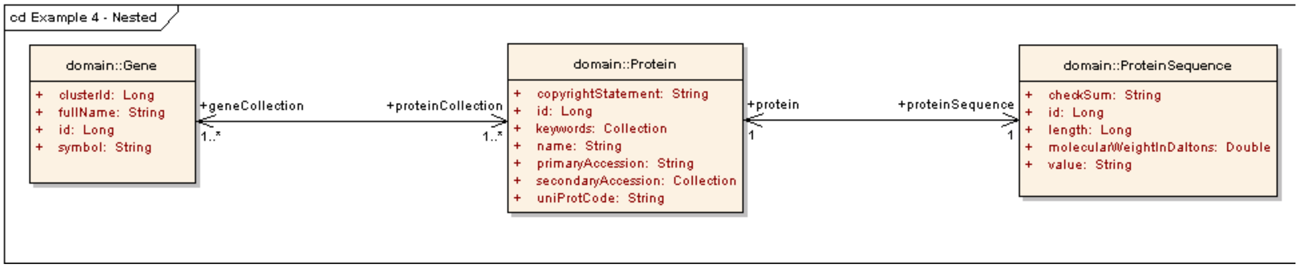


```
1   use CaCORE::ApplicationService;
2   use CaCORE::CaBIO;
3   my $appsvc = CaCORE::ApplicationService->
4      instance(“http://cabio.nci.nih.gov/cacore32/ws/caCOREService”);
5   my $geneCriteria = new CaCORE::CaBIO::Gene;
6   $geneCriteria ->setSymbol(“brca*”);
7   my @genes = $appsvc->queryObject(“CaCORE::CaBIO::Gene”, $geneCriteria);
8   foreach my $gene (@genes) {
9      print “Symbol: “ . $gene->getSymbol .
10           “ Name: “ . $gene->getFullName .
11           “ Taxon: “ . $gene->getTaxon->getScientificName .
12           “\n”;
13  }
```

Lines	***Description***
3-4	Returns the instance of the singleton class ApplicationService; this class defines the service methods used to access caCORE server.
5-6	Creates a criterion object that defines the attribute values for which to query.
7	Calls the queryObject method of the ApplicationService implementation and passes it the type of objects to return and the criteria that returned objects must meet, defined by the $geneCriteria object; the search method returns objects in an array.
8	Iterates through the Gene objects in the @genes array.
9	Prints the symbol attribute.
10	Prints the fullName attribute.
11	Fetches an associated object of type Taxon (note that this is an extra call to caCORE server) and prints its scientificName attribute.

Example Two: Nested Search

A nested search is one where a traversal of more than one class-class association is required to obtain a set of result objects given the criteria object. This example demonstrates one such search in which the criteria object passed to the search method is of type Gene, and the desired objects are of type ProteinSequence. Because there is no direct association between these two classes, the path of the traversal is passed to the search method enabling the query to be performed.



```
 1  my $geneCriteria = new CaCORE::CaBIO::Gene;
 2  $geneCriteria ->setSymbol(“TP53”);
 3  my $appsvc = CaCORE::ApplicationService->
 4     instance(“http://cabio.nci.nih.gov/cacore32/ws/caCOREService”);
 5  my @proteinSequences = $appsvc->queryObject(
 6     “CaCORE::CaBIO::ProteinSequence,CaCORE::CaBIO::Protein”,
 7     $geneCriteria);
 8  foreach my $proteinSequence (@proteinSequences) {
 9     print “Id: “ . $proteinSequence->getId .
10          “ Value: “ . $proteinSequence->getValue .
11          “\n”;
12  }
```

Lines	Description
1-2	Creates a Gene object and sets the symbol to "TP53".
6	Defines search path as traversing from the criteria object of type Gene through Protein to ProteinSequence; note that the first element in the path is the desired class of objects to be returned, and that subsequent elements traverse back to the criteria object.
7	Sets the criteria object to the previously-created Gene.

Example Three: Throttled Search

Depending on the search criteria, a search may yield a large result set, which cause slower response time and increase the likelihood of failure. A throttle mechanism is provided by:

```
ApplicationServive->query(targetClassName, criteria,
startIndex, requestedSize);
```

In the following example:

```
 1  use CaCORE::ApplicationService;
 2  use CaCORE::CaBIO;
 3  my $appsvc = CaCORE::ApplicationService->
 4     instance(“http://cabio.nci.nih.gov/cacore32/ws/caCOREService”);
 5  my $geneCriteria = new CaCORE::CaBIO::Gene;
 6  $geneCriteria ->setSymbol(“brca*”);
 7  my @genes = $appsvc->query (“CaCORE::CaBIO::Gene”, $geneCriteria, 301, 200);
 8  foreach my $gene (@genes) {
 9     print “Symbol: “ . $gene->getSymbol .
10           “ Name: “ . $gene->getFullName .
11           “\n”;
12  }
```

Lines	Description
7	Request return result to start from 301, request 200 objects.

Limitations

Since caCOREperl depends on the caCORE Web services API, the limitations of that interface apply here as well, most notably:

- By default, the ApplicationService->queryObject method limits the result set to 1000 objects, even if the size of the result set is larger. To retrieve the objects past the 1000th record, you must use the ApplicationService->query method.

- Because the Web services invocation has an inherent timeout behavior, queries which take a long time to execute may not complete. If this is the case, use the ApplicationService->query method to specify a smaller result count.

CHAPTER 4

ENTERPRISE VOCABULARY SERVICES

This chapter describes the Enterprise Vocabulary Services (EVS) project and its application programming interface.

Topics in this chapter include:

Introduction

The Enterprise Vocabulary Services (EVS) project is a collaborative effort of the NCI Center for Bioinformatics and the NCI Office of Communications. Controlled vocabularies are important to any application involving electronic data sharing. Two areas where the need is perhaps most apparent are clinical trials data collection and reporting and more generally, data annotation of any kind. The *NCI Thesaurus* is a biomedical thesaurus developed by EVS in response to a need for consistent shared vocabularies among the various projects and initiatives at the NCI as well as in the entire cancer research community. The EVS project also produces the *NCI Metathesaurus*, which is based on NLM's Unified Medical Language System Metathesaurus (UMLS) supplemented with additional cancer-centric vocabulary.

A critical need served by the EVS is the provision of a well designed ontology covering cancer science. Such an ontology is required for data annotation, inferencing and other functions. The data to be annotated might be anything from genomic sequences to case report forms to cancer image data. The NCI Thesaurus covers all of these domains. A few of the included specialties it includes are pertinent to disease, biomedical instrumentation, anatomical structure, and gene/protein information. The NCI Thesaurus is updated monthly to keep up with developments in cancer science.

The NCI Thesaurus is implemented as a Description Logic vocabulary and, as such, is a self-contained and logically consistent terminology. Unlike the NCI Thesaurus, the purpose of the NCI Metathesaurus is *not* to provide unequivocal or even necessarily consistent definitions. The purpose of the NCI Metathesaurus, like the UMLS Metathesaurus, is to provide mappings of terms across vocabularies. The caCORE EVS objects described in this chapter provide access to both the NCI Thesaurus and the NCI Metathesaurus.

The caCORE EVS API provides access to the NCI Metaphrase, which hosts the Metathesaurus database, and the NCI Distributed Terminology Server (DTS), which hosts the NCI Thesaurus and several other vocabularies.

NCI licenses the Metaphrase and DTS servers from Apelon Inc. Each server has a proprietary Java API. Because of the proprietary nature of these APIs, these interfaces cannot be made available to the public. Furthermore, NCI has extended and otherwise modified the Metaphrase and DTS servers to provide functionality that is not present in the commercial version of these products. Therefore, NCI developed a public domain open source wrapper that provides full access to the basic and enhanced capabilities of both servers. This public API is a component of caCORE.

Before actually describing the caCORE Java API to the EVS, a brief overview of the UMLS Metathesaurus is provided, upon which the NCI Metathesaurus is based. This is followed by a short discussion of description logic, its role in the area of knowledge representation, and its implementation in the NCI Thesaurus.

The UMLS Metathesaurus

The NCI Metathesaurus is based on the UMLS Metathesaurus, supplemented with additional cancer-centric vocabulary. Excellent documentation on the UMLS is available at the UMLS Knowledge Sources web site.

A brief overview of the UMLS Metathesaurus is included here, but it is strongly recommended that users who wish to gain a deeper understanding refer to the above web site. Only those features of the UMLS Metathesaurus that are relevant to accessing the NCI Metathesaurus are described here.

The UMLS Metathesaurus is a unifying database of concepts that brings together terms occurring in over 100 different controlled vocabularies used in biomedicine. When adding terms to the Metathesaurus, the UMLS philosophy has been to preserve all of the original meanings, attributes, and relationships defined for those terms in the source vocabularies, and to retain explicit source information as well. In addition, the UMLS editors add basic information about each concept and introduce new associations that help to establish synonymy and other relationships among concepts from different sources.

Given the very large number of related vocabularies incorporated in the Metathesaurus, there are instances where the same concept may be known by many different names, as well as instances where the same names are intended to convey different concepts. To avoid ambiguity, the UMLS employs an elaborate indexing system, the central kingpin of which is the *concept unique identifier* (CUI). Similarly, each unique concept name or string in the Metathesaurus has a string unique identifier (SUI).

In cases where the same string is associated with multiple concepts, a numeric tag is appended to that string to render it unique as well as to reflect its multiplicity. In addition, the UMLS Metathesaurus editors may create an alternative name for the concept that is more indicative of its intended interpretation. In these cases, all three names for the concept are preserved.

Several types of relationships are defined in the UMLS Metathesaurus, and four of these are captured by the NCI Metaphrase interface:

Broader (RB)	The related concept has a more general meaning.
Narrower (RN)	The related concept has a more specific meaning.
Synonym (SY)	The two concepts are synonymous.
Other related (RO)	The relationship is not specified but is something other than synonymous, narrower, or broader.

The UMLS *Semantic Network* is an independent construct whose purpose is to provide consistent categorization for all concepts contained in the UMLS Metathesaurus, and to define a useful set of relationships among these concepts. As of the 2005AC release, the Semantic Network defined a set of 135 basic semantic types or categories, which could be assigned to these concepts, and 54 relationships that could hold among these types.

The major groupings of semantic types include organisms, anatomical structures, biologic function, chemicals, events, physical objects, and concepts or ideas. Each UMLS Metathesaurus concept is assigned at least one semantic type, and in some cases, several. In all cases, the most specific semantic type available in the network hierarchy is assigned to the concept.

The NCI Metathesaurus includes most of the UMLS Metathesaurus, with certain proprietary vocabularies of necessity excluded. In addition, the NCI Metathesaurus includes terminologies developed at NCI along with external vocabularies licensed by NCI. The local vocabularies developed at NCI are described in Table *4.1*. As noted there, a limited model of the NCI Thesaurus is also accessible via the NCI Metathesaurus, as the NCI Source. Additional external vocabularies include MedDRA, SNOMED, ICD-O-3, and other proprietary vocabularies.

The NCI Metathesaurus is available through the Java API described in this chapter.

Vocabulary	***Content***	***Usage***
NCI Source	Limited model of the NCI Thesaurus	Reference terminology for cancer research applications
NCIPDQ	Expanded and re-organized PDQ	CancerLit indexing and clinical trials accrual
NCISEER	SEER terminology	Incidence reporting
CTEP	CTEP terminology	Clinical trials administration
MDBCAC	Topology and morphology	Cancer genome research
ELC2001	NCBI tissue taxonomy	Tissue classification for genetic data such as cDNA libraries.

Table 4.1 NCI local source vocabularies included in the Metathesaurus

Vocabulary	Content	Usage
ICD03	Oncology classifications	Cancer genome research and incidence reporting
MedDRA	Regulatory reporting terminology	Adverse event reporting
MMHCC	Mouse Cancer Database terminology	Mouse Models of Human Cancer Consortium
CTRM	Core anatomy, diagnosis and agent terminology	Translational research by NCICB applications

Table 4.1 NCI local source vocabularies included in the Metathesaurus (Continued)

Knowledge Representations and Description Logic

Knowledge representation has long been a prime focus in artificial intelligence research. This area of research asks how one can accurately encode the rich and highly detailed world of information that is required for the application area being modeled and yet, at the same time, capture the implicit commonsense knowledge. One of the most common approaches to this problem in the 1970s was to utilize *frame-based representations*.

The basic idea of a frame is that important objects in our world fall into natural classes, and that all members of these classes share certain properties or attributes, called *slots*. For example, all dogs have four legs, a tail (or vestige of one), whiskers, etc. Restaurants generally have tables, chairs, eating utensils, and menus. Thus, when we enter a new restaurant or encounter a new dog, we already have a "frame of reference" and some expectations about the properties and behaviors of these entities.

In a seminal paper by Marvin Minsky published in 1975, he placed the frame representation paradigm in the context of a semantic network of nodes, attributes, and relations. Figure 4.1 shows a simple frame-based representation of an earthquake, as it might be used in a semantic network of news stories.[1]

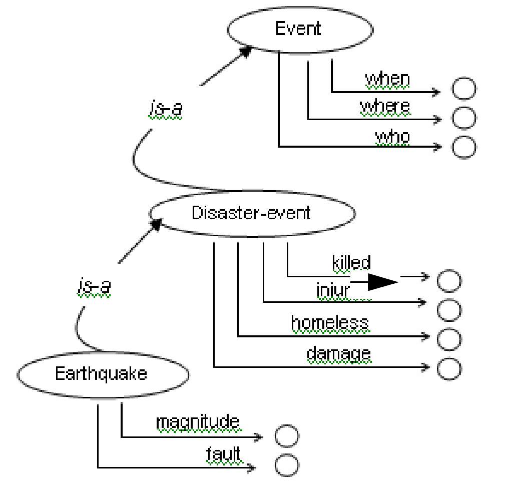


Figure 4.1 An earthquake in a semantic network of news stories

[1] This example is excerpted from *Artificial Intelligence*, by Patrick Winston, Addison-Wesley, 1984.

At the same time that frame-based representations were being explored, a popular alternative approach was to use (some subset of) first-order predicate logic (FOL)-often implemented as a Prolog program. While propositional logic allows one to make simple statements about concrete entities, a complete first-order logic allows one to make general statements about anonymous elements, with the introduction of variables as placeholders. The example below contrasts the difference in expressivity between propositional logic and FOL:

Propositional Logic	***First-order Predicate Logic***
All men are mortal. Socrates is a man. Socrates is mortal.	$\forall x$: Man(x) Mortal(x) Man(Socrates)

In other words, in FOL it is possible to express general rules of inference that can be applied to all entities whose attributes satisfy the left-hand side of the inference → operator. Thus, simply asserting *Man*(Socrates) entails *Mortal*(Socrates).

Since logic programming is based on the tenets of classical logic and comes equipped with automated theorem-proving mechanisms, this approach allowed the development of inference systems whose soundness and completeness could be rigorously demonstrated. But while many of these early inference systems were logically sound and complete, they were often not very useful, as they could only be applied to highly proscribed areas or "toy problems." The problem was that a complete first-order predicate logic is itself computationally intractable, as certain statements may prove *undecidable*.

Suppose for example that we are trying to establish that some theorem, *P(x)*, is true. The way a theorem prover works is to first negate the theorem and, subsequently, to combine the negated theorem ($\neg P(x)$) with stored axioms in the body of knowledge to show that this leads to a logical contradiction. Ultimately, when the theorem prover derives the conclusion $P(x) \wedge \neg P(x)$, the program terminates and the theorem is considered proven.

This method of proof by refutation is guaranteed to terminate when it is indeed upheld by the body of knowledge. The problems arise when the initial theorem is not valid, as its negation may not produce a logical contradiction, and thus the program may not terminate.

In contrast, the frame representations offered a rich, intuitive means of expressing domain knowledge, yet they lacked the inference mechanisms and rigor that predicate logic systems could provide. As suggested by *Figure 4.1*, the frame representation captures a good deal of implicit knowledge. For example, we expect that all disaster events, including earthquakes, have information about fatalities and injuries and the extent of loss and property damage. In addition, we expect that these events will have locations, dates, and individuals associated with them.

Early efforts to apply predicate logic to frame representations in order to make this information explicit however, soon revealed that the problem was computationally intractable. This occurred for two reasons: (1) The frame representation was too permissive; more rigorous definitions were required to make the representation computational; and (2) the intractability of first-order predicate logic itself.

Several subsets of complete FOL have since been defined and successfully applied to develop useful computational models capable of significant reasoning. For example, the Prolog programming language is based on a subset of FOL that severely limits the use of negation. The family of description logic (DL) systems is a more recent development, and one that is especially well-suited to the development of ontologies, taxonomies, and controlled vocabularies, as an important function of a DL is as an auto-classifier.

Description Logic

Description logic can be viewed as a combination of the frame-based approach with FOL. In the process, both models had to be scaled back to achieve an effective solution. Like frames, the DL representation allows for concepts and relationships among concepts, including simple taxonomic relations as well as other meaningful types of association. Certain restrictions however, are placed on these relations. In particular, any relation that involves class membership, such as the *isa* or *inverse-isa* relations, must be strictly acyclic.

The predicate logic used in a description logic system is also limited in various ways, depending on the implementation. For example, the most minimal form of a DL does not allow any form of existential quantification. This limitation allows for a very easily computed solution space, but the resulting expressivity is severely diminished. The next step up in representational power allows limited existential quantification but without atomic negation.

Indeed, there is today a large family of description logics that have been realized, with varying levels of expressivity and resulting computational complexities. In general, DLs are decidable subsets of FOL, and the decidability is due in large part to their acyclicity. The theory behind these models is beyond the scope of this discussion, and the interested reader is referred to *The Description Logic Handbook*, by Franz Baader, et al. (eds.), Cambridge University Press, 1993, ISBN number 0-521-78176-0.

The two main ingredients of a DL representation are *concepts* and *roles*. A major distinction between description logics and other subsets of FOL is its emphasis on set notations. Thus a DL concept never corresponds to a particular entity but rather to a *set* of entities, and the notations used for logical conjunction and disjunction are set intersection and union.

DL concepts can also be thought of as unary predicates in FOL. Thus the DL expression $Person \cap Young$ can be interpreted as the set of all children, with the corresponding FOL expression $Person(x) \wedge Young(x)$. Syntactically then, DL expressions are variable free, with the understanding that the concepts always reference sets of elements.

A DL *role* is used to indicate a relationship between the two sets of elements referenced by a pair of concepts. In general, DL notations are rather terse, and the concept (or set of elements) of interest is not explicitly represented. Thus, to represent the set of individuals whose children are all female, we would use: $\forall x$ *hasChild.Female*. The equivalent expression in FOL might be something like:

$\forall x$ *:hasChild(y,x)*$\longrightarrow$*female(x)*

In terms of set theory, a role potentially defines the Cartesian product of the two sets. Roles can have restrictions, however, which place limitations on the possible relations. A *value* restriction limits the type of elements that can participate in the relation; a *number* restriction limits the number of such relations an element can participate in.

In addition, each role defines a *directed* relation. For example, if *x* is the child of *y*, *y* is not also the child of *x*. In the above example *hasChild*, the parent concept is considered the *domain* of the relation, and the child is considered the *range*. Elements belonging to the set of objects defined by the range concept are also called role fillers. Number restrictions apply to the number of role fillers that are required or allowed in a relation. For example, a parent can be defined as a person having at least one child:

$$Person \cap (child)$$

A DL representation is constructed from a ground set of *atomic concepts* and *atomic roles*, which are simply asserted. *Defined concepts* and *defined roles* are then derived from these atomic elements, using the set operations of intersection, union, negation, etc. Most DLs also allow existential and universal quantifiers, as in the above examples. Note, however, that these quantifiers always apply to the role fillers only.

The fundamental inference operation in DL is *subsumption*, and is usually indicated with subset notation. Concept A is said to subsume B, or $\mathrm{A} \subseteq \mathrm{B}$, when all members of concept B are contained in the set of elements defined by concept A, but not vice versa. That is, if B is a proper subset of A, then A subsumes B. This capability has far-reaching repercussions for vocabulary and ontology developers, as it enables the system to automatically classify newly introduced concepts. Moreover, correct subsumption inferencing can be highly nontrivial, as, in general, this requires examining all of the relationships defined in the system and the concepts that participate in those relations.

Description Logic in the NCI Thesaurus

The NCI Thesaurus is currently developed using the proprietary Apelon Inc. Ontylog™ implementation of description logic. Ontylog is distributed as a suite of tools for terminology development, management, and publishing. Although the underlying inference engine of Ontylog is not exposed, the implementation has the characteristics of what is called an AL- (attributive language) or FL- ("Frame Language") description logic. It does not support atomic negation but does appear to provide all other basic description logic functionality.

The NCI Thesaurus is edited and maintained in the Terminology Development Environment (TDE) provided by Apelon. The TDE is an XML-based system that implements the DL model of description logic based on Apelon's Ontylog Data Model. The Data Model uses four basic components: *Concepts*, *Kinds*, *Properties*, and *Roles*.

As in other DL systems, *Concepts* correspond to nodes in an acyclic graph, and *Roles* correspond to directed edges defining relations between concept members. Each *Concept* has a unique *Kind*. Formally, *Kinds* are disjoint sets of *Concepts* and represent major subdivisions in the NCI Thesaurus.

More concretely, *Kinds* are used in the *Role* definitions to constrain the *domain* and range values for that *Role*. Each *Role* is a *directed* relation that defines a triplet consisting of two concepts and the way in which they are related. The domain defines the *Concept* that the *Role* applies to, and the *Range* defines the possible values-in other words, *Concepts*, that can fill that *Role*. For example, the Role *geneEncodes* might have its domain restricted to the *Gene_Kind* and its range to the *Protein_Kind*. This Role then, essentially states that *Genes* encode *Proteins*.

As in all DLs, all roles are passed from parent to child in the inheritance hierarchy. For example, a "Malignant Breast Neoplasm" has the role *located-in*, connecting it to the concept "Breast." Thus, since the concept "Breast Ductal Carcinoma" *is-a* "Malignant

Breast Neoplasm," it inherits the *located_in* relation to the "Breast" concept. These lateral nonhierarchical relations among concepts are referred to as associative or semantic roles - in contrast to the hierarchical relations that reflect the *is-a* roles.

In the first-order algebra upon which Ontylog DL is based, every defined relationship also has a defined inverse relationship. For example, if *A* is contained by *B*, then *B* contains *A*. Inverse relationships are useful and are expected by human users of ontologies. However, they have a computational cost. If the edges connecting concept nodes are bi-directional, then the computation quickly becomes intractable. Therefore in the Ontylog implementation of DL, inverse relationships are not stored explicitly but computed on demand.

Figure 4.2 gives an overview of how the NCI Thesaurus is deployed. Apelon provides both graphical and programmatic interfaces to its Distributed Terminology System and its Terminology Development Environment.

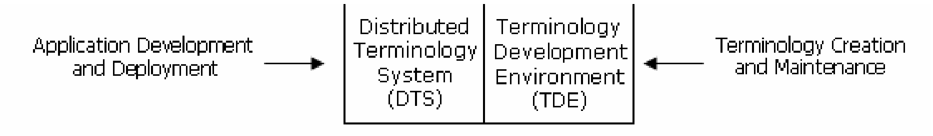

Application Development and Deployment
Distributed Terminology System (DTS)
Terminology Development Environment (TDE)
Terminology Creation and Maintenance

Figure 4.2 An overview of the NCI Thesaurus infrastructure

The graphical interfaces to the DTS are available for browsing at the NCICB EVS Download site; the APIs however are proprietary, and thus not available to the public. The domain objects described in this chapter have been implemented to provide a public API to the DTS, including the NCI-specific extensions to the DTS that support functionality such as concept history.

Concept Edit History in the NCI Thesaurus

One of the primary uses of the NCI Thesaurus is as a resource for defining tags or retrieval keys for the curation of information artifacts in various NCI repositories. Since these tags are defined at a fixed point in time, however, they necessarily reflect the content and structure of the NCI Thesaurus at that time only. Given the rapidly evolving terminologies associated with cancer research, there is no guarantee that the tags used at the time of curation in the repository will still have the same definition in subsequent releases of the Thesaurus. In most cases the deprecation or redefinition of a previously defined tag is not disastrous, but it may compromise the completeness of the information that can be retrieved.

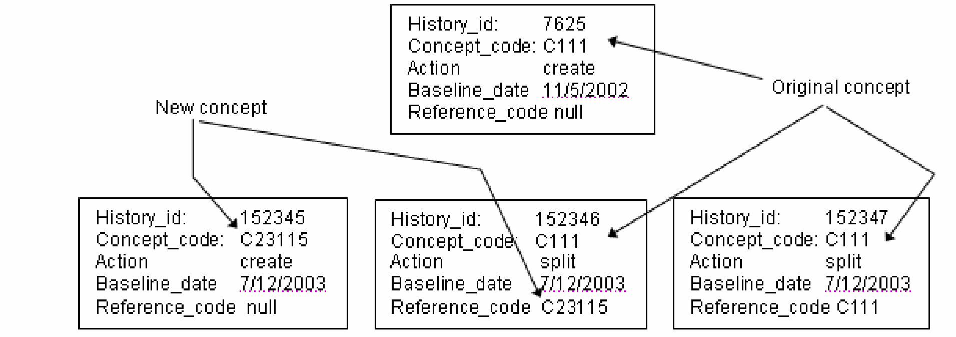
In order to address this issue, the EVS team has developed a *history* mechanism for tracing the evolution of concepts as they are created, merged, modified, split, or retired. (In the NCI Thesaurus, no concept is ever deleted.) The basic idea is that each time an edit action is performed on a concept, a record is added to a history table. This record contains information about relations that held for that concept at the time of the action as well as other information, such as version number and timestamp that can be used to reconstruct the state when the action was taken. Table *4.2* summarizes the information stored in the history table.

Column Name	***Description***
History_ID	Unique consecutive number for use as the database primary key
Concept_Code	The concept code for the concept currently being edited
Action	Edit Action: {Create, Modify, Split, Merge, Retire}
Baseline_Date	Date of NCI Thesaurus Baseline (see discussion below)
Reference_Code	This field contains the concept code of a second concept either participating in or affected by the editor's action. Captures critical information concerning the impact of the edit actions on other concepts. The value will always be null if the action is Create or Modify.

Table 4.2 The NCI Thesaurus concept history table

Capturing the history data for a Split, Merge, or Retire action is more complicated. In a Split, a concept is redefined by partitioning its defining attributes between two concepts, one of which retains the original concept's code and one that is newly created. This action is taken when ambiguities in the original concept's meaning require clarification by narrowing its definition.

In the case of a Split, three history records will be created: one for the newly created concept, (with a null Reference_Code), and two for the original concept that is being split. In the first of these two records, the Reference_Code is the code for the new concept; in the second it is the code of the split concept.



History_id: 7625
Concept_code: C111
Action create
Baseline_date 11/5/2002
Reference_code null

History_id: 152345
Concept_code: C23115
Action create
Baseline_date 7/12/2003
Reference_code null

History_id: 152346
Concept_code: C111
Action split
Baseline_date 7/12/2003
Reference_code C23115

History_id: 152347
Concept_code: C111
Action split
Baseline_date 7/12/2003
Reference_code C111

Figure 4.3 History records for the split action

For Merge actions, the situation is similar to a Split. In this case, two ambiguous concepts must be combined, and only one of the original concepts is retained. Again there will be three history records created: two for the concept that will be retired during the merge, and one for the "winning" concept. The Reference_Code in the history record for the "winning" concept will be the same as the Concept_Code; i.e., the concept points to itself as a descendant in the Merge action. The Reference_Code will be null in one of the entries for the retiring concept, while the second entry will have the code of the "winning" concept; thus, this Reference column points to the concept into which the concept in the Concept_Code column is being merged.

Finally, if the action is Retire, there will be as many history entries as the concept has parent concepts. The Reference column in these entries will contain the concept code of the parent concepts, one parent concept per history entry. The motivation for this is that end-users with documents coded by such retired concepts may find a suitable replacement among the concept's parents at the time of retirement.

The caCORE EVS APIs support concept history queries, and for programmatic consistency, a minimal history is added to all vocabularies served from the DTS that are not edited by the EVS group. Concepts in vocabularies that are *not* edited by EVS will have a single history entry associated with them-a *Create* action with date "May 1, 2003."

In the case of the NCI Thesaurus, concept history tracking has been ongoing internally since December 2002. However, for the purpose of publication in the DTS, a specific baseline has been selected to serve as "time zero" for concept history. This baseline is (internal) version 03.08c, which immediately preceded the NCI Thesaurus Version 2.0 released in caCORE 2.0. All of the concepts in this baseline have a *Create* action associated with them, dated "August 12, 2003", the date of the 03.08c build.

caCORE EVS API

The caCORE 3.2 EVS API is a public domain open source wrapper that provides full access to the basic and enhanced capabilities of the Metaphrase and DTS (Distributed Terminology Server) Servers. The NCI Metaphrase Server hosts the Metathesaurus database and the NCI DTS Server hosts the NCI Thesaurus and several other vocabularies. Java clients accessing the NCI Thesaurus and Metathesaurus vocabularies communicate their requests via the open source caCORE EVS API. The proprietary APIs are included as jar files in the caCORE 3.2 Server distribution.

The NCI Metathesaurus and the NCI Thesaurus (Description Logic Vocabulary) are maintained by NCICB. The caCORE EVS API also provides access to Description Logic vocabularies maintained by external entities like "GO", "LOINC","HL7" etc.

The UML Class diagram in *Figure 4.4* provides an overview of the caCORE 3.2 EVS domain object classes. The DescLogicConcept and MetaThesaurusConcept are two central Concept classes in the model, with most of the other classes organizing themselves around these entities.

The Vocabulary and SecurityToken classes were added to the model in the 3.2 release. The SecurityToken class can be used to specify security credentials like username, password, security token etc.

A DAO Security model has also been implemented in this release to provide data level security to Vocabularies. The MedDRASecurity class, the class that implements the DAOSecurity interface, validates a token against the MedDRA vocabulary and prevents unauthorized users from performing any of the queries against MedDRA. To access MedDRA via the caCORE EVS API, a user must obtain a valid token from NCICB.

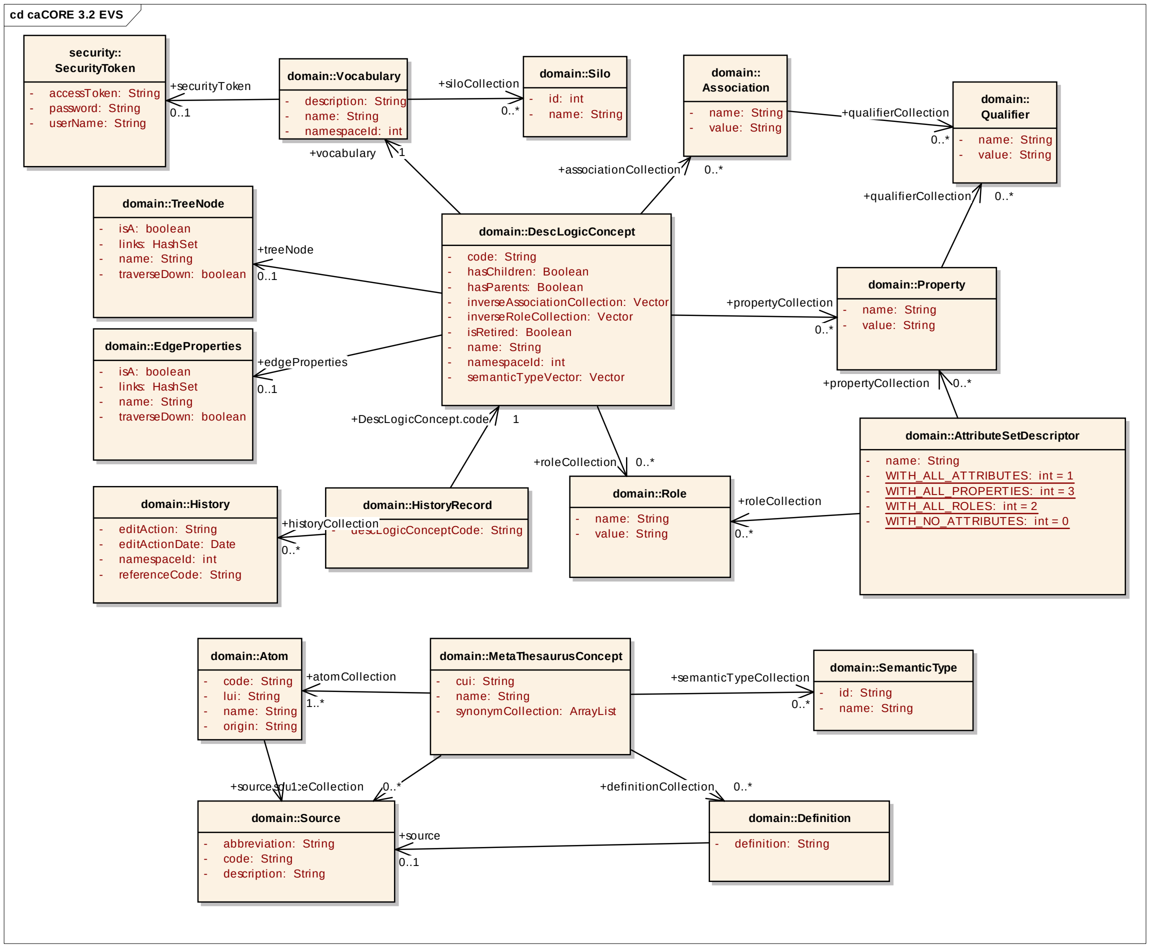


Figure 4.4 The caCORE EVS API domain object classes

The EVS API diverges somewhat from the other caCORE domain models in its search mechanisms, as described in the next section. While the other APIs have direct access to their databases, the EVS API does not. Since all EVS queries are passed through the proprietary APIs provided by Apelon, the search and retrieval capabilities are effectively proscribed by the features implemented by these third-party tools.

EVS Domain Object Catalog

The caCORE EVS domain objects are implemented as Java beans in the gov.nih.nci.evs.domain package. Table *4.3* lists each class and a description. Detailed descriptions about each class and its methods are present in the caCORE 3.2 Java-

Docs. The only interface implemented by the EVS domain objects is java.io.serializable.

EVS Domain Object	Description
Association	Relates a concept or a term to another concept or term. Association falls into three categories; concept association, term association, and synonyms, which are concept-term associations.
Atom	An occurrence of a term in a source.
AttributeSetDescriptor	set of concept attributes that should be retrieved by a given operation.
Definition	Textual definition from an identified source
DescLogicConcept	the fundamental vocabulary entity in the NCI Thesaurus.
EdgeProperties	Specifies the relationship between a concept and its immediate parent when a DefaultMutableTree is generated using the getTree method.
EditActionDate	Stores edit action and date information. This class is deprecated and will be removed from a future release. Please use History class instead.
History	Stores the concept history information.
HistoryRecord	Stores the DescriptionLogicConcept code.
MetaThesaurusConcept	fundamental vocabulary entity in the NCI MetaThesaurus
Property	an attribute of a concept. Examples of properties are "Synonym", "Preferred_Name", "Semantic_Type" etc.
Qualifier	Attached to associations and properties of a concept.
Role	Defines a relationship between two concepts.
SemanticType	a category defined in the semantic network that can be used to group similar concepts
Silo	A repository of customized concept terminology data from a knowledge base. There can be a single silo or multiple silos, each consisting of semantically related concepts and extracted character strings associated with those concepts.
SecurityToken	Stores security information for a Vocabulary.
Source	The source is a knowledge base.
TreeNode	Specifies the relationship between a concept and its immediate parent when a DefaultMutableTree is generated using the getTree method. This class is deprecated and will be removed from a future release. Please use EdgeProperties instead.
Vocabulary	Vocabulary entity or namespace.

Table 4.3 caCORE EVS domain objects and descriptions

EVS Data Sources

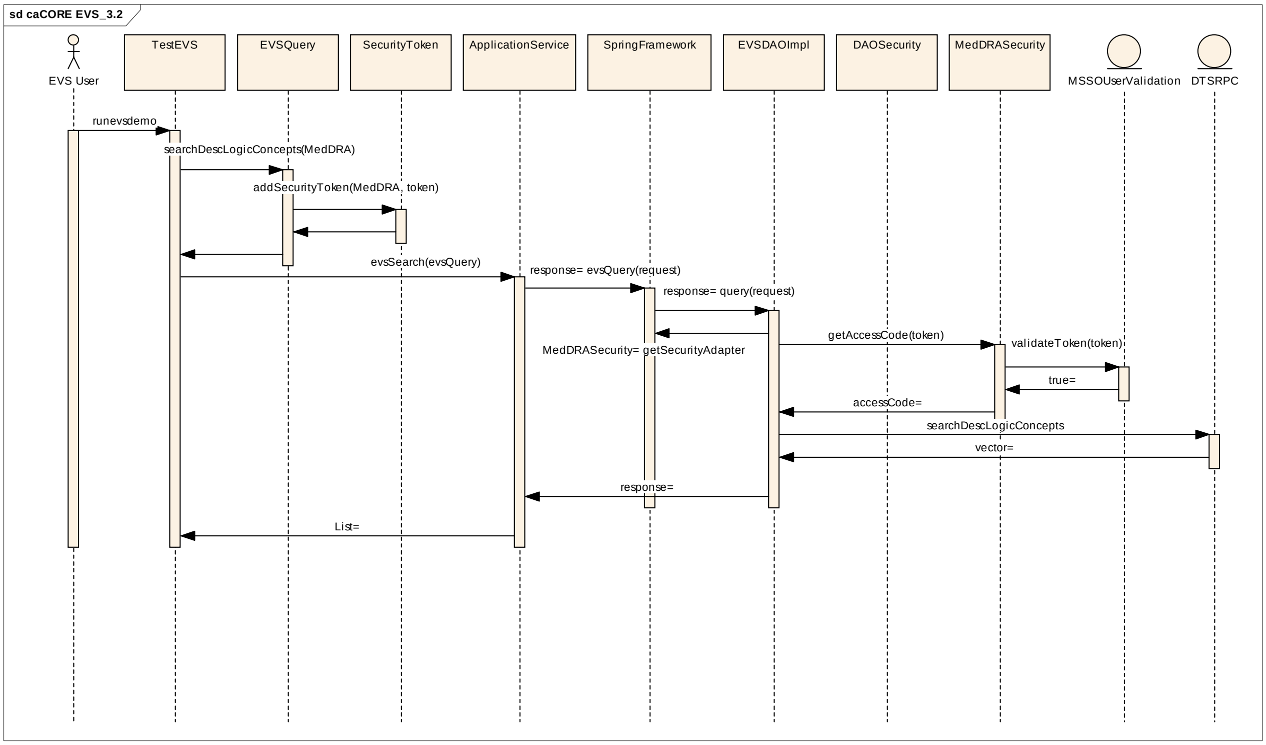
The EVS provides NCI with services and resources for controlled biomedical vocabularies, and includes both the NCI Thesaurus and the NCI Metathesaurus. The NCI Thesaurus is composed of over 27,000 concepts represented by about 78,000 terms. The Thesaurus is organized into 18 hierarchical trees covering areas such as Neoplasms, Drugs, Anatomy, Genes, Proteins, and Techniques. These terms are deployed by NCI in its automated systems for uses such as keywording and database coding.

The NCI Metathesaurus maps terms from one standard vocabulary to another, facilitating collaboration, data sharing, and data pooling for clinical trials and scientific databases. The Metathesaurus is based on the NLM's Unified Medical Language System (UMLS) and is composed of over 70 biomedical vocabularies.

Both the data tier and API for the backend terminology servers, the DTS serving the standalone vocabularies and the Metaphrase serving the NCI Metathesaurus, are commercial software products licensed from Apelon. Neither these backend server components nor the databases, or the schemas, are released in the caCORE distribution.

EVS Search Paradigm

The sequence diagram below provides an overview of the caCORE 3.2 EVS API search mechanism implemented to access the NCI EVS vocabularies.



An EVS search is performed by calling the *evsSearch* operation defined in the *ApplicationService* class.

```
List evsSearch(EVSQuery evsQuery);
```

EVSQuery and EVSQueryImpl

The gov.nih.nci.evs.query package consists of the *EVSQuery.java* interface and the *EVSQueryImpl.java* class. The methods defined in the `EVSQuery.java` file can be used to query the Metaphrase and DTS Servers. The query object generated by this class can hold one query at a time. The following example code segment demonstrates an *EVSQuery* object that calls the *searchDescLogicConcept* method.

Example:

```
String vocabularyName = "GO";
String conceptCode = "GO:0005667";
EVSQuery evsQuery = new EVSQueryImpl();
evsQuery.searchDescLogicConcept(vocabularyName,
conceptCode,true);
```

To perform a search on the Description Logic Vocabulary you must specify the vocabulary name. In most instances methods that do not require vocabulary names are NC MetaThesaurus queries.

EVSQuery Methods and Parameters

Most of the methods defined in the *EVSQuery* accept concept names or concept codes. If a vocabulary name is required as a parameter along with a concept code or name, a valid *DescLogicConcept* name or code needs to be passed to the search method.

Note: A search term is a String and is not considered as a valid concept name. To get a valid *DescLogicConcept* name you must perform a search using the *searchDescLogicConcept* method. Likewise to get a valid *MetaThesaurusConcept* name or *CUI* (Concept Unique Identifier) you must perform a search using the *searchMetaThesarus* method. Most of the search methods defined in the *EVSQuery* require a valid concept name or code.

Some of the methods defined in the EVSQuery are listed in the following table.

Method name	***Parameter***	***Comments***	***Returned by evsSearch***
searchDescLogicConcept	String vocabularyName	A valid Description Logic vocabulary name such as "NCI_Thesaurus", "GO", "HL7" etc	Returns one or more DescLogicConcepts in a List.
	String searchTerm	Any string value	
	int limit	Maximum number of records	

Table 4.4 Methods defined in the EVSQuery

Method name	Parameter	Comments	Returned by evsSearch
searchMetaThesaurus	String searchTerm	Any String value or a valid Concept Unique Identifier. A Concept Unique identifier is used to uniquely identify concepts in the MetaThesaurus.	Returns one or more MetaThesaurusConcepts in a List.
	int Limit	Maximum number of records	
	String Source	Source abbreviation. Each Source has a source abbreviation that can uniquely identify a source.	
	boolean Cui	This value is set to true if a concept unique identifier is used as a search term	
	boolean shortResponse	Set to true for short response	
	boolean score	Set to true for score	
getHistoryRecords	String vocabularyName	A valid Description Logic vocabulary name such as "NCI_Thesaurus", "GO", "HL7" etc	Returns one or more HistoryRecords in a List.
	String conceptCode	A valid code of a DescLogicConcept.	
getVocabularyNames			Returns one or more Description Logic vocabulary names in a List
getMetaSources			Returns one or more Source objects in a List.
searchSourceByCode	String code	A valid Atom code.	Returns one or more MetaThesaurusConcepts in a List.
	String sourceAbbreviation	Valid source abbreviation	

Table 4.4 Methods defined in the EVSQuery (Continued)

Method name	Parameter	Comments	Returned by evsSearch
getTree	String vocabularyName	A valid Description Logic vocabulary name	Returns a DescLogicConcept tree in a List
	String rootName	A valid DescLogicConcept name	
	boolean direction	Set to true if traverse down	
	boolean isaFlag	Set to true if relationship is child	
	int attributes	Sets a AttributeSetDescriptor value	
	int levels	Depth of the tree	
	Vector roles	Valid role names	

Table 4.4 Methods defined in the EVSQuery (Continued)

Accessing Secured Vocabularies

MedDRA is a Secured vocabulary for which a user must obtain a valid security token to access. The example below depicts the syntax of setting a security token to access a secured Vocabulary.

```
gov.nih.nci.evs.query.EVSQuery evsQuery = new
gov.nih.nci.evs.query.EVSQueryImpl();
gov.nih.nci.evs.security.SecurityToken token = new
gov.nih.nci.evs.security.SecurityToken();
/*
```

You must obtain a valid security token from NCICB to access MedDRA via the caCORE EVS API. Note that the security token value used in the following example is not valid.

```
*/
token.setAccessToken(“123456”);
evsQuery.addSecurityToken(“MedDRA”, token);
evsQuery.getDescLogicConcept(“MedDRA”, “Blood”, false);
```

Use the following instructions to create an EVS search request.

1. Create an *ApplicationService* instance.

   ```
   ApplicationService appService = ApplicationServiceProvider.getApplicationService()
   ```

2. Instantiate an *EVSQuery* instance and set the method name and parameters.

   ```
   EVSQuery evsQuery = new EVSQueryImpl();
   ```

   ```
   evsQuery.searchDescLogicConcepts("NCI_Thesaurus","blood*",10);
   ```

3. Set the security token value. This step can be omitted if the vocabulary does not require a security token.

   ```
   gov.nih.nci.evs.security.SecurityToken token = new
   gov.nih.nci.evs.security.SecurityToken();
   token.setAccessToken("xxxxxx");
   evsQuery.addSecurityToken(vocabularyName, token);
   ```

4. Call the *evsSearch* method defined in the *ApplicationService* class to query EVS.

   ```
   List evsResults = (List)appService.evsSearch(evsQuery);
   ```

5. The result objects are populated. The return type varies based on the search method call set in the *EVSQuery* instance.

Examples of Use

Example 1: Search for DescLogicConcepts Starting Term 'Blood'

```
1   public static void main(String[] args) {
2   try{
3   ApplicationService appService =
ApplicationServiceProvider.getApplicationService();
4  EVSQuery evsQuery = new EVSQueryImpl();
5
evsQuery.searchDescLogicConcepts("NCI_Thesaurus","blood*",10)
;
6   List evsResults = (List)appService.evsSearch(evsQuery);
…
7       }catch(ApplicationException ex){
   …
9       }
10   }
```

Lines	*Description*
3	Creates an instance of a class that implements the ApplicationService interface; this interface defines the service methods used to access data objects.
4	Creates a new EVSQuery object.
5	Specifies the search method and parameters. The searchDescLogicConept method performs a search in the "NCI_Thesaurus" vocabulary for a term that starts with "blood" and returns a maximum of ten Concepts if found.
6	Calls the evsSearch method of the ApplicationService implementation passing the EVSQuery object. This method returns a List Collection. The type of object that is returned depends on the search parameters set in the EVSQuery object; in this case the searchDescLogicConcept method was invoked, the resulting objects are of type DescLogicConcept.

Example 2: Search MetaThesaurusConcepts by Atom

```
1   try{
2   ApplicationService appService =
ApplicationServiceProvider.getApplicationService();
3   EVSQuery evsQuery = new EVSQueryImpl();
4   evsQuery.searchSourceByAtomCode("10834-0","*");
5   List evsResults = (List)appService.evsSearch(evsQuery);
6   for(int m=0; m<evsResults.size(); m++ ){
7   MetaThesaurusConcept concept =
(MetaThesaurusConcept)evsResults.get(m);
8   System.out.println("\nConcept code: "+concept.getCui()
+"\n\t"+concept.getName());
```

```
9   List sList = concept.getSourceCollection();
10   System.out.println("\tSource-->" + sList.size());
11   for(int y=0; y<sList.size(); y++){
12   Source s = (Source)sList.get(y);
13   System.out.println("\t - "+s.getAbbreviation());
21   }
14   List semanticList = concept.getSemanticTypeCollection();
15   System.out.println("\tSemanticType---> count ="+
semanticList.size());
16   for(int z=0; z<semanticList.size(); z++){
17   SemanticType sType = (SemanticType) semanticList.get(z);
18   System.out.println("\t- Id: "+sType.getId()+"\n\t- Name
: "+sType.getName());
19   }
20   List atomList = concept.getAtomCollection();
21   System.out.println("\tAtoms -----> count = "+
atomList.size());
22   for(int i=0;i<atomList.size(); i++){
23   Atom at = (Atom)atomList.get(i);
24   System.out.println("\t -Code: "+ at.getCode()+" -Name:
"+ at.getName()
25   +" -LUI: "+ at.getLui()+" -Source: "+
at.getSource().getAbbreviation());
26   }
27   List synList = concept.getSynonymCollection();
28   System.out.println("\tSynonyms -----> count = "+
synList.size());
29   for(int i=0; i< synList.size(); i++){
30   System.out.println("\t - "+ (String) synList.get(i));
31   }
32   }
33   }catch(ApplicationException ex){
34   …
35   }
```

Lines	*Description*
2	Creates an instance the ApplicationService.
3	Creates a new EVSQuery object.
4	Specifies the search method and parameters. The searchSourceByAtomCode method performs a search on all the sources specified in the MetaThesaurus for MetaThesaurusConcepts that has an Atom code value "10834-0 ". The source abbreviation specified is "*"; therefore all sources are searched for the Atom specified.
5	Calls the evsSearch method of the ApplicationService implementation passing the EVSQuery object. This method returns a List Collection. The type of object that is returned depends on the search parameters set in the EVSQuery object; in this case the searchSourceByAtomCode method was invoked, the resulting objects are of type MetaThesaurusConcept.
6	Traverse through the result set.

Lines	Description
7	Cast the result object to a MetaThesaurusConcept.
6-31	Prints the attributes and association values of the MetaThesaurusConcept.

Downloading the NCI Thesaurus

The NCI Thesaurus can be downloaded in several formats, including simple tab-delimited ASCII format, Apelon's proprietary Ontylog XML format, and OWL format (the Web Ontology Language). The ASCII- and XML-formatted files are available for download at the NCICB download site, as ThesaurusV2_0Flat.zip and ThesaurusV2_0XML.zip. The OWL formatted version is available at http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl. Users who prefer to use FTP for download can go to the caCORE FTP site.

The format of the ASCII flat file is extremely simple. For each concept, the download file includes the following information:

1. The concept code: all terms have the "C" prefix, followed by its integer index;
2. The concept name: this name may contain embedded punctuation and spaces;
3. A pipe-delimited list of parent concepts, as identified in the NCI Thesaurus by *isa* relations;
4. A pipe-delimited list of synonyms, the first of which is the preferred name; and
5. One of the NCI definitions for the term-if one exists.

Each of these separate types of information is tab-delimited; within a given category, the individual entries are separated by pipes ("|"). Only the third and fourth categories, i.e., the parent concepts and synonyms, have multiple entries requiring the pipe separators. Note that while much of the information available from the interactive Metaphrase server is included in the download, any information outside the NCI Thesaurus description logic vocabulary (e.g., Diagnosis, Laboratory, Procedures, etc.) is not.

For example, the flat file download for the term "*Mercaptopurine*" is as follows:

```
C6 Mercaptopurine Immunosuppressants|Purine Antagonists
Mercaptopurine|1,3-AZP|1,7-Dihydro-6H-purine-6-thione|3H-
Purine-6-thiol|6
Thiohypoxanthine|6 Thiopurine|6-MP|6-Mercaptopurine|6-
Mercaptopurine
Monohydrate|6-Purinethiol|6-Thiopurine|6-Thioxopurine|6H-
Purine-6-thione,
1,7-dihydro- (9CI)|6MP|7-Mercapto-1,3,4,6-
tetrazaindene|AZA|Alti-
Mercaptopurine|Azathiopurine|BW 57-323H|CAS
50442|Flocofil|Ismipur|Leukerin|Leupurin|MP|Mercaleukim|Merca
leukin|Mercap|Me
rcaptina|Mercapto-6-
purine|Mercaptopurinum|Mercapurin|Mern|NCI-C04886|NSC
755|Puri-Nethol|Purimethol|Purine-6-thiol (8CI)|Purine-6-
thiol
Monohydrate|Purine-6-thiol,
Monohydrate|Purinethiol|Purinethol|U-4748|WR-2785
```

```
An anticancer drug that belongs to the family of drugs called
antimetabolites.
```

Users who have access to the Apelon Ontylog software may wish to download the XML encoded file. All other users who prefer to use an encoded format rather than the simple ASCII form should download the OWL encoding of the NCI Thesaurus, which is described below.

OWL Encoding of the NCI Thesaurus

OWL, as specified and proposed by the World Wide Web Consortium (W3C), is an emerging standard for the representation of semantic content on the web. Building on the earlier groundwork laid by XML, the Resource Description Framework (RDF) and RDF schema; and subsequently, by DAML+OIL, OWL represents the culmination of what has been learned from these previous efforts.

While XML provides surface syntax rules and XML Schema provides methods for validating a document's structure, neither of these can in itself impose semantic constraints on how a document is interpreted. RDF provides a data model for specifying objects (resources) and their relations, and RDF Schema allows one to associate properties with the individual resources as well as taxonomic relations among the objects. Yet even these extensions could not provide the breadth and depth of representation needed to encode nontrivial real-world information. OWL adds vocabulary for describing arbitrary nonhierarchical relations between classes, cardinality constraints, resource equivalences, richer typing of properties, and enumerated classes.

A major focus of the W3C is the establishment of the The Semantic Web-a far-reaching infrastructure whose purpose is to provide a framework whereby autonomous self-documenting agents and web services can exchange meaningful information without human intervention. OWL is the first step towards realizing this vision. As a result of collaborative efforts with Dr. James Hendler and the University of Maryland, the NCI Thesaurus is now available for download in OWL format; this section describes the mapping of the NCI Thesaurus to OWL.

The mapping of the NCI Thesaurus into OWL format proceeds via the Ontylog XML elements declared in Apelon's Ontylog DTD. The four basic elements are *Kinds*, *Concepts*, *Roles*, and *Properties*, where:

- Kinds are the top-level super classes in the Thesaurus; they enumerate the different possible categories of all concepts, and include such things as Anatomy, Biological Processes, Chemicals and Drugs, etc. *Each NCI Thesaurus Kind is converted to an owl:Class*.
- An NCI Thesaurus Concept describes a specific concept under one of the Kind categories. *Each NCI Thesaurus Concept is converted to an owl:Class*.
- Roles capture how concepts relate to one another. Generally, Roles have restricted domains and ranges, that limit the sets of concepts which can participate in the Role according to their categories-i.e., Kinds. The "defining roles" within a concept definition provide these local restrictions on the ranges of roles. *Each NCI Thesaurus Role is converted to an owl:ObjectProperty.*
- NCI Thesaurus Properties encode the attributes that pertain to a class; they contain metadata that describes the class, but not its instantiations or subclasses. *Each NCI Thesaurus Property is converted to an owl:AnnotationProperty.*

The bulk of the Thesaurus comprises concept definitions; this is also where the most complex semantics occur. Each concept in the Thesaurus has three main types of associated data: defining concepts, defining roles, and properties. A "defining concept" is essentially a super class; the defined concept in OWL has an *rdfs:subClassOf* relationship to the defining concept.

The defining roles and properties are mapped as described above; the *owl:AnnotationProperty* is actually a subclass of *rdf:Property*, and, like *rdfs:comment* and *rdfs:label*, can be attached to any class, property or instance. This allows properties from the Thesaurus to be associated directly with a concept's corresponding class, without violating the rules of OWL.

In addition to any explicitly named properties, each element in the Thesaurus also has a uniquely defined "code" and "id" attribute associated with it. These are used as unique identifiers in the Apelon development software, and, as such, are not defined explicitly as roles or properties. In mapping these identifying attributes to OWL, we have treated these as special cases of the explicit property elements, and just like other properties in the Thesaurus, they are mapped as owl:AnnotationProperties. Table *4.5* summarizes the mapping of elements in the Ontylog DTD to OWL elements.

Ontylog Name Conversion

In mapping to OWL, all Ontylog concept *names* must be converted to proper RDF identifiers (rdf:id) following the RDF naming rules. This is achieved by removing any spaces in the original names and substituting all illegal characters with underscores. Names that begin with numbers are also prefixed with underscores to make them legal. The original concept name however, is preserved as an rdfs:label. The following steps summarize the conversion of names:

1. Any "+" characters are replaced with the text "plus."
2. All role names are prefixed with an "r" to ensure that roles and properties with the same name do not clash.
3. Any characters that are not alphanumeric, or one of "-" and "_," are replaced with an underscore ("_").
4. All names with leading digits are prefixed with an underscore.
5. Multiple adjacent underscores in the corrected name are replaced with a single underscore.

Ontylog Element	***Owl Element***	***Comment***
kindDef	owl:Class	
roleDef	owl:ObjectProperty	
propertyDef	owl:AnnotationProperty	
conceptDef	owl:Class	
name*	rdf:ID	Applies to the name subelement of *kindDef*, *roleDef*, *propertyDef*, and *conceptDef*.*

Table 4.5 Ontylog DTD to OWL conversions

Ontylog Element	***Owl Element***	***Comment***
name	rdfs:label	Because the *conceptDef name* contains some useful semantics, the original form is retained as an *rdfs:label*. No other name elements are retained in *rdfs:label*.
code	owl:AnnotationProperty	Defined as an *owl:AnnotationProperty* with *rdf:ID*="code". Code values remain the same for each concept.
id	owl:AnnotationProperty	Defined as an *owl:AnnotationProperty* with *rdf:ID*="ID". ID values remain the same for each concept.
definingConcepts	rdfs:subClassOf	The *concept* subelement of *definingConcepts* is mapped to the *rdf:resource* attribute of the *rdfs:subClassOf* element.
domain	rdfs:domain	
range	rdfs:range	
definingRoles / role / name	owl:onProperty	*definingRoles* are converted to owl restrictions on properties. The *name* child element of *definingRoles/role* is taken as the *rdf:resource* attribute of the *owl:onProperty* element.
definingRoles / role / value	owl:someValuesFrom	*definingRoles* are converted to owl restrictions on properties. The *value* child element of *definingRoles/role* is taken as the *rdf:resource* attribute of the *owl:someValuesFrom* element.

Table 4.5 Ontylog DTD to OWL conversions (Continued)

* name Ontylog elements are converted to rdf:ID as described in the Ontylog Name Conversion section. namespaceDef and namespace elements are not mapped to OWL.

Additional information about the Ontylog encoding is available in the Ontylog DTD, which can be downloaded from the NCICB EVS FTP site, along with the zipped ASCII flat file and the Ontylog XML encoding. The current OWL translation of the NCI Thesaurus contains over 500,000 triples and is available in zipped format from the FTP site, as well as in unzipped format at http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl, the mindswap web site for download or online viewing.

Ontylog Mappings

Mapping of Gene Ontology to Ontylog

The NCI DTS provides access to the Gene Ontology^{TM} Consortium's (GO) controlled vocabulary. The GO ontologies are widely used-most likely due to their simplicity of design and their potential for automated transfer of biological annotations, from model organisms to more complex organisms based on sequence similarities. GO comprises three independent controlled vocabularies (ontologies) encoding biological process, molecular function, and cellular components for eukaryotic genes. GO terms are connected via two relations, *is-a* and *part-of*, that define a directed acyclic graph. Although concepts in the ontologies were initially derived from only three model systems (yeast, worm, and fruitfly), the goal was to encode concepts in such a way that the information is applicable to *all* eukaryotic cells. Thus, species-specific anatomies are not represented, as this would not support a unifying reference for species-divergent nomenclatures.

Each month NCI will load the latest version of GO into a test instance of the DTS server, and, following validation in the Ontylog environment, will promote it to a production server for programmatic access by NCI applications. NCI converts GO into the Ontylog XML representation (necessary for import into the DTS server) via a stylesheet transformation followed by some post-processing to satisfy Ontylog constraints. It is NCI's intent that the version of GO on the DTS server will not be more than a month behind the current version available from http://www.geneontology.org. However, it might be necessary to skip releases if unforeseen complications arise.

The tables in this section summarize the encoding of GO elements into Ontylog.

Ontylog Entity	*Instance Name (and optional description)*
namespaceDef	**GO**
kindDef	**GO_Kind**
RoleDef	**part-of:** This role is unused; however, the software requires that at least one role be declared.
propertyDef	**Preferred_Name**
propertyDef	**Synonym**
propertyDef	**DEFINITION**
propertyDef	**dbxref:** complex property containing two XML-marked up GO entities: "go:database_symbol," and "go:reference," using tags "database_symbol" and "reference," respectively.
propertyDef	**part-of:** complex property containing two XML-marked up GO entities: "go:name" and "go:accession," using tags "go-term" and "go-id," respectively.

Table 4.6 Ontylog elements used for GO mapping

The go:name stored in Preferred_Name is as declared in GO. However, the go:name used in the Ontylog name might have been modified during the conversion process (by appending underscores) to make the Ontylog name unique.

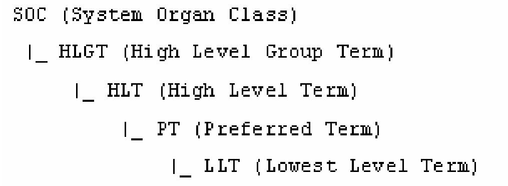
GO term element	conceptDef element	(propertyDef)
go:accession	code	
go:name	name	
go:isa	definingConcepts	
go:name	property	Preferred_Name
go:synonym	property	Synonym
go:definition	property	DEFINITION
go:part-of	property	part-of
go:dbxref	property	dbxref

Table 4.7 Mapping of GO term to Ontylog conceptDef

Mapping of MedDRA to Ontylog

Vocabulary Hierarchy Structure

The Ontylog version of MedDRA reflects the native hierarchy, with terms organized according to their term type as shown in *Figure 4.5*.



```
SOC (System Organ Class)
 |_ HLGT (High Level Group Term)
     |_ HLT (High Level Term)
         |_ PT (Preferred Term)
             |_ LLT (Lowest Level Term)
```

Figure 4.5 Hierarchy of MedDRA

The Special Search Categories (SSC) are under the concept *AssociativeTerm-Group(SSCs)*, which has been created by the EVS as a header concept for the SSC terms to be grouped together. All the System Organ Class (SOC) concepts as well as the top header concept for the SSCs are under the *MedDRA[V-MDR]* root node. Although Low Level Terms (LLTs) can have any type of relationship to their Preferred Term (PT) (for example, a synonym of the PT), the Ontylog version presents them all as children concepts. The *Associative Term Group (SSCs)* concept has a special code and term type not found in MedDRA to distinguish it from other terms in the vocabulary.

Concept Codes and Names

The concept name is created from the MedDRA term followed by the MedDRA code enclosed in brackets. The Ontylog concept name must be unique so including the code in the name guarantees uniqueness. For display purposes, the property `Preferred_Name` should be used instead of the concept name; it contains the unadorned MedDRA term. The Ontylog concept code is the MedDRA code.

Roles

A single role has been defined. The role has_associated_term is utilized to relate SSC top level categories with their associated PT terms. All the concepts in the vocabulary are primitive.

Properties

The properties defined for MedDRA 6 in Ontylog are shown in Table *4.8*; their provenance in the MedDRA distribution is indicated.

Ontylog Property	***MedDRA Entity***
Code_in_Source	MedDRA code (llt_code, pt_code, hlt_code, and so forth)
Cross-reference, cross reference to WHOART	COSTART, ICD9-CM, and so forth
Descriptor_ID	pt_code of an LLT
MedDRA_Abbreviation	soc_abbrev, spec_abbrev
NCI_META_CUI	-
Preferred_Name	MedDRA name (llt_name, pt_name, hlt_name, and so forth)
Primary_SOC	pt_soc_code of a PT
Serial_Code_International_SOC_Sort_Order	intl_ord_code
Term_Type	-
UMLS_CUI	-

Table 4.8 Properties defined in the Ontylog version of MedDRA. Properties that are not derived directly from MedDRA have a dash in the MedDRA Entity column.

Of the MedDRAderived properties, only Cross-reference is not a straightforward name-value pair. This property has subfields encoded in xml; the xml elements are source and sourcecode, where the sourcecode contains a code or symbol assigned by an external vocabulary source to a specific term.

Two properties are not derived from MedDRA: NCI_META_CUI and UMLS_CUI. These properties contain the Concept Unique Identifier (CUI) of concepts in the NCI Metathesaurus containing MedDRA terms. The property name indicates whether the CUI is assigned to the concept by Unified Medical Language System (UMLS) or by NCI. The Term_Type property is indirectly derived from MedDRA and indicates the hierarchy level of a term with the term types as shown in *Figure 4.5*; in addition, the term type for Obsolete Lower Level Terms (OLLT) is also used.

Mapping of MGED Ontology to Ontylog

The native MGED Ontology (MO) is edited in OilEd and distributed in the Defense Advanced Research Projects Agency (DARPA) Agent Markup Language (DAML) + Ontology Inference Layer (OIL) XML format. DAML+OIL can be converted to the Ontylog Description Logic (DL) in a relatively straightforward manner. However, some valid DAML+OIL constructions cannot be represented in Ontylog DL, including enumerations and specific combinations of ObjectProperties that result in classification cycles in

Ontylog. In MO version 1.1.9, two ObjectProperties have been asserted near the top of the hierarchy on the MGEDCoreOntology class. On conversion to Ontylog these assertions generate classification cycles, however, the data cannot be massaged as was done in preliminary conversions with previous versions of MO because the fix would have required modifications to every converted concept. Consequently, beginning with MO v 1.1.9, all the ObjectProperties in DAML+OIL are converted to Ontylog properties (rather than Ontylog roles), which are annotations ignored by the classifier.

Vocabulary Hierarchy Structure

The MO class hierarchy structure is preserved in the Ontylog conversion. One minor difference is that MO class instances are also represented in the Ontylog concept hierarchy (since there is no distinction between classes and instances in Ontylog). A nonMO top level concept, OrphanConcepts, has been added in the Ontylog representation to hold MO instances of Thing.

Concept IDs, Codes, and Names

MO classes and instances are identified solely by their name; no codes or numeric IDs are assigned. For the conversion to Ontylog, MO class or instance names are retained as concept names. As Ontylog concepts also require unique codes and IDs, a code and an ID are created during the conversion. The ID reflects the position of the class or instance in the XML tree. The code is derived from the ID by adding an “X-MO-” prefix to it; therefore, the code is not guaranteed to remain invariant from version to version of the MO. A mapping table is made available whenever the MO is updated.

Roles

No roles have been defined; all the concepts are primitive.

Properties

All the object and datatype properties defined in MO have been converted to Ontylog properties. With the exception of has_reason_for_deprecation and has_database, all the properties have been 'manually” propagated to children concepts in the database in order to mimic the expected role inheritance. In addition, new properties have been defined as shown in Table *4.9*.

Ontylog Property	***MGED Ontology Entity***
DEFINITION	*rdfs:comment* value
Preferred_Name	*rdf:about* value
Synonym	*rdf:about* value
Concept_Type	-

Table 4.9 New properties defined in the Ontylog version of the MGED Ontology and their provenance (if applicable) in the daml+oil file

The `Preferred_Name` property is recommended for display purposes, while `Synonym` is recommended for searches by dependent applications (even though the value of both properties is the same, the EVS tries to maintain a certain consistency in the usage of properties for the benefit of all users). The `Concept_Type` property holds one of two values: `mged_class`, or `mged_instance`.

CHAPTER 5

CANCER DATA STANDARDS REPOSITORY

This chapter describes the Cancer Data Standards Repository (caDSR) and its application programming interface.

Topics in this chapter include:

Introduction

The Cancer Data Standards Repository (caDSR) at NCI is part of a larger effort associated with the 11179 standard defined by the ISO (International Organization for Standardization) and IEC (International Electrotechnical Commission). The purpose of the ISO/IEC 11179 is to regularize the metadata used in representing and annotating shared electronic data. This chapter describes the Java API to the repository and introduces the Java classes that participate in this programmatic interface. The first two sections provide a brief review of the ISO/IEC 11179 standard and its realization as an Oracle database at NCI. The caDSR is a conforming implementation of ISO/IEC 11179 Edition 2 with extensions, though some of the attributes in the ISO 11179 pertaining to Stewardship/Submission and Registrar are not exposed in the user interfaces.

Modeling Metadata: The ISO/IEC 11179 Standard

Regardless of the application domain, any particular data item must have associated with it a variable name or tag, a conceptualization of what the item signifies, a value, and an intended interpretation of that value. For example, an entry on a case report form may be intended to capture the patient's place of birth, and the corresponding value may be tagged electronically as *Patient_placeOfBirth*. But what is the intended concept? Is the data element designed to capture the country, the city, or the specific hospital where the person was born? Assuming that the intended concept is country, how is the resulting value to be represented electronically? Possible representations might include the full name of the country, a standard two- or three-letter abbreviation, a standard country code, or perhaps a specific encoding unique to the application.

Metadata is "data about data," and refers to just this type of intentional information that must be made explicit in order to ensure that electronically exchanged data can be correctly interpreted. The purpose of the ISO/IEC 11179 standard is to define a framework and protocols for how such metadata can be specified, consistently maintained, and shared across diverse domains. The caDSR conforms to this standard; while it contains extensions developed specifically to support registration of forms used in clinical trials data management, usage of the caDSR is not limited to clinical applications.

The ISO/IEC 11179 standard defines a fairly complex meta-model; even the notion of metadata itself is a rather abstract concept. To facilitate understanding the model, this discussion uses a divide-and-conquer approach, and defines two very general types of components:

1. Information components whose purpose is to represent content; and
2. Organizational and administrative components whose purpose is to manage the repository.

This partitioning is not intrinsic to the ISO/IEC 11179, and indeed, some of the components do not neatly fit into the separate categories. Nevertheless, it provides a useful framework.

The fundamental information component in the ISO/IEC 11179 model is the *data element*, which constitutes a single unit of data considered indivisible in the context in which it is used. Another way of saying this is that a data element is the smallest unit of information that can be exchanged in a transaction between cooperating systems. More specifically, a data element is used to convey the *value* of a selected *property* of a defined *object*[2], using a particular *representation* of that value.

A critical notion in the metadata model is that any concept represented by a data element must have an explicit definition that is independent of any particular representation. In order to achieve this in the model, the ISO/IEC 11179 standard specifies the following four components:

1. A *DataElementConcept* consists of an *object* and a selected *property* of that object;
2. The *ConceptualDomain* is the set of all intended meanings for the possible values of an associated *DataElementConcept*;

[2] The term *object* is used here in the sense defined by the ISO/IEC 11179 (see definition in *Table 5.1* 1) and does not have any literal correspondence to a caCORE Java object.

3. The *ValueDomain* is a set of accepted representations for these intended meanings; and
4. A *DataElement* is a combination of a selected *DataElementConcept* and a *ValueDomain*.

The example in *Figure 5.1* diagrams these definitions.

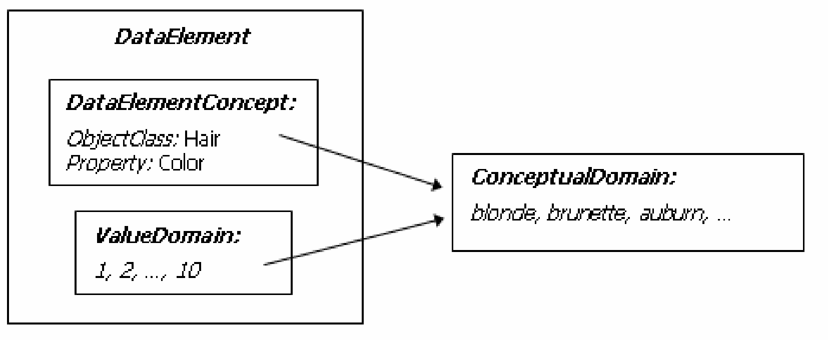


Figure 5.1 Representing data in the ISO/IEC 11179 model

Figure 5.1 shows a *DataElement* that might be used to represent hair color. The associated *DataElementConcept* uses the *ObjectClass* `Hair` and the *Property* `Color` to define the intended concept. The intended meanings for this data element are the familiar hair colors blonde, brunette, etc., but the *ValueDomain* uses a numeric representation that is mapped to these intended meanings. Both the *DataElementConcept* and the *ValueDomain* are components of the *DataElement*, and each references the same *ConceptualDomain*, which is defined outside the *DataElement*. Important principles of this design are:

- The *DataElementConcept* (DEC) is used to signify a concept *independent of representation*.
- The *ValueDomain* (VD) specifies a set of representational values *independent of meaning*.
- The *DataElement* (DE) combines a specific object and property with a value representation.
- The *ConceptualDomain* (CD) specifies the complete set of value meanings for the concept and allows the interpretation of the representation.

Figure 5.2 uses a UML Class diagram to show the cardinality constraints that hold for these relations.

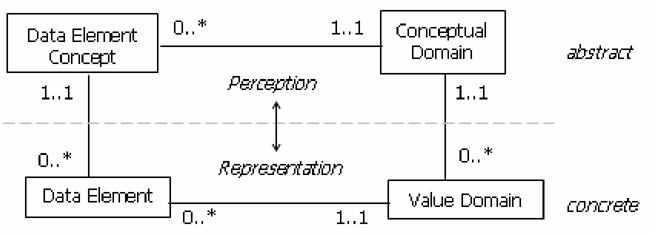


Figure 5.2 Abstract and concrete components of the data representation

Each *DataElement* must specify exactly one *DataElementConcept* and one *ValueDomain*, in order to fully specify the data element. Similarly, each *DataElementConcept* and *ValueDomain* must specify exactly one *ConceptualDomain*. Conversely, a *Concep-*

tualDomain may be associated with any number of *ValueDomains* and any number of *DataElementConcepts*.

Figure 5.3 shows an example of this, using the `color` property of different geometric objects as *DataElementConcepts*, and alternate color representations for the *ValueDomains*.

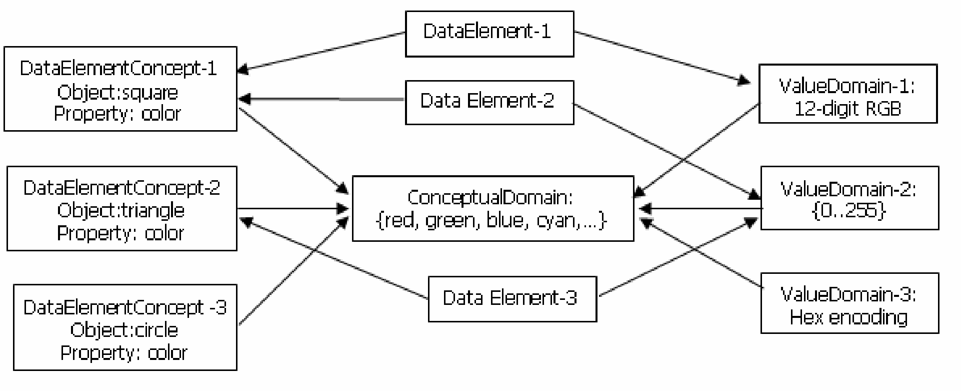


Figure 5.3 Many-to-one mappings of information elements in the metadata model

A constraint not shown in any of these figures is that it is not possible to reuse the same *DataElementConcept-ValueDomain* pair to define a new *DataElement*, as this defines a logical redundancy. Thus, the "0..*" cardinality constraints implied by Figure 5.2 are not quite as open-ended as they imply. Specifically,

- a *DataElement* specifies exactly one *DataElementConcept* and one *ValueDomain*;
- a *DataElementConcept* specifies exactly one *ConceptualDomain*;
- a *ValueDomain* specifies exactly one *ConceptualDomain*;
- a *ConceptualDomain* may be associated with any number of *ValueDomains*;
- a *ConceptualDomain* may be associated with any number of *DataElementConcepts*;
- a *DataElementConcept* may be associated with as many *DataElements* as there are *ValueDomains* (i.e. alternate representations) associated with the *ConceptualDomain*; and
- a *ValueDomain* may be associated with as many *DataElements* as there are *DataElementConcepts* associated with the *ConceptualDomain*. Many additional information components collaborate with these four core elements to provide the ISO/IEC 11179 infrastructure for content representation. These are described in the caDSR model in the next section, along with the organizational and administrative components that are used to document, classify, and in general, manage the information components.

caDSR Metamodel

Figure 5.4 again shows the four elements discussed thus far, but this time in the context of other components that collectively define the infrastructure for content representation.

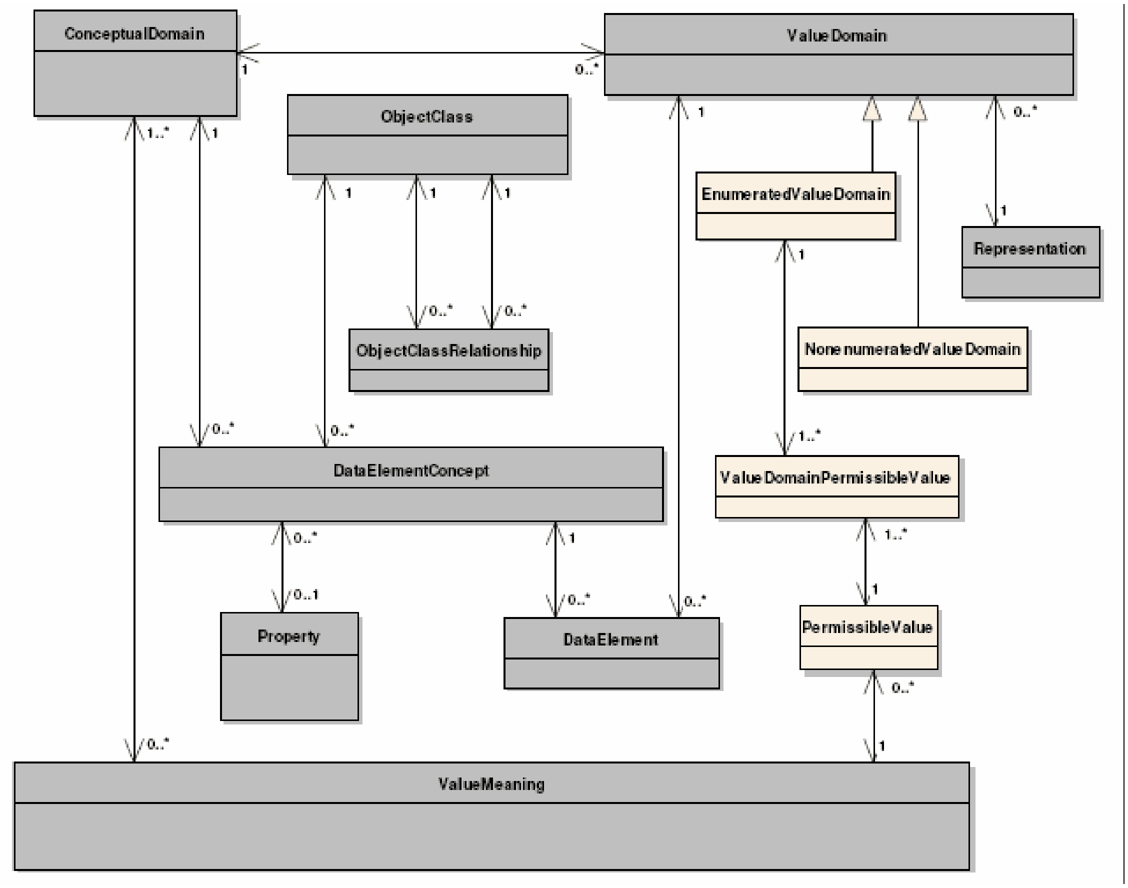


Figure 5.4 Information classes in the caDSR metamodel

All of the components in *Figure 5.4* that are highlighted in light gray are Administered Components and must be *administered*. Pragmatically, this means that there is a formal protocol for creating these components; that there is an approval process in place for accepting newly proposed elements; and that there is a designated authority in charge of stewarding the component. Technically, this means that each of the highlighted components is derived from a parent class named *AdministeredComponent*. *Table 5.1* provides definitions for the new components introduced in *Figure 5.4*.

Component Name	***Definition***	***Administered Component***
Conceptual-Domain	A subclass of Administered Component that depicts the set of all valid value meanings of a Data Element Concept expressed without representation.	Yes

Table 5.1 Information classes in the caDSR metamodel

Component Name	**Definition**	**Administered Component**
DataElement	A subclass of Administered Component that depicts a unit of data for which the definition, identification, representation, and permissible values are specified by means of a set of attributes.	Yes
DataElementConcept	A subclass of an Administered Component that depicts a characteristic of something in the real world that can be represented in the form of a data element, independent of any particular representation.	Yes
ObjectClassRelationship	An affiliation between two instances of ObjectClass.	
EnumeratedValueDomain	A subtype of value domain expressed as a list of all permissible values.	
NonenumeratedValueDomain	A subtype of value domain expressed by a generative rule or formula; for example: "all even integers less than 100."	
ObjectClass	A subclass of an Administered Component that depicts a set of ideas, abstractions, or things in the real world that can be identified with explicit boundaries and meaning and whose properties and behavior follow the same rules.	Yes
PermissibleValue	The exact names, codes, and text that can be stored in a data field in an information management system.	
Property	A A subclass of an Administered Component that depicts a characteristic common to all members of an Object Class. It may be any feature naturally used to distinguish one individual object from another. It is conceptual and thus has no particular associated means of representation.	Yes
Qualifier	A term that helps define and render a concept unique. For example, given the ObjectClass `household` and the Property `annual income`, a Qualifier for the Property could be used to indicate `previous year`. One or more Qualifiers can be present for ObjectClass or Property.	

Table 5.1 Information classes in the caDSR metamodel (Continued)

Component Name	Definition	Administered Component
Representation	A subclass of Administered Component that depicts a mechanism by which the functional and/or presentational category of an item may be conveyed to a user. Examples: 2-digit country code, currency, YYYY-MM-DD, etc.	Yes
ValueDomain	A subclass of an Administered Component that describes the attributes of a set of permissible values for a data element.	Yes
ValueDomain-Permissibl-eValue	The many-to-many relationship between value domains and permissible values; allows one to associate a value domain to a permissible value.	
ValueMeaning	The significance or intended meaning of a permissible value.	Yes

Table 5.1 Information classes in the caDSR metamodel (Continued)

An *AdministeredComponent* is literally a component for which administrative information must be recorded. It may be a *DataElement* itself or one of its associated components (*Representation*, *ValueDomain*, *DataElementConcept*, *ConceptualDomain*, *ObjectClass*, or *Property*) that requires explicit specifications for reuse in or among enterprises-an *AdministeredComponent* is a generalization for all of the descendant components that are highlighted in *Figure 5.4*. *Table 5.2* lists the class attributes of the Administration Record common to all *AdministeredComponents*.

Attribute Name	Type	Required/ Options	caDSR UI Name
id (Public ID)	String	required	
shortName	String	required	
preferredDefinition	String	required	
longName*	String	required	
version	Float	required	
workflowStatusName	String	required	
workflowStatusDescription	String	required	
latestVersionIndicator	Boolean	required	
beginDate	Date	required	
endDate	Date	required	
deletedIndicator	Boolean	optional	
changeNote	String	optional	

Table 5.2 Class attributes of an AdministeredComponent

Attribute Name	Type	Required/ Options	caDSR UI Name
unresolvedIssue	String	optional	
origin	String	optional	
dateCreated	Date	required	
dateModified	Date	required	
registration (Registration Status)	String	optional	

Table 5.2 Class attributes of an AdministeredComponent

***Note**: The longName in caDSR equates to the ISO/IEC 11179 "PreferredName".

The attributes listed in *Table 5.2* tell only part of the story; additional critical information about an *AdministeredComponent* derives from its associations with the organizational and administrative items depicted in *Figure 5.5*. Of these components, the only element that is also itself an Administered Component is the *ClassificationScheme*.

Three "regions" are outlined in *Figure 5.5*: (1) the Naming and Identification region (upper right), (2) the Classification region (lower left), and (3) the Contact Information section (bottom center). The *ReferenceDocument* component is not included in either region. Each *AdministeredComponent* may be associated with one or more *Reference-*

Documents that identify where and when the component was created and provide contact information for the component's designated registration authority.

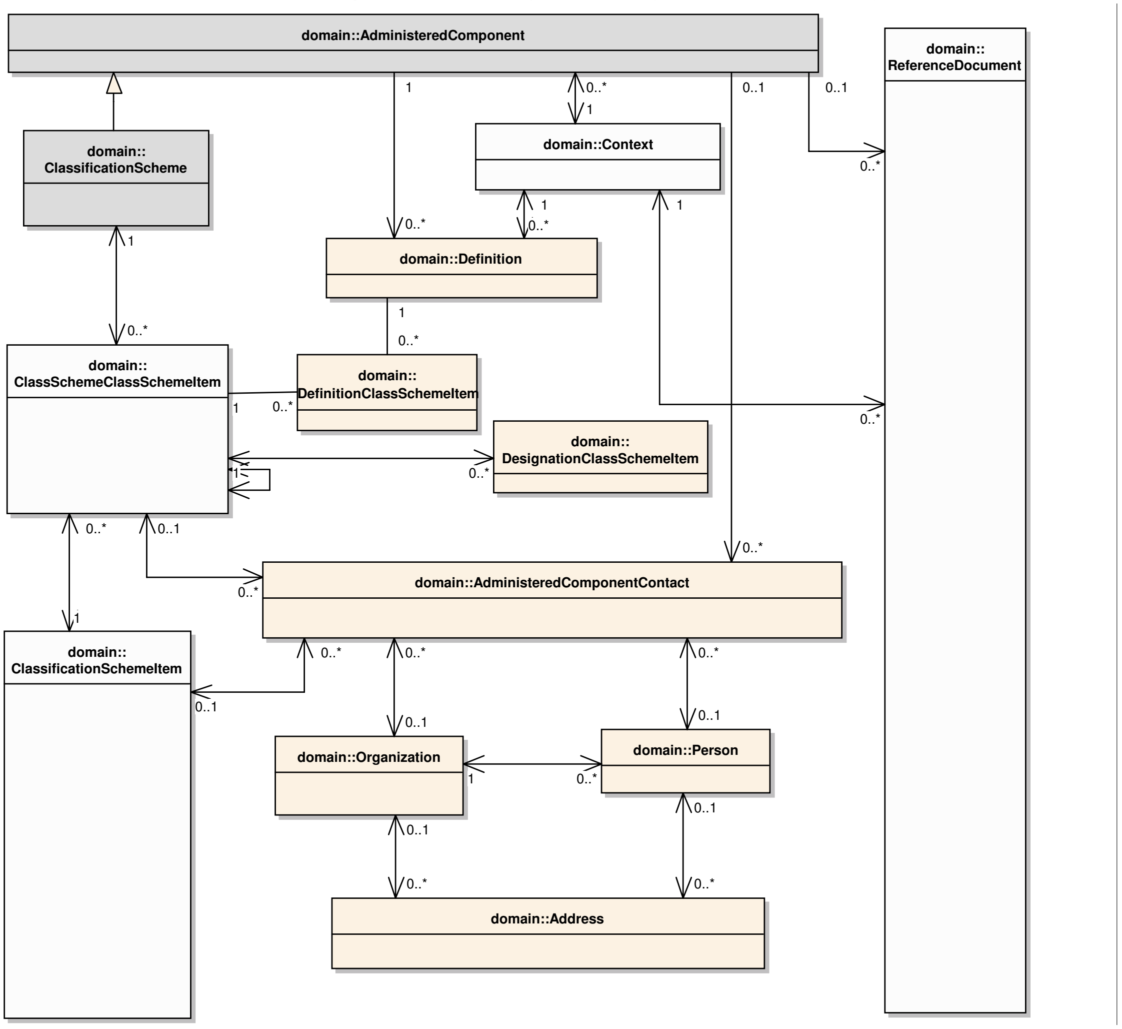

domain::AdministeredComponent
domain::ReferenceDocument
domain::ClassificationScheme
domain::Context
domain::Definition
domain::ClassSchemeClassSchemeItem
domain::DefinitionClassSchemeItem
domain::DesignationClassSchemeItem
domain::AdministeredComponentContact
domain::ClassificationSchemeItem
domain::Organization
domain::Person
domain::Address
1
0..*
0..1

Figure 5.5 Administrative and organizational components of the caDSR metamodel

The purpose of the Naming and Identification region is to manage the various names by which components are referenced in different contexts. Many components may be referenced by different names depending on the discipline, locality, and technology in which they are used. In addition to the name attributes contained in the component itself (*shortName*, *longName*), an administered component may have any number of alternative *Designations*. Each *Designation* is associated with exactly one *Context* reflecting its usage, and may be classified by one or more Classification Scheme Items.The Classification region is used to manage classification schemes and the administered components that are in those classification schemes. Classification is a fundamental and powerful way of organizing information to make the contents more accessible. Abstractly, a classification *scheme* is any set of organizing principles or dimensions along which data can be organized. In the ISO/IEC 11179 model, a *ClassificationScheme* may be something as simple as a collection of keywords or as complex as an ontology. The classification scheme element in Figure 5.5 is highlighted in light gray to reflect that it is an administered component.

Classification schemes that define associations among components can greatly assist navigation through a large network of elements; the associations may describe simple subsumption hierarchies or more complex relations such as causal or temporal relations. In particular, classification schemes with inheritance can enhance self-contained definitions by contributing the definition of one or more ancestors.

The *ClassificationScheme* component serves as a container-like element that collects the *ClassificationSchemeItems* participating in the scheme. In addition, the *ClassificationScheme* component identifies the source of the classification system and contains an indicator specifying that the scheme is alphanumeric, character, or numeric.

A *ClassificationSchemeItem* may be a node in a taxonomy, a term in a thesaurus, a keyword in a collection, or a concept in an ontology-in all cases, it is an element that is used to *classify* administered components. It is quite natural for an administered component that is used in different contexts to participate in several classification schemes. Classification schemes may coexist and a classified component may have a different name in each one, since each scheme is from a different context.

The *ClassSchemeClassSchemeItem* in the caDSR model is not a component of the ISO/IEC 11179 metamodel, but serves an important role in the implementation of the many-to-many mappings between *ClassificationSchemeItems* and *Classification-Schemes*. This component is used to associate a set of classification scheme items with a particular classification scheme, and to store details of that association such as the display order of the items within that scheme.

The *Organization*, *Person*, and *Address* classes constitute the Contact region of Figure 5.5. A *Person* or an *Organization* can be the contact for an *AdministeredComponent* and either can be reached at an *Address*. In particular:

- An *Organization* or *Person* may be the contact for multiple *AdministeredComponents*. Each *AdministeredComponent* may have only a single *Person* or *Organization* as its contact.
- An *Organization* may have one or more *Persons* recorded as members. A *Person* may be a member of only one organization in the current caDSR model.
- The *Organization* and *Person* classes hold basic identifying information, such as names, and for *Persons*, that person's position name.
- Both *Organizations* and *Persons* may have one or more *Addresses*. An *Address* has the attributes necessary to record a postal service delivery location.

In addition to the caDSR components corresponding to elements of the ISO/IEC 11179 metamodel, the caDSR model defines a collection of domain-specific elements for capturing data associated with a clinical trial protocol. These components are known as Forms and are not limited to usage in clinical trials, but can be used to support any data collection effort based on the notion of forms. All of the components described up to this point provide the infrastructure for managing shared data. The clinical trials components exercise the representational power of the metamodel, and are used to specify how clinical trials data should be captured and exchanged.

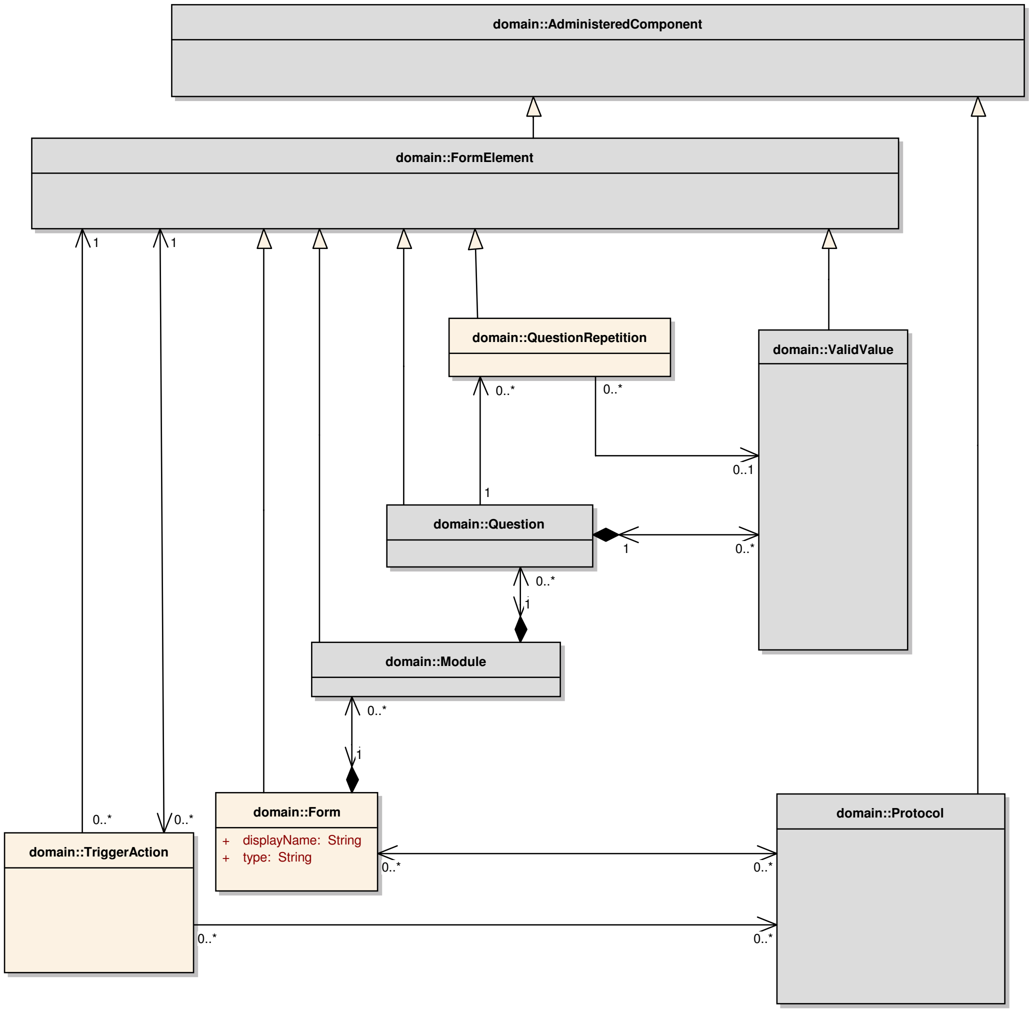


Figure 5.6 Components in the caDSR metamodel for clinical trials data

All of the components in *Figure 5.6* are highlighted in light gray, as they are *AdministeredComponents* designed for use in NCI-sponsored clinical trials. Note that because these elements are not part of the ISO-11179 specification, they are not technically speaking, ISO administered components. This caDSR design decision was made to ensure that these shared data elements could be stewarded and controlled adequately.

NCI-sponsored programs can populate the registry with instances of these components as needed to specify the metadata descriptors needed for that program. Programs currently participating in this effort include:

- The Cancer Therapy Evaluation Project (CTEP)
- Specialized Programs of Research Excellence (SPOREs)
- The Early Detection Research Network (EDRN)
- The Division of Cancer Prevention (DCP)
- The Cancer Imaging Program (CIP)
- The Division of Cancer Epidemiology and Genetics (DECG)

- The Cancer Bioinformatics Infrastructure Objects Project (caBIO)

Component Name	***Component Description***
Form	
FormElement	A generic class holding the common attributes and operations for the more specific classes that make up a Form: Modules, Questions, QuestionRepetitions, and ValidValues.
Module	A logical grouping of data elements on a Form.
Question	The text that accompanies a data element being collected on a Form; used to clarify the information being requested.
QuestionRepetition	A second or greater occurrence of a Question already on a Form.
ValidValue	One or more acceptable responses to a Question.
TriggerAction	A conditional branching between FormElements triggered by a certain response to a Question. For example: If the response to a Gender question is "Female", go to Question nn; otherwise known as a "skip pattern."
Protocol	A document defining the scope, objectives, and approach for conducting a clinical trial.

Table 5.3 caDSR component names and descriptions

caDSR API

The previous section described three broad categories of component in the caDSR metamodel and presented each of these independently, thus implying that there are no dependencies among these groupings. *Figure 5.7* brings these components together and exposes the associations actually occurring between components in different categories.

As in the previous diagram, all components highlighted in gray are descendants of the *AdministeredComponent* class. We emphasize, however, that some of these elements- i.e., those supporting clinical trials specific data are not defined in the ISO/IEC 11179 standard, are nevertheless implemented as subclasses of the *AdministeredComponent* class in the caDSR implementation for pragmatic reasons.

Because so many components are *AdministeredComponent* subclasses, we use color coding instead of the standard UML generalization notation (a line ending in an open triangle) to indicate this. Other superclass-subclass relations, such as the *ValueMeaning* class derived from *PermissibleValue*, do however use the standard UML notation.

The three categories of components are also outlined in the figure: administrative and organizational components are in the upper left, forms components are in the upper right, and information components are centered underneath these two.

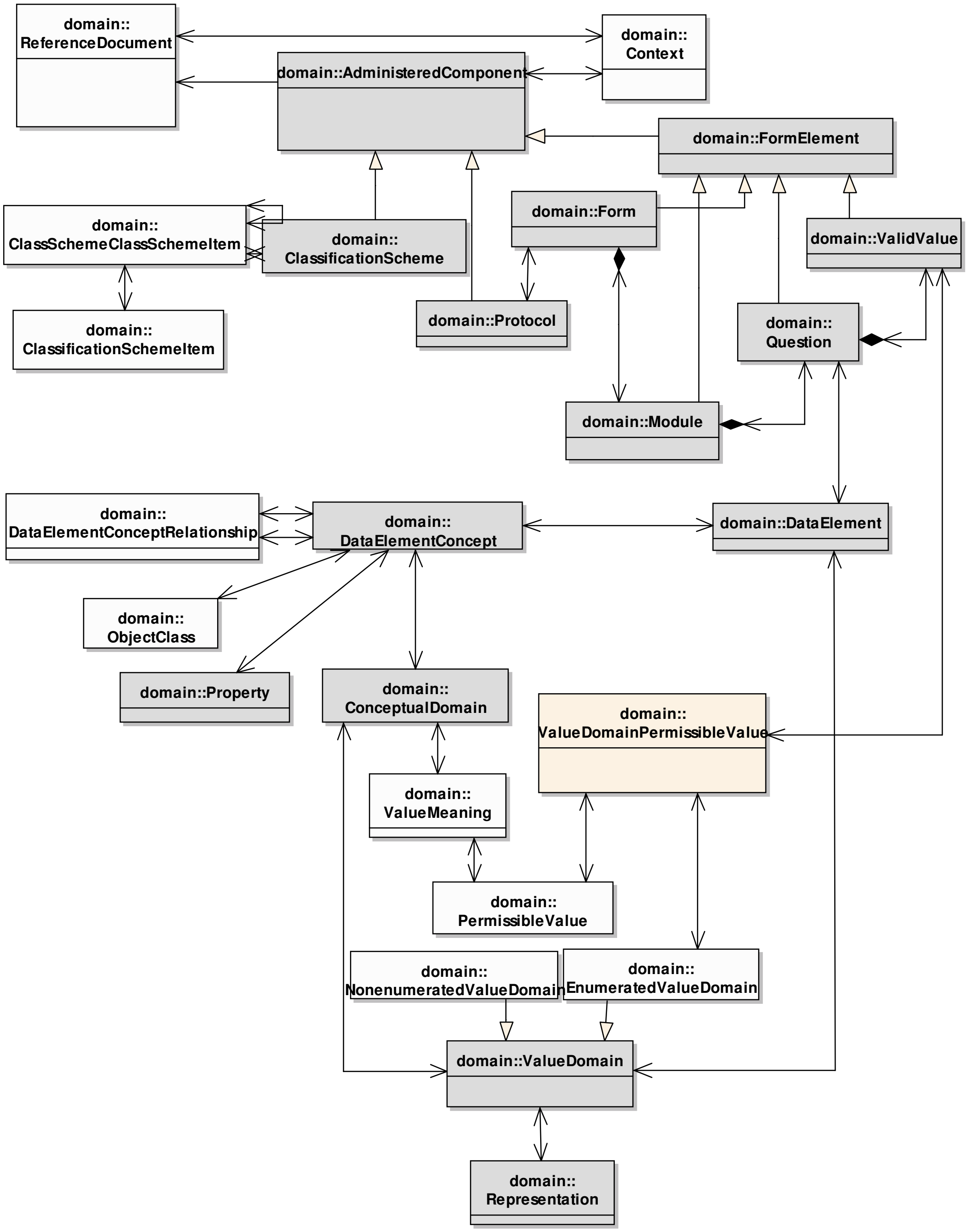

domain::ReferenceDocument
domain::Context
domain::AdministeredComponent
domain::FormElement
domain::Form
domain::ValidValue
domain::ClassSchemeClassSchemeItem
domain::ClassificationScheme
domain::Protocol
domain::ClassificationSchemeItem
domain::Question
domain::Module
domain::DataElementConceptRelationship
domain::DataElementConcept
domain::DataElement
domain::ObjectClass
domain::Property
domain::ConceptualDomain
domain::ValueDomainPermissibleValue
domain::ValueMeaning
domain::PermissibleValue
domain::NonenumeratedValueDomain
domain::EnumeratedValueDomain
domain::ValueDomain
domain::Representation

Figure 5.7 caDSR domain objects in the caCORE Java API

Figure 5.8 summarizes the caDSR API class hierarchy. At the most abstract level, a single distinguished object called the *DomainObject* class is the ancestor to all other classes. At the next level, the *AdministeredComponent* class is defined, along with all other classes that do not represent elements requiring administration. Among the

AdministeredComponent subclasses, only the *ValueDomain* class has further specialization, i.e., the *EnumeratedValueDomain* and *NonEnumeratedValueDomain* classes.

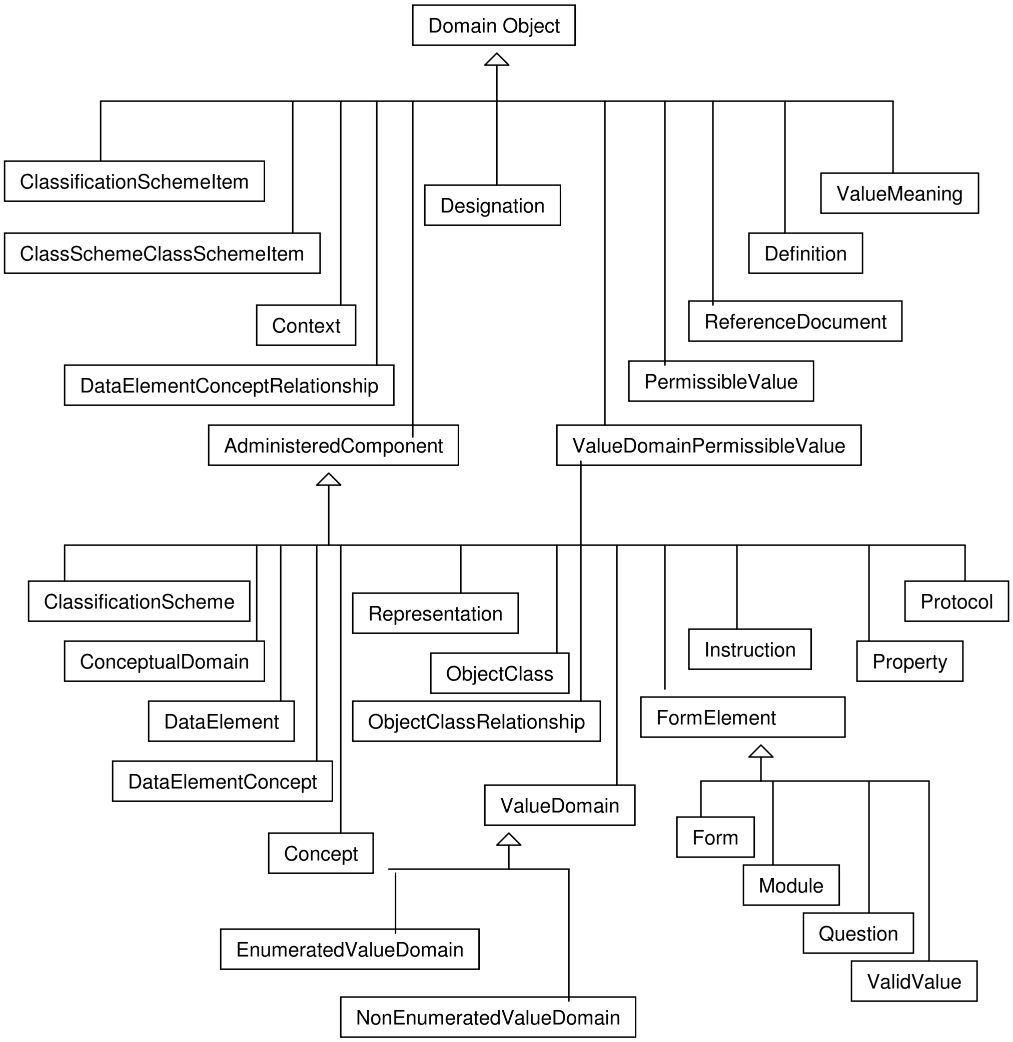


Figure 5.8 caDSR API class hierarchy

caDSR Domain Object Catalog

The previous discussion was intended to provide a descriptive overview of the domain objects included in the caCORE API to the caDSR. The caDSR UML model is published as an EA (Enterprise Architect) diagram at http://ncicb.nci.nih.gov/NCICB/content/ncicblfs/EA/caCORE3-2Model/index.htm. *Table 5.4* lists each class and a description. Detailed descriptions about each class and its methods are present in the caCORE 3.2 JavaDocs.

caDSR Domain Object	***Description***
Address	A physical location at which Persons or Organizations can be contacted.

Table 5.4 caDSR domain objects and descriptions

caDSR Domain Object	Description
Administered Component	A class for which attributes (or characteristics) are collected; Data Elements are one type of administered component. Other administered components have relationships to data elements as well as each other.
AdministeredComponent-ClassSchemeItem	A class that serves to allow many to many relationships between Administered Component (AC) and ClassScheme-ClassSchemeItem (CS/CSI), providing uniqueness to the CS/CSI pairing to an AC.
AdministeredComponentContact	A relationship between an Administered Component and contact information (e.g. Address).
CaseReportForm	A questionnaire that documents all the patient data stipulated in the protocol and used by clinicians to record information about patient's visits while on the clinical trial.
ClassificationScheme	A class that serves to describe an arrangement or division of objects into groups based on characteristics that the objects have in common, e.g., origin, composition, structure, application, function, etc. Adds information not easily included in definitions, helps organize the registry and facilitates access to the registry. ISO DEF: the descriptive information for an arrangement or division of objects into groups based on characteristics, which the objects have in common.
ClassificationSchemeItem	A component of content in a Classification Scheme. This may be a node in a taxonomy or ontology or a term in a thesaurus, etc.
ClassificationSchemeItemRelationship	The affiliation between two occurrences of Classification Scheme Items.
ClassificationSchemeRelationship	The affiliation between two occurrences of Classification Schemes.
ClassSchemeClassSchemeItem	Information pertaining to the association between Classification Schemes and Classification Scheme Items. This information is used to get all Classification Scheme Items that belong to a particular Classification Scheme as well as the information about it
ComponentConcept	The concept component(s) used for a concept derivation
ComponentLevel	Level of the component of the derivation rule
Concept	A class that serves to describe an administered component
ConceptDerivationRule	The derivation rule between one or more concepts.
ConceptualDomain	The set of all possible Valid Value meanings of a Data Element Concept expressed without representation.
Context	A designation or description of the application environment or discipline in which a name is applied or from which it originates.

Table 5.4 caDSR domain objects and descriptions (Continued)

caDSR Domain Object	Description
DataElement	A class that serves to describe a unit of data for which the definition, identification, representation and permissible values are derived from its association with one DataElementConcept and one ValueDomain.
DataElementConcept	A class that serves to describe a concept that can be represented in the form of a data element, described independently of any particular representation.
DataElementConceptRelationship	A description of the affiliation between two occurrences of Data Element Concepts.
DataElementDerivation	The data element component(s) used for a derived data element.
DataElementRelationship	The affiliation between two occurrences of Data Elements.
Definition	A defintion for an Administered Component in a specific Context.
DefinitionClassSchemeItem	A class that serves to allow many to many relationships between Definitions and ClassSchemeClassSchemeItem, providing uniqueness to the CS/CSI pairing to a definition. *Please be advised this class will be removed during a future release*
DerivationType	The type of Derived Data Element that is being created. For example a Data Element that is derived/created by subtracting two dates represented by other data elements would be a Calculated Representation Type. Types include: Calculated, Complex Recode, Compound, Concatenation, Object Class, and Simple Recode.
DerivedDataElement	The Data Element that is derived from one or more data elements. ISO DEF: the relationship among a Data Element which is derived, the rule controlling its derivation, and the Data Element(s) from which it is derived.
Designation	A name by which an Administered Component is known in a specific Context. Also a placeholder to track the usage of Administered Components by different Contexts.
DesignationClassSchemeItem	A class that serves to allow many to many relationships between Designation and ClassSchemeClassSchemeItem, providing uniqueness to the CS/CSI pairing to an Designation.
EnumeratedValueDomain	A ValueDomain with an associated set of discrete PermissibleValues; the alternative to a NonenumeratedValueDomain.
Form	A questionnaire that documents all the patient data stipulated in the protocol and used by clinicians to record information about patient's visits while on the clinical trial.

Table 5.4 caDSR domain objects and descriptions (Continued)

caDSR Domain Object	Description
FormElement	An element on a Case Report Form. Examples: The form itself, groups of questions (modules), questions, valid values.
Function	Function to be applied to the relationship
Instruction	Instruction for a Form, Module, Question or Valid Value on a Form
Module	A collection of data elements, or Common Data Elements, logically grouped on a case report form.
NonenumeratedValueDomain	A value domain not expressed as a list of all permissible values.
ObjectClass	A class that serves to describe a set of ideas, abstractions, or things in the real world that can be identified with explicit boundaries and meaning and whose properties and behavior follow the same rules.
ObjectClassRelationship	A class that provides a description of the affiliation between two occurrences of Object Classes
Organization	Information about an Organizational unit like a laboratory, institute or consortium
PermissibleValue	An enumerated list of the exact names, codes and text that can be stored in a data field in an information management system. ISO DEF: An expression of a value meaning in a specific value domain.
Person	Information about a contact person
Property	A class that serves to describe a characteristic common to all members of an Object Class. It may be any feature that humans naturally use to distinguish one individual object from another. It is conceptual and thus has no particular associated means of representation by which property
Protocol	A class that serves to provide identification of a Clinical Trial Protocol document and its collection of Case Report Forms (CRFs). Note: Protocols will be uniquely identified within each of the 3 areas of caCORE - caBIO, SPORES and caDSR- using the following three attributes: Protocol ID, Protocol Version and Context Name. This class will serve as a 'hook' across the three caCORE domains allowing a user to navigate across databases.
ProtocolFormsSet	Identification of a Clinical Trial Protocol document and its collection of Case Report Forms (CRFs). Note: Protocols will be uniquely identified within each of the 3 areas of caCORE - caBIO, SPORES and caDSR- using the following three attributes: Protocol ID, Protocol Version and Context Name. This class will serve as a 'hook' across the three caCORE domains allowing a user to navigate across databases.

Table 5.4 caDSR domain objects and descriptions (Continued)

caDSR Domain Object	Description
ProtocolFormsTemplate	The collection of components (modules, questions and valid values) comprising a template Case Report Form. A template form is not associated with any particular clinical trial.
Question	The actual text of the data element as specified on a Case Report Form of a Protocol
QuestionRepitition	Information about the default valid values everytime the question repeats on a form
ReferenceDocument	A place to document additional information about Administered Components that is not readily stored elsewhere.
Representation	A class that serves to describe the mechanism by which the functional and/or presentational category of an item maybe conveyed to a user. Component of a Data Element Name that describes how data are represented (i.e. the combination of a Value Domain, data type, and if necessary a unit of measure or a character set.) The Representation occupies the last position in the Data Element name (i.e. rightmost). Examples: Code - A system of valid symbols that substitute for specified values e.g. alpha, numeric, symbols and/or combinations. Count: Non-monetary numeric value arrived at by counting. Currency: Monetary representation. Date: Calendar representation e.g. YYYY-MM-DD Graphic: Diagrams, graphs, mathematical curves, or the like Image: usually a vector image. Icon: A sign or representation that stands for its object by virtue of a resemblance or analogy to it. Picture: A visual representation of a person, object, or scene ? usually a raster image. Quantity: A continuous number such as the linear dimensions, capacity/amount (non-monetary) of an object. Text: A text field that is usually unformatted. Time: Time of day or duration e.g. HH:MM:SS.SSSS.
TriggerAction	A conditional branching between to FormElements triggered by a certain response to a Question.
ValidValue	The allowable values for a Question on a Form.
ValueDomain	A class that serves to describe the attributes for a set of permissible values for a data element.
ValueDomainPermissibleValue	This captures the many-to-many relationship between value domain and permissible values and allows to associate a value domain to a permissible value.
ValueDomainRelationship	The affiliation between two occurrences of Value Domains.
ValueMeaning	The significance associated with an allowable/permissible value.

Table 5.4 caDSR domain objects and descriptions (Continued)

The *Concept* object encapsulates a concept in a vocabulary served by EVS. It is modeled as a new administered component type. The following table contains a brief description of some of its attributes.

Concept Object Attribute	***Description***
Concept Attribute	Data
Short Name	Contains the immutable concept code of an EVS concept
Long Name	Contains the preferred name of an EVS concept
Preferred Definition	Contains the definition of an EVS concept
Preferred Name	Contains the immutable concept code of an EVS concept
Long Name	Contains the preferred name of an EVS concept
Preferred Definition	Contains the definition of an EVS concept

Table 5.5 Concept object attributes

The *ConceptDerivationRule* object encapsulates a rule or formula that is applied to a collection of concepts resulting in a compound concept. It is modeled as an aggregate that is composed of a ordered collection of concepts. Each concept in the aggregation is referred to as a component concept and is encapsulated by *ComponentConcept* object. Each *ComponentConcept* object is associated to exactly one *Concept* object. Each *ConceptDerivationRule* object is also associated with one *ConceptDerivationType* object which encapsulates a type of concept derivation rule.

ConceptDerivationRule is a very key object as it is associated to several existing administered component types. It enables creation of various administered component types based on concepts that are served by EVS vocabularies.

As illustrated in *Figure 5.9*, the following administered components types are associated with the ConceptDerivationRule object:

- ObjectClass
- Property
- Representation
- ValueDomain
- ConceptualDomain

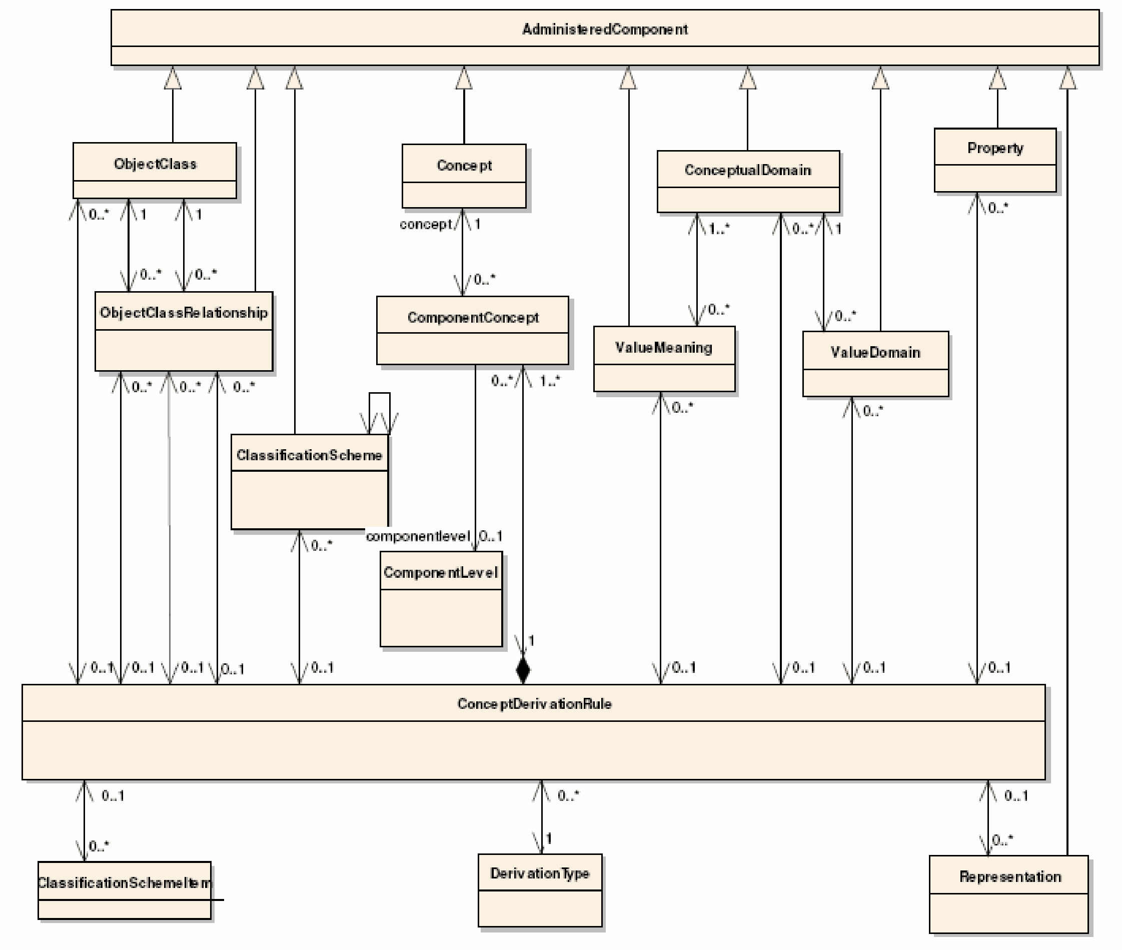


Figure 5.9 Extensions to the caDSR model

The *ObjectClassRelationship* object encapsulates relationship/association information between two object classes.

Each object listed above is associated to zero or one *ConceptDerivationRule* object. Each *ConceptDerivationRule* object could be used by one or more administered component type objects.

The *ObjectClassRelationship* object encapsulates relationship/association between two object classes and is only used to store the details of association between two UML classes.

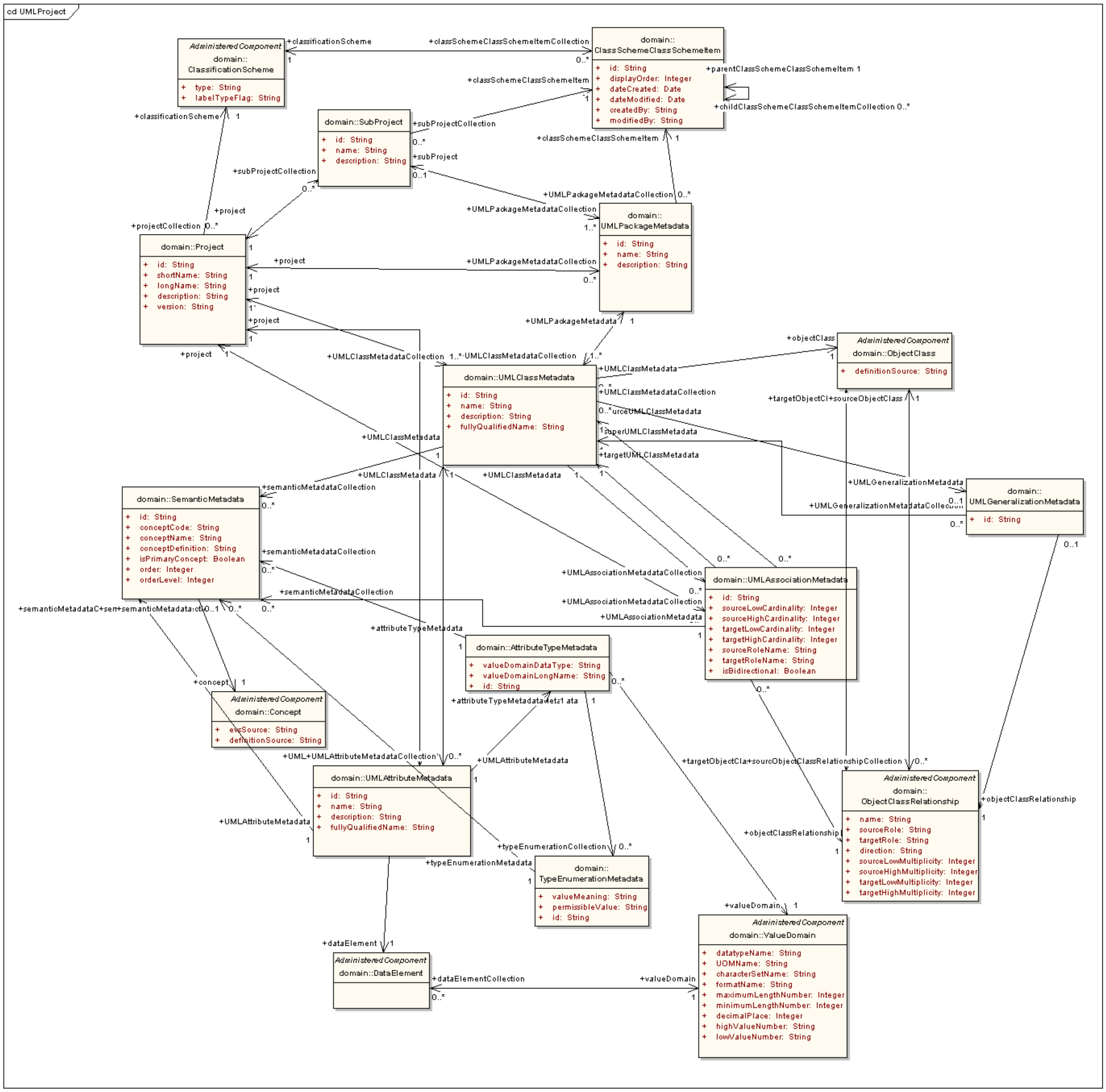


Figure 5.10 UML Project Class Diagram

caDSR UMLProject Domain Objects	***Description***
AttributeTypeMetadata	Class attribute type. Contains value domain name, data type, and a reference to the corresponding value domain in caDSR

Table 5.6 caDSR UML project domain objects

caDSR UMLProject Domain Objects	Description
Project	Used by the UML loader to group UML models, e.g. caCORE. Contains a reference to the corresponding classification scheme.
SemanticMetadata	Concept related information. Also is the superclass of all metadata classes.
SubProject	Optional groupings of UML models within one project, e.g. caBIO. A subproject contains reference to the project to which it belongs and a collection of UMLPackageMetaData.
TypeEnumerationMetadata	A subclass of SemanticMetadata that corresponds to an enumerated value domain.
UMLAssociationMetadata	A description of the affiliation between two UML classes.
UMLAttributeMetadata	UML Attribute. Contains a reference to the corresponding data element.
ClassMetadata	UML Class.
UMLGeneralizationMetadata	A description of the inheritance relationship between two classes.
UMLPackageMetadata	UML package. Contains a reference the corresponding classification scheme item, the project, and subject to which it belongs, and a collection of ClassMetadata objects that correspond to the UML classes in this package.

Table 5.6 caDSR UML project domain objects (Continued)

Downloading the caDSR

The following caDSR distributions can be downloaded from the NCICB download site.

caDSR Tool	Description
CDE Curation Tool Distribution	Contains the cdecurate.war file and the CDE Curation Tool Installation Guide.
CDE Browser Distribution	Contains cdebrowser.ear file and installation instructions for the application. CDE Browser makes Java Database Connectivity (JDBC) connections to caDSR repository database so it is a prerequisite to have access to caDSR repository for installing CDE Browser.
caDSR Sentinel Tool	Contains source, third party libraries, configuration files, documentation and installation instructions for the application. Access to a caDSR repository is required.
caDSR Repository/ Administration Tool Distribution	Contains the caDSR Installation Guide and scripts for both the caDSR repository and Administration Tool.

Table 5.7 caDSR tools available for download

caDSR Tool	***Description***
caDSR Repository/ Administration Tool Source Code Distribution	Contains the complete PL/SQL scripts for both the repository and the Administration Tool.
UML Model Browser Distribution	Contains `umlmodelbrowser.ear` file and installation instructions for the application.

Table 5.7 caDSR tools available for download

Follow the instructions provided with the distributions to install the software. caDSR content is not downloaded via this process. caDSR data element content may be downloaded via the online CDE Browser.

caDSR API Examples

Using the caDSR Java API

Example One: Querying the latest version of a DataElement

This example queries the latest version of a DataElement. It then queries associated objects such as DataElementConcept, ValueDomain and prints, out the PermissibleValues for the ValueDomain.

```
// Import the caDSR api objects
import gov.nih.nci.cadsr.domain.*;
import gov.nih.nci.cadsr.umlproject.domain.*;
// Import the ApplicationService objects
import gov.nih.nci.system.applicationservice.ApplicationService;
import gov.nih.nci.system.applicationservice.ApplicationServiceProvider;

import java.util.Collection;
import java.util.Iterator;
import java.util.List;

/**
 *  A sample use of the caDSR api.
 *  @author caDSR team.
 */
public class TestCaDsrApi {

    ApplicationService appService = null;

    public static void main(String[] args) {
        ApplicationService appService =
ApplicationServiceProvider.getRemoteInstance("http://cabio.nci.nih.gov/
cacore32/http/remoteService");
        System.out.println("Searching for DataElements");
        DataElement dataElement = new DataElement();
        // * is used as a wild card.
        // Set the search criteria
        dataElement.setLongName("Patient Race Category*");
        dataElement.setLatestVersionIndicator("Yes");
        try {
             // Set the case sensitivity of search to false
             appService.setSearchCaseSensitivity(false);
             // Search for objects of type DataElement
             List results = appService.search(DataElement.class,
dataElement);
for(Iterator iterate=results.iterator();iterate.hasNext();){
                 DataElement dataElementQ = (DataElement)iterate.next();
                 System.out.println("Data Element
"+dataElementQ.getLongName());
                // query the DataElementConcept
                 DataElementConcept dataElementConcept =
dataElementQ.getDataElementConcept();
System.out.println("Data Element Concept
"+dataElementConcept.getLongName());
                 // query the ValueDomain
                 ValueDomain valueDomain = dataElementQ.getValueDomain();
                 System.out.println("Value Domain
"+valueDomain.getLongName());
.
                                              if (valueDomain instanceof
EnumeratedValueDomain){
                    // Get the PermissibleValues for the ValueDomain
                     EnumeratedValueDomain evd =
(EnumeratedValueDomain)valueDomain;
                     Collection vdpvs=
evd.getValueDomainPermissibleValueCollection();
                     for (Iterator
iter2=vdpvs.iterator();iter2.hasNext();){
                         ValueDomainPermissibleValue vdpv =
(ValueDomainPermissibleValue)iter2.next();
                         PermissibleValue pv =
vdpv.getPermissibleValue();
                         System.out.println(" Permissible Value :
"+pv.getValue());
                     }
                 }
             }
        }
        catch (Exception exception){
            exception.printStackTrace();
            System.out.println("Error in the TestCaDsrApi");
        }
    }
}
```

Using the caDSR Web Services API

Example Two: Using datatypes generated from Apache Axis

This example uses the stubs, skeletons and data types generated with Apache Axis 1.2.1 WSDL-to-java tool. This also queries the latest version of a DataElement. It then queries associated objects such as DataElementConcept, ValueDomain and prints out the PermissibleValues for the ValueDomain.

```
// Import the generated stubs.
import gov.nih.nci.cabio.cacore32.ws.caCOREService.WSQuery;
import gov.nih.nci.cabio.cacore32.ws.caCOREService.WSQueryService;
import gov.nih.nci.cabio.cacore32.ws.caCOREService.WSQueryServiceLocator;
// Import the api
import gov.nih.nci.cadsr.domain.ws.*;

/**
 *  A sample use of the caDSR webservice API. This uses the stubs,
skeletons, and data types
 *  generated using apache axis 1.2.1 WSDL-to-java tool.
 *
 *  @author caDSR team
 */
public class TestCaDsrWSApi {

    public static void main(String[] args) {
       try {
        // Get a caCORE Service  instance.
         WSQuery wsQuery = new
WSQueryServiceLocator().getcaCOREService();
         System.out.println("Query a DataElement");
         DataElement dataElement = new DataElement();
         // Set the search criteria. * is used as a wild card.
         dataElement.setLongName("Patient Race Category*");
         dataElement.setLatestVersionIndicator("Yes");
         //Search for DataElements
         System.out.println("Searching for data elements");
         Object[] results =
wsQuery.queryObject("gov.nih.nci.cadsr.domain.ws.DataElement",
dataElement);
for (int i=0;i<results.length;i++){
             DataElement dataElementQ = (DataElement)results[i];
             System.out.println("Queried DataElement
"+dataElementQ.getLongName());
             //Query DataElementConcept
             DataElement de= new DataElement();
             de.setId(dataElementQ.getId());
             DataElementConcept dec =
(DataElementConcept)wsQuery.queryObject("gov.nih.nci.cadsr.domain.ws.Data
ElementConcept",de)[0];
System.out.println("Queried DataElementConcept "+dec.getLongName());
             //Query ValueDomain
             ValueDomain vd =
(ValueDomain)wsQuery.queryObject("gov.nih.nci.cadsr.domain.ws.ValueDomain
", de)[0];
             System.out.println("Queried ValueDomain "+vd.getLongName());
             if (vd instanceof EnumeratedValueDomain){
                //Query Permissible Values
                EnumeratedValueDomain evd = new EnumeratedValueDomain();
                evd.setId(vd.getId());
                Object[] valueDomainPermissibleValues=
wsQuery.queryObject("gov.nih.nci.cadsr.domain.ws.ValueDomainPermissibleVa
lue", evd);
                for (int j=0;j<valueDomainPermissibleValues.length;j++){
                    ValueDomainPermissibleValue vdpv =
(ValueDomainPermissibleValue)valueDomainPermissibleValues[j];
                    ValueDomainPermissibleValue vdpv2 = new
ValueDomainPermissibleValue();
                    vdpv2.setId(vdpv.getId());
                    PermissibleValue pv =
(PermissibleValue)wsQuery.queryObject("gov.nih.nci.cadsr.domain.ws.Permis
sibleValue", vdpv2)[0];
                    System.out.println("Queried permissible value
"+pv.getValue());
                }
             }
         }
       }
       catch(Exception exception){
           exception.printStackTrace();
           System.out.println("Error testing web service api.");
       }
    }
}
```

UML Project API Examples

Example one: Using the caCORE client Java API

This example queries the UML model related objects through the caCore API. It first queries for all UML projects sorted by name. It prints out the name, version and context of the project. The second part of the example retrieves all classes named "gene", display class related information. The search criteria are not case sensitive. The last part of the example shows how to retrieve all attributes related information of a class.

```
import gov.nih.nci.cadsr.umlproject.domain.Project;
import gov.nih.nci.cadsr.umlproject.domain.UMLAttributeMetadata;
import gov.nih.nci.cadsr.umlproject.domain.UMLClassMetadata;
import gov.nih.nci.system.applicationservice.ApplicationService;

import gov.nih.nci.system.applicationservice.ApplicationServiceProvider;

import java.util.Iterator;
import java.util.List;

import org.hibernate.criterion.DetachedCriteria;
import org.hibernate.criterion.Order;


/**
 * @author Jane Jiang <a href="mailto: jane.jiang@oracle.com"></a>
 * @version 1.0
 */


/**
       * TestClient.java demonstartes various ways to execute searches with and without
      * using Application Service Layer (convenience layer that abstracts building
criteria
      * Uncomment different scenarios below to demonstrate the various types of searches
*/
public class

TestUml {


   public static void main(String[] args) {
      Project project = null;
      System.out.println("*** TestUml...");
      try {
         ApplicationService appService =
ApplicationService.getRemoteInstance("http://cabio.nci.nih.gov/cacore32/http/remoteServic
e");

         System.out.println("Using basic search.  Retrieving all projects");
         DetachedCriteria projectCriteria =
            DetachedCriteria.forClass(Project.class);
         projectCriteria.addOrder(Order.asc("shortName"));

         try {
            System.out
            .println("Scenario 1: Using basic search.  Retrieving all projects, display
version and context information...");

            List<Project> resultList =
               appService.query(projectCriteria, Project.class.getName());
            ;
            System.out.println(resultList.size() + " projects retrieved..");
            for (Iterator resultsIterator = resultList.iterator();
                 resultsIterator.hasNext(); ) {
               project = (Project)resultsIterator.next();
               System.out.println("Project name: " + project.getShortName());
               System.out.println("        version: " + project.getVersion());
               System.out
               .println("        context: " + project.getClassificationScheme()
                                  .getContext().getName());
            }
```

```
  System.out.println();
            System.out
            .println("Scenario 2: Retrieving class named Gene, display class 
information");
            UMLClassMetadata umlClass = new UMLClassMetadata();
            umlClass.setName("gene");
            resultList = appService.search(UMLClassMetadata.class, umlClass);
            System.out.println(resultList.size() + " classes retrieved..");
            for (Iterator resultsIterator = resultList.iterator();
                 resultsIterator.hasNext(); ) {
               umlClass = (UMLClassMetadata)resultsIterator.next();
               System.out
               .println("  class full name: " + umlClass.getFullyQualifiedName());
               System.out
               .println("  class description: " + umlClass.getDescription());
               System.out
               .println("  project version: " + umlClass.getProject().getVersion());
               System.out
               .println("  object class public id: " + umlClass.getObjectClass()
                                  .getPublicID());
            }

            System.out.println();
            System.out
            .println("Scenario 3: Retrieving attributes for a class, display attribute 
information");
            if (umlClass != null) {
               for (Iterator resultsIterator =
                       umlClass.getUMLAttributeMetadataCollection().iterator();
                    resultsIterator.hasNext(); ) {
                  UMLAttributeMetadata umlAttribute =
                     (UMLAttributeMetadata)resultsIterator.next();
                  printAttributeInfo(umlAttribute);

               }
            }

            System.out.println();
            System.out
            .println("Scenario 4: Retrieving attributes named *id, display attribute 
information");
            UMLAttributeMetadata umlAttr = new UMLAttributeMetadata();
            umlAttr.setName("*:id");
            resultList = appService.search(UMLAttributeMetadata.class, umlAttr);
            System.out.println(resultList.size() + " attributes retrieved..");

         } catch (Exception e) {
            e.printStackTrace();
         }
      } catch (RuntimeException e2) {
         e2.printStackTrace();
      }
   }
   private static void printAttributeInfo(UMLAttributeMetadata umlAttribute) {
      System.out.println("  Attribute name: " + umlAttribute.getName());
      System.out
      .println("  Attribute type: " + umlAttribute.getAttributeTypeMetadata()
                         .getValueDomainDataType());
      System.out
      .println("  Data Element public id: " + umlAttribute.getDataElement()
                         .getPublicID());
   }
}
```

CHAPTER 6

CANCER BIOINFORMATICS INFRASTRUCTURE OBJECTS

This chapter describes the Cancer Bioinformatics Infrastructure Objects (caBIO) model and its application programming interfaces.

Topics in this chapter include:

Introduction

The Cancer Bioinformatics Infrastructure Objects (caBIO) model and architecture was the first of several model-driven information systems that make up caCORE and continues to be an on-going effort to model the genomic domain. The caBIO objects simulate the behavior of actual genomic components such as genes, chromosomes, sequences, libraries, clones, ontologies, etc. They provide access to a variety of genomic data sources including GenBank, Unigene, LocusLink, Homologene, Ensemble, GoldenPath, and NCICB's CGAP (Cancer Genome Anatomy Project) data repositories. The full list of data sources is listed starting on page 112.

caBIO API

Most of the domain objects defined in the caBIO API are objects that specialize in bioinformatics applications. The caBIO domain objects are implemented as Java beans in the gov.nih.nci.cabio.domain package and include those classes that correspond to biological entities and bioinformatic concepts. The caBIO UML model is published as an EA (Enterprise Architect) diagram at http://ncicb.nci.nih.gov/NCICB/content/ncicblfs/EA/caCORE3-1Model/index.htm. Table *6.1* lists each class and a description. Detailed descriptions about each class and its methods are present in the caCORE 3.2 JavaDocs.

caBIO Domain Object	***Description***
Agent	A therapeutic agent (drug, intervention therapy) used in a clinical trial protocol.
Anomaly	An irregularity in either the expression of a gene or its structure (i.e., a mutation).
Chromosome	An object representing a specific chromosome for a specific taxon; provides access to all known genes contained in the chromosome and to the taxon.
ClinicalTrialProtocol	The protocol associated with a clinical trial; organizes administrative information about the trial such as Organization ID, participants, phase, etc., and provides access to the administered Agents.
Clone	An object used to hold information pertaining to I.M.A.G.E. clones; provides access to sequence information, associated trace files, and the clone's library.
CloneRelativeLocation	Provides the end (5'/3') of a clone insert read.
Cytoband	Represents the positions of cytogenetic bands within a chromosome.
CytogenicLocation	Provides cytoband information for SNP, Gene and NucleicAcidSequence objects.
DiseaseOntology	Disease objects specify a disease name and ID; disease objects also provide access to: ontological relations to other diseases; clinical trial protocols treating the disease; and specific histologies associated with instances of the disease.
DiseaseOntology Relationship	Specifies the relationship among diseases.
Gene	Gene objects are the effective portal to most of the genomic information provided by the caBIO data services; organs, diseases, chromosomes, pathways, and sequence data are among the many objects accessible via a gene.
GeneAlias	An alternative name for a gene; provides descriptive information about the gene (as it is known by this alias), as well as access to the Gene object it refers to.
GeneOntology	An object providing entry to a Gene object's position in the Gene Ontology Consortium's controlled vocabularies, as recorded by LocusLink; provides access to gene objects corresponding to the ontological term, as well as to ancestor and descendant terms.
GeneOntology Relationship	Specifies the Gene Ontology relationship.

Table 6.1 caBIO domain objects and descriptions

caBIO Domain Object	Description
GeneRelativeLocation	Provides the location (intron, upstream, downstream etc.) of a SNP with respect to its associated genes.
GenericArray	Represents the physical chip along with its features and the features' annotations.
GenericReporter	Represents some biological material (clone, set of oligos, etc.) on an array, which will report on some biosequence or biosequences.
Histopathology	Represents anatomical changes in a diseased tissue sample associated with an expression experiment; captures the relationship between organ and disease.
Homologous Association	Links Homologous Genes
Library	An object representing a CGAP library; provides access to information about: the tissue sample and its method of preparation, the library protocol that was used, the clones contained in the library, and the sequence information derived from the library.
Location	Super class of PhysicalLocation and CytogenicLocation
NucleicAcidSequence	An object representation of a gene sequence; provides access to the clones from which it was derived, the ASCII representation of the residues it contains, and the sequence ID.
OrganOntology	A representation of an organ whose name occurs in a controlled vocabulary; provides access to any Histopathology objects for the organ, and, because it is "ontolog-able," provides access to its ancestral and descendant terms in the vocabulary.
OrganOntologyRelationship	An object that describes relationships among organs.
Pathway	An object representation of a molecular/cellular pathway compiled by BioCarta. Pathways are associated with specific Taxon objects, and contain multiple Gene objects, which may be Targets for treatment.
PhysicalLocation	Provides chromosomal start and end positions and is associated with SNP and NucleicAcidSequence objects.
PopulationFrequency	Represents the major and minor alleles of a SNP and their respective frequencies in different populations.
Protein	An object representation of a protein; provides access to the encoding gene via its GenBank ID, the taxon in which this instance of the protein occurs, and references to homologous proteins in other species.
ProteinAlias	An alternate name for a protein.
ProteinSequence	The sequence of a protein.
Protocol	An object representation of the protocol used in assembling a clone library.
ProtocolAssociation	An association class relating Clinical Trial Protocols to Diseases.

Table 6.1 caBIO domain objects and descriptions (Continued)

caBIO Domain Object	Description
SNP	An object representing a Single Nucleotide Polymorphism; provides access to the clones and the trace files from which it was identified, the two most common substitutions at that position, the offset of the SNP in the parent sequence, and a confidence score
Target	A gene thought to be at the root of a disease etiology, and which is targeted for therapeutic intervention in a clinical trial.
Taxon	An object representing the various names (scientific, common, abbreviated, etc.) for a species associated with a specific instance of a Gene, Chromosome, Pathway, Protein, or Tissue.
Tissue	A group of similar cells united to perform a specific function.
Vocabulary	Describes the vocabulary.

Table 6.1 caBIO domain objects and descriptions (Continued)

Data Sources in the caBIO Database

This section describes the internal and external data sources for caBIO and how the information these sources provide can be accessed via caBIO objects.

The caBIO application programming interfaces were developed primarily in response to the need for programmatic access to the information at several NCI web sites, including:

- Cancer Genome Anatomy Project (CGAP)
- CGAP Genetic Annotation Initiative (GAI)
- Mouse Models of Human Cancers Consortium (MMHCC)
- Cancer Molecular Analysis Project (CMAP)
- Affymetrix
- University of California, Santa Cruz (UCSC)
- Integrated Molecular Analysis of Genomes and their Expression (IMAGE) Consortium
- and others

While this information is, in theory, available from multiple public sites, the number of links to traverse and the extent of collation that would have to be performed is daunting. The CGAP, CMAP, and GAI web sites have distilled this information from both internal and public databases, and the caBIO data warehouses have optimized it for access with respect to the types of queries defined in the APIs.

While the caBIO data are extracted from many sources that include information from a wide variety of species, we emphasize that *only genomic data pertaining to human and mouse are available from caBIO*.

caBIO provides access to curated data from both internal (NCI) and external sites. Table *6.2* contains data sources from that NCI-related and Table *6.3* contains data sources from sites outside of NCI.

NCI Datasource	***Description***
CGAP	CGAP (Cancer Genome Anatomy Project) provides a collection of gene expression profiles of normal, pre-cancer, and cancer cells taken from various tissues. The CGAP interface allows users to browse these profiles by various search criteria, including histology type, tissue type, library protocol, and sample preparation methods. The goal at NCI is to exploit such expression profile information for the advancement of improved detection, diagnosis, and treatment for the cancer patient. Researchers have access to all CGAP data and biological resources for human and mouse, including ESTs, gene expression patterns, SNPs, cluster assemblies, and cytogenetic information. The CGAP web site provides a powerful set of interactive data-mining tools to explore these data, and the caBIO project was initially conceived as a programmatic interface to these tools and data. Accordingly, most of the data that are available from CGAP can also be accessed through the caBIO objects. Exceptions are those data sets having proprietary restrictions, such as the Mitleman Chromosome Aberration database. CGAP also provides access to lists of sequence-verified human and mouse cDNA IMAGE clones supplied by Invitrogen.
CMAP	CMAP (Cancer Molecular Analysis Project) is powered by caCORE. The goal of CMAP is to enable researchers to identify and evaluate molecular targets in cancer. The CMAP *Profile Query* tool finds genes with the highest or lowest expression levels (using SAGE and microarray data) for a given tissue and histology. Selecting a gene from the resulting table then leads to a *Gene Info* page. This page provides information about cytogenetic location, chromosome aberrations, protein similarities, curated and computed orthologs, and sequence-verified as well as full-length MGC clones, along with links to various other databases.

Table 6.2 NCI-related data sources in the caBIO database

NCI Datasource	Description
CTEP	CTEP (Cancer Therapy Evaluation Program) funds an extensive national program of basic and clinical research to evaluate new anti-cancer agents, with a particular emphasis on translational research to elucidate molecular targets and drug mechanisms. In response to this emergent need for translational research, there has been a groundswell of translational support tools defining controlled vocabularies and registered terminologies to enhance electronic data exchange in areas that have heretofore been relatively non-computational. The caCORE trials data are updated with new CTEP data on a quarterly basis, and many of the objects are designed to support translational research. For example, a caCORE *Target* object represents a molecule of special diagnostic or therapeutic interest for cancer research, and an *Anomaly* object is an observed deviation in the structure or expression of a *Target*. An *Agent* is a drug or other intervention that is effective in the presence of one or more specific *Targets*. The *ClinicalTrialProtocol* object organizes administrative information pertaining to that protocol.
GAI	GAI (CGAP Genetic Annotation Initiative) is an NCI research program to explore and apply technology for identification and characterization of genetic variation in genes important in cancer. The GAI utilizes data-mining to identify "candidate" variation sites from publicly available DNA sequences, as well as laboratory methods to search for variations in cancer-related genes. All GAI candidate, validated, and confirmed genetic variants are available directly from the GAI web site, and all validated SNPs have been submitted to the NCBI dbSNP database as well.
HomoloGene	HomoloGene is an NCBI resource for curated and calculated gene homologs. The caBIO data sources capture only the calculated homologs stored by HomoloGene. These calculated homologs are the result of nucleotide sequence comparisons performed between each pair of organisms represented in UniGene clusters.
IMAGE	The IMAGE Consortium provides extensive data on clones. The data for a *Clone* object includes image clones from the IMAGE Consortium. The data behind the *CloneRelativeLocation* object which provides the end (5'/3') of a clone insert read comes from UniGene.
LocusLink	LocusLink contains curated sequence and descriptive information associated with a gene. Each entry includes information about the gene's nomenclature, aliases, sequence accession numbers, phenotypes, UniGene cluster IDs, OMIM IDs, gene homologies, associated diseases, map locations, and a list of related terms in the Gene Ontology Consortium's ontology. Sequence accessions include a subset of GenBank accessions for a locus, as well as the NCBI Reference Sequence.

Table 6.2 NCI-related data sources in the caBIO database (Continued)

NCI Datasource	Description
dbSNP	In collaboration with the National Human Genome Research Institute, the NCBI has established the dbSNP database to serve as a central repository for both single base nucleotide substitutions and short deletion and insertion polymorphisms. Once discovered, these polymorphisms could be used by additional laboratories, using the sequence information around the polymorphism and the specific experimental conditions. (Note that dbSNP takes the looser 'variation' definition for SNPs, so there is no requirement or assumption about minimum allele frequency.)
UniGene	Unigene provides a nonredundant partitioning of the genetic sequences contained in GenBank into gene clusters. Each cluster has a unique UniGene ID and a list of the mRNA and EST sequences that are included in that cluster. Related information stored with the cluster includes tissue types in which the gene has been expressed, mapping information, and the associated LocusLink, OMIM, and HomoloGene IDs, thus providing access to related information in those NCBI databases as well. Because the information in UniGene is centered around genes, access to Unigene is provided via the caBIO *Gene* objects. Specifically, the method *getClusterId()* associated with a *Gene* object can be used to fetch the gene's UniGene ID. Similarly, the database IDs for the NCBI OMIM and LocusLink databases can be obtained using the *getDatabaseCrossReferenceCollection()* method. While there is no explicit caBIO object corresponding to a Unigene cluster, all of the information associated with the cluster is available directly via the caBIO *Gene* object's methods.

Table 6.2 NCI-related data sources in the caBIO database (Continued)

External Datasource	Description
Affymetrix	Affymetrix provides the majority of data for the caBIO *SNP* object. The data provides information on allele frequencies of the SNP in different populations, and is represented by the *PopulationFrequency* object. The *GeneRelativeLocation* object provides the location (intron, upstream, downstream etc.) of a SNP with respect to its associated genes. The validation status for a SNP comes from NCBI. The SNP Consortium (TSC) Ltd. a non-profit foundation provides the TSC id's for SNPs.
BioCarta	BioCarta and its Proteomic Pathway Project (P3) provide detailed graphical renderings of pathway information concerning adhesion, apoptosis, cell activation, cell signalling, cell cycle regulation, cytokines/chemokines, developmental biology, hematopoeisis, immunology, metabolism, and neuroscience. NCI's CMAP web site captures pathway information from BioCarta, and transforms the downloaded image data into Scalable Vector Graphics (SVG) representations that support interactive manipulation of the online images. The CMAP web site displays BioCarta pathways selected by the user and provides options for highlighting *anomalies*, which include under- or overexpressed genes as well as mutations. caCORE 3.2 provides a class for manipulating SVG diagrams, which is described on *Manipulating SVG Diagrams* on page 117.

Table 6.3 External data sources in the caBIO database

External Datasource	Description
UniProt PIR	Universal Protein Resource (UniProt) is a complete annotated protein sequence database and is a central repository of protein sequence and function created by joining the information contained in Swiss-Prot, TrEMBL, and PIR. The UniProt Knowledge base provides access to extensive curated protein information, including the amino acid sequence, protein name or description, taxonomic data and protein aliases.
Gene Ontology Consortium	The Gene Ontology Consortium provides a controlled vocabulary for the description of molecular functions, biological processes, and cellular components of gene products. The terms provided by the consortium define the recognized attributes of gene products and facilitate uniform queries across collaborating databases. In general, each gene is associated with one or more biological processes, and each of these processes may in turn be associated with many genes. In addition, the GO ontologies define many parent/child relationships among terms. For example, a branch of the ontology tree under `biological_process` contains the term `cell cycle control`, which in turn bifurcates into the "child" terms `cell cycle arrest`, `cell cycle checkpoint`, `control of mitosis`, etc. caBIO does not extract ontology terms directly from the Gene Ontology Consortium but, instead, extracts those terms stored with the LocusLink entry for that gene.
SNP Consortium	The SNP Consortium Ltd. is a non-profit foundation organized for the purpose of providing public genomic data. Its mission is to develop up to 300,000 SNPs distributed evenly throughout the human genome and to make the information related to these SNPs available to the public without intellectual property restrictions.
UCSC	UCSC (University of California, Santa Cruz Distributed Annotation System) provides the data for the Chromosomal start and end positions of mRNA sequences. The positions of cytogenetic bands within a chromosome, represented by the caBIO `Cytoband` object, are also obtained from the UCSC.

Table 6.3 External data sources in the caBIO database (Continued)

caGrid Identifiers

The functionality provided by caGrid's Identifier Services Framework and the integration in caBIO is to have "identifiers" for each caBIO domain object. The identifier is essentially a forever globally unique name for the caBIO data-object such that it can be unambiguously used to refer to the data from different application contexts.

This identifier, obtained from the caGrid Identifier Service Framework, is essentially a string and a forever globally unique name for the caBIO domain object. Furthermore, the identifier can be (globally) resolved to the associated caBIO domain object.

In order to abstract the identifier's object properties, the data service implementations and the resolution mechanisms, the identifier's value must be treated as a "meaningless" opaque string by the consumer applications. Any leaking of design and implementation choices for the identifier framework in the applications is undesirable from an architecture point of view as it makes the implementations brittle and susceptible to future changes. Of course resolution information will have to be embedded in identifier name, but this should only be meaningful for resolution service related components that are layered below the application.

caBIO Specific Utilities

Manipulating SVG Diagrams

caBIO 3.2 provides a utility class called *SVGManipulator* for manipulating pathway SVG diagrams. BioCarta and its Proteomic Pathway Project (P3) provide detailed graphical renderings of pathway information. NCI's CMAP web site captures pathway information from BioCarta, and transforms the downloaded image data into Scalable Vector Graphics (SVG) representations that support interactive manipulation of the online images. The *SVGManipulator* utility class provides the capability to do the following:

- Change the display colors for each gene contained in an SVG diagram.
- Modify the URL linking a gene in the SVG diagram to external gene information. The default gene URL links to the CMAP website.
- Disable all genes or nodes within an SVG diagram.
- Retrieve a gene’s color.
- Reset a gene or node to its original state.
- Retrieve/set SVG diagram attributes via getter/setter methods.

Figure 6.1 shows an example of how to use *SVGManipulator* class to modify content of an SVG diagram. This example uses the *gov.nih.nci.cabio.domain.Pathway* and *gov.nih.nci.cabio.domain.Gene* objects.

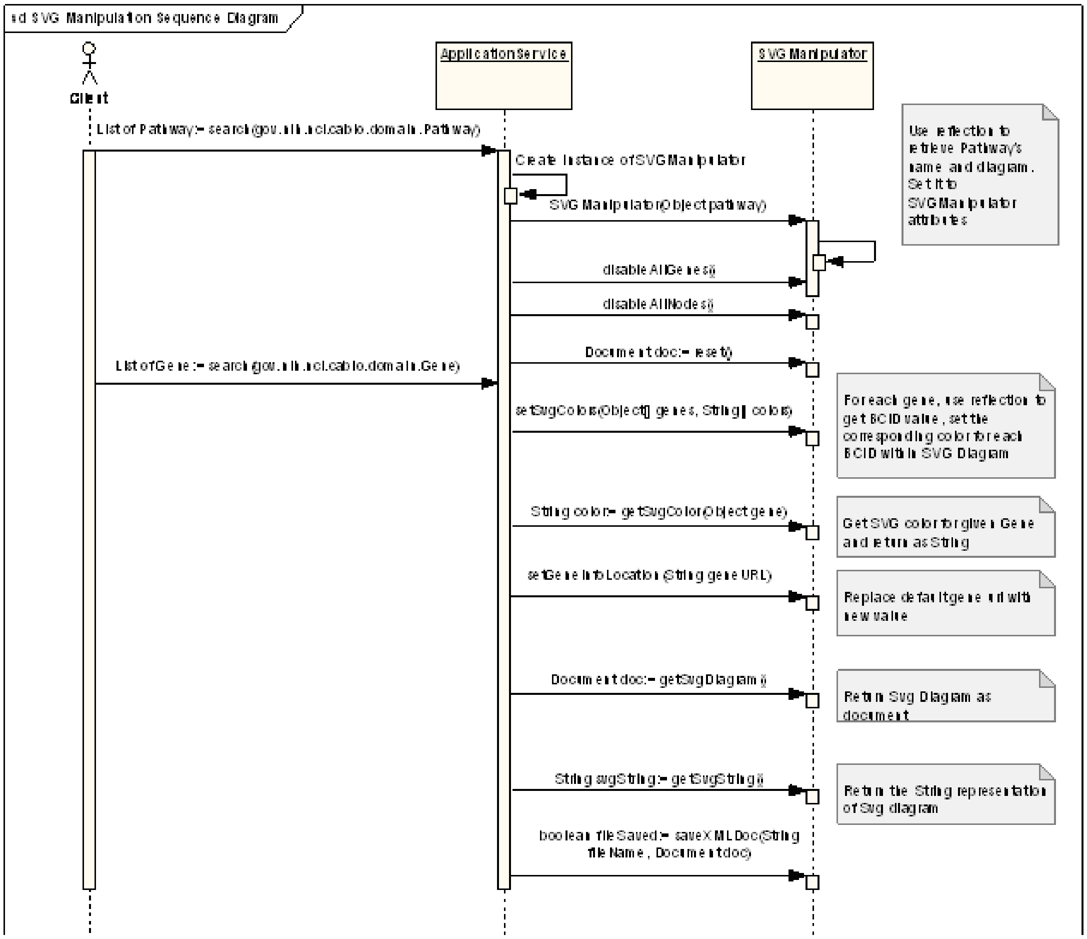

sd SVG Manipulation Sequence Diagram
Client
ApplicationService
SVG Manipulator
List of Pathway:= search(gov.nih.nci.cabio.domain.Pathway)
Create Instance of SVGManipulator
SVG Manipulator(Object pathway)
Use reflection to retrieve Pathway's name and diagram. Set it to SVGManipulator attributes
disableAllGenes()
disableAllNodes()
Document doc:= reset()
List of Gene := search(gov.nih.nci.cabio.domain.Gene)
setSvgColors(Object[] genes, String[] colors)
For each gene, use reflection to get BCID value, set the corresponding color for each BCID within SVG Diagram
String color= getSvgColor(Object gene)
Get SVG color for given Gene and return as String
setGeneInfoLocation(String gene URL)
Replace default gene url with new value
Document doc:= getSvgDiagram()
Return Svg Diagram as document
String svgString:= getSvgString()
Return the String representation of Svg diagram
boolean fileSaved:= saveXMLDoc(String fileName, Document doc)

Figure 6.1 Sequence diagram using caBIO Pathway and Gene objects

SVG Diagram Manipulation Utility Example

The following test client demonstrates how to use the *SVGManipulation* class to change the appearance of a pathway diagram associated with a given pathway.

```
import gov.nih.nci.system.applicationservice.*;
import java.util.*;

import gov.nih.nci.camod.domain.*;
import gov.nih.nci.camod.domain.impl.*;

import gov.nih.nci.cadsr.domain.*;
import gov.nih.nci.cadsr.domain.impl.*;

import gov.nih.nci.cabio.domain.*;
import gov.nih.nci.cabio.domain.impl.*;
import gov.nih.nci.common.util.*;

import org.hibernate.criterion.*;
```

```
public class TestClient {

    public static void main(String[] args) {

       System.out.println("*** TestClient...");
        try{

       ApplicationService appService =
       ApplicationServiceProvider.getApplicationService();

    /**** Test scenarios for SVG Pathway Diagram using SVGManipulator.java
*****/

    try {

        System.out.println("Using basic search.  Retrieving Pathway");
        Pathway pw = new PathwayImpl();
        pw.setId(new Long(251));

        try {
            List resultList = appService.search(Pathway.class, pw);
            System.out.println("result count: " + resultList.size());
            for (Iterator resultsIterator = resultList.iterator();
                    resultsIterator.hasNext();) {
                Pathway returnedPw = (Pathway) resultsIterator.next();
                String pathwayDiagram = returnedPw.getDiagram();

                SVGManipulator svgM = new SVGManipulator(returnedPw);
                Document orgSvgDoc = svgM.getSvgDiagram();

                Document org0 = svgM.reset();

                svgM.disableAllGenes();
                Document disableGenesDoc = svgM.getSvgDiagram();
                disableGenesDoc);

                Document org1 = svgM.reset();

                svgM.disableAllNodes();
                Document disableNodesDoc = svgM.getSvgDiagram();
                disableNodesDoc);

                Document org = svgM.reset();

                Gene[] genes= new Gene[2];
                String[] colors=new String[2];

                Gene p53=new GeneImpl();
                p53.setId(new Long(1031));
                List resultList1 = appService.search(Gene.class, p53);
                if(resultList1.size()> 0)
                    genes[0]=(Gene)resultList1.get(0);

                Gene p54=new GeneImpl();
                p54.setId(new Long(2));
                List resultList2 = appService.search(Gene.class, p54);
                genes[1] = (Gene)resultList2.get(0);
                colors[0]="255,255,255";
                colors[1]="0,255,255";

                svgM.setSvgColors(genes, colors);

                Document geneColors = svgM.getSvgDiagram();

                String genep53color = svgM.getSvgColor(genes[0]);
                System.out.println("geneP53 color: " + genep53color);

                Document org10 = svgM.reset();

                String genep53color1 = svgM.getSvgColor(genes[0]);
                System.out.println("geneP53 color1: " + genep53color1);
                String svgString = svgM.toString();
                System.out.println("toString:\n" + svgString);

                svgM.setGeneInfoLocation("http://www.google.com");

                Document geneLocation = svgM.getSvgDiagram();

                // Using Map
                Map geneColors = new HashMap();
                geneColors.put("rab7", "0,255,255");
                geneColors.put("rab1", "0,255, 255");

                svgM.setSvgColors(geneColors);

                Document d = svgM.getSvgDiagram();

            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    } catch (RuntimeException e2) {
        // TODO Auto-generated catch block
        e2.printStackTrace();
      }
    }
}
```

CHAPTER 7

COMMON PACKAGE

This chapter describes the Common Package of the caCORE API.

Topics in this chapter include:

- *Introduction* on this page
- *Common Package API* on this page
- *Common Package Specific Utilities* on page 122

Introduction

The "common" package of the caCORE API contains objects that can be shared across all domain packages of caCORE to find information about data provenance and related data stores.

Common Package API

The following table lists each class in the Common Package and its description. Detailed descriptions about each class and its methods are present in the caCORE 3.2 JavaDocs at http://ncicb.nci.nih.gov/NCICB/content/ncicblfs/caCORE3-2_JavaDocs.

Common Domain Object	***Description***
DatabaseCrossReference	Provides links to related data hosted by other sources.
Source	A class representing a source. Will be subclassed into a variety of specialized classes.
InternetSource	An implementing subclass of Source. Describes a source for which an electronic online version is available.

Table 7.1 Classes in the Common Package and their descriptions

Common Domain Object	Description
PublicationSouce	An implementing subclass of Source. Describes a source for which an electronic online version is not available, but for which a printed version of the data is available.
ResearchInstitutionSource	An implementing subclass of Source that describes a research institution (commercial, academic, or government). This is used for information with attribution, which lacks an online electronic format.
SourceReference	A reference (an electronic reference, publication citation, etc.) to the untransformed data at a source.
URLSourceReference	An implementation of the abstract SourceReference that contains a URL to the original information.
WebServicesSourceReference	
Provenance	A record describing the source of an assertion (datum) contained in an object.

Table 7.1 Classes in the Common Package and their descriptions (Continued)

Common Package Specific Utilities

XMLUtility

The caCORE 3.2 release includes an *XMLUtility* class that can be used to transform java bean representations of domain objects to and from XML. The *XMLUtility* class takes advantage of the open source Castor XML project to handle this conversion. The utility uses the *caCOREMarshaller* and *caCOREUnmarshaller* classes contained within the `gov.nih.nci.common.util` package, which in turn uses Castor. However, the *XMLUtility* class was designed to be flexible and allow developers to create their own marshaller and unmarshaller classes should they choose to do so as long as the *gov.nih.nci.common.util.Unmarshaller* and *`gov.nih.nci.common.util.Marshaller`* interfaces are implemented.

The caCORE client also contains a file named `xml-mapping.xml` that is used by Castor to determine how to process an object's attributes and associations to and from xml. It is important to note that the `xml-mapping` file used in caCORE limits serialization and deserialization to an object's immediate attributes and not its associated relationships.

Also packaged with the caCORE client are xml schema files, one for each domain package. These xsd files can be used to validate xml representations of caCORE domain objects.

The following sample client demonstrates how to use the *XMLUtility* class to convert a caBIO *Gene* object to and from xml and also how to validate the xml against the `gov.nih.nci.cabio.domain.xsd` file:

```
public class TestXML {
    public static void main(String[] args) {


try{
       ApplicationService appService = ApplicationService.getRemoteInstance(
              "http://cabio.nci.nih.gov/cacore32/server/HTTPServer");
       try {
              Gene gene = new Gene();
              gene.setId(Long.valueOf(2));
              try {
              XMLUtility myUtil = new XMLUtility();
              List resultList = appService.search(Gene.class, gene);
              System.out.println("Result list size: " + resultList.size() + "\n");
              long startTime = System.currentTimeMillis();
              for (Iterator resultsIterator = resultList.iterator();
                     resultsIterator.hasNext();) {
                     Gene returnedGene = (Gene)resultsIterator.next();
                     System.out.println("Id: "+ returnedGene.getId() + "\n");
                     System.out.println("Fullname: "+ returnedGene.getFullName());
                     System.out.println("ClusterId: "+ returnedGene.getClusterId());
                     System.out.println("Symbol: "+ returnedGene.getSymbol());
                     File myFile = new File("C:/test.xml");
                     FileWriter myWriter = new FileWriter(myFile);
                     myUtil.toXML(returnedGene,myWriter);
                     DocumentBuilder parser
       DocumentBuilderFactory.newInstance().newDocumentBuilder();
                            Document document = parser.parse(myFile);
                     SchemaFactory factory =
       SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI);
                     Source schemaFile = new StreamSource(new
       File("C:/gov.nih.nci.cabio.domain.xsd"));
                     Schema schema = factory.newSchema(schemaFile);
                     Validator validator = schema.newValidator();
                     System.out.println("Validating gene against the schema......);
                     validator.validate(new DOMSource(document));
                     System.out.println("Retrieving gene from xml ....\n\n");
                     Gene myGene = (Gene) myUtil.fromXML(myFile);
                     System.out.println(“Id: "+ myGene.getId());
                     System.out.println("Fullname: "+ myGene.getFullName());
                     System.out.println("ClusterId: "+ myGene.getClusterId());
                     System.out.println(" Symbol: "+ myGene.getSymbol());
              }
       } catch (ParserConfigurationException e) {
              ea.printStackTrace();
       } catch (SAXException e) {
              eb.printStackTrace();
       } catch (IOException e) {
              ec.printStackTrace();
       } catch (Exception e) {
              e.printStackTrace();
       }
} catch (RuntimeException e) {
       e2.printStackTrace();
}
} catch (RuntimeException e) {
       e2.printStackTrace();
}

}
catch(Exception ex){
```

```
ex.printStackTrace();
}
}
}
```

CHAPTER 8

COMMON SECURITY MODULE

For a complete description of the Common Security Module, please refer to the document CSM_Guide_ApplicationDevelopers.pdf located on GForge.

CHAPTER 9

COMMON LOGGING MODULE

This chapter describes the Common Logging Module (CLM), which provides a separate service under caCORE for Audit and Logging Capabilities. The CLM also includes a web-based locator tool, which can be used by a client application directly without using any other components like CSM. For this purpose, it is a stand alone module.

Topics in this chapter include:

Introduction

This chapter provides all the information application developers need to successfully integrate with NCICB's Common Logging Module (CLM). The CLM was chartered to provide a comprehensive solution to Audit and Logging objectives and also to prevent the duplication of the effort and time involved in developing a logging solution. CLM is flexible enough to allow application developers to integrate audit logging into almost any code design with minimal coding effort. This phase of the Common Logging Module brings the NCICB team one step closer to the goal of CFR 21 / part 11 (FDA) compliance.To use the information in this chapter, begin by reading *CLM Overview* on page 128 to learn the CSM concepts and how they apply to your own application. Next, read *Workflow for CLM Integration* on page 131 to understand how to successfully integrate with CSM. *Deployment Models* on page 132 describes how to deploy the services and how to integrate with them.

This chapter shows how to deploy and integrate the CLM services, including event logging and automated object state logging. It also shows how to deploy and configure the web locator tool for the purpose of browsing through the logs.

Begin by reading *CLM Overview* on page 128 to learn the CLM concepts and how they apply to your own application. Next read *Workflow for CLM Integration* on page 131 to understand how to successfully integrate with CLM. Finally, *Deployment Models* on page 132 describes how to deploy CLM services.

CLM Overview

Explanation

CLM is a powerful set of auditing and logging tools implemented in a flexible and comprehensive solution. CLM provides the following features:

- **Event Logging** - This feature provides a log4j-based solution allowing users to log events. Since this feature provides the ability to propagate and store user information, it can be used for anything from auditing a user to detecting security breaches.
- **Automated Object State Logging** - This tool provides an automated, hibernate based, object state logging mechanism used to log the changes to an object's state.
- **Asynchronous Logging to database** - This feature provides a log4j based JDBC appender that can log the messages to the database asynchronously. Asynchronous logging increases performance for applications that generate a high volume of log messages.
- **Query APIs** - This feature provides the interface to query the log messages programmatically. The Query APIs allows specifying a SearchCriteria and methods to retrieve the results in various ways. Refer the JavaDocs for details. The web-based log locator tool uses Query APIs to retrieve the logs.
- **Web-based Log Locator tool (LLT)** - This tool is a web-based application that displays, searches, and filters log messages. The interface can be used to retrieve logs based on a timestamp, user ID, Object ID, Object Name, Operation, log level, etc. Refer the LLT Application Developers Guide for more information on LLT.

Figure 9.1 shows how CLM works with an application and independent entities, such as the credential providers and authorization schema, to perform authentication and authorization.

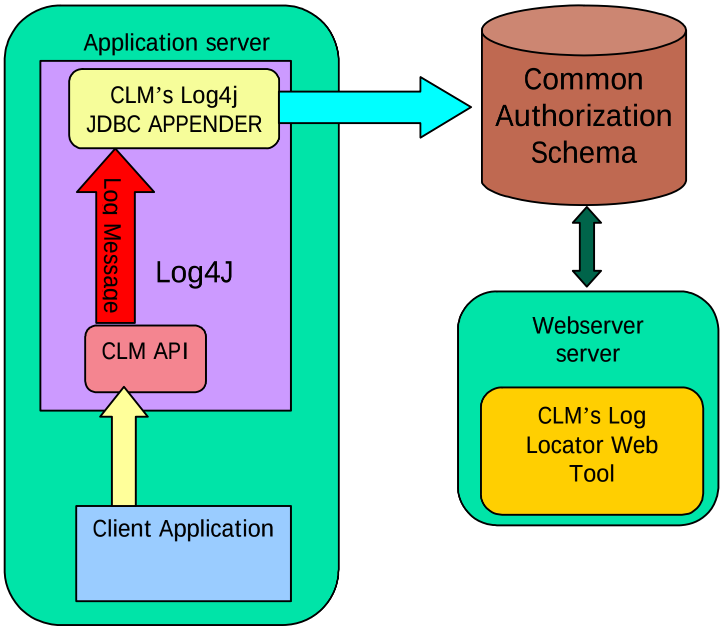

Application server
CLM's Log4j JDBC APPENDER
Log Message
Log4J
CLM API
Client Application
Common Authorization Schema
Webserver server
CLM's Log Locator Web Tool

Figure 9.1 CLM interactions Audit Logging and Web Locator Tool

The CLM APIs provide the following major components of the Audit and Logging capabilities provided by CSM.

Event Logger

CLM APIs are based on log4j framework. To allow the client application to integrate easily, this framework provides its logging capabilities through the log4j logger. To log an event for the purpose of audit tracking, a client application must first obtain a logger using the log4j's Logger class. Using this logger, the client class can log all the events. Obtaining this logger is similar to obtaining a logger for regular logging from log4j. It is advisable to maintain the same logger name while obtaining the logger in various classes. Also it is advisable to the keep the log level consistent. By keeping these elements consistent makes configuring this logger easier in the log4j configuration file.

The events that an application can log for audit purposes include: logins, logouts, invalid login attempts, data manipulations actions triggered by the user etc. The event is actually passed as a string message to log4j. This message is persisted into a persistent store using a special appender provided by CLM. In order to enable this logger, it should be configured in the `log4j.xml` config file of JBoss.

Object State Logger

Along with the Event Logger, CLM provides an automated Object State Logger for Hibernate. This Logger automatically track changes in the object state whenever the object is updated, created or deleted using hibernate. It implements an interceptor which listens to the hibernate session. This logger is transaction aware and only logs the object state changes if the transaction is committed. However in case of a transaction rollback, the logs are discarded. Whenever the client application performs any create, update or delete operation on this session, the interceptor is invoked. This interceptor introspects the object and converts its contents into a message string. Finally, it logs it using log4j logger. When there is a create or delete operation, the current state of the object is logged. However, when there is an update operations both the previous object state and the current values are logged. However if any other operation is performed on the object within the context of the same transaction then the previous state of the object is lost, and hence unavailable for logging.

In order to use the object state logger, the client application needs to obtain the session using the help class provided by the CLM APIs. The session returned is a hibernate session with an attached interceptor to track all the object state changes. This is the only code specific to object state logger that is required to use this facility. The object state logger also needs to be configured in the ObjectStateLoggerConfig.xml file. Here the application needs to specify the logger name that it intends to use for the object state log4j logger. Also the log4j log level needs to be specified to set the corresponding log4j log level for all the object state logs. A list of objects that needs to be logged should also be added in this file. If no objects are specified then the object state loggers ignores all the object state changes and no logs are generated. In order to enable this object state logger, it should be configured in the `log4j.xml` config file of JBoss.

User Information

In order to track which user is performing the specific operation for the purpose of Audit Logging, CLM needs to know user information like user id and session id. Since these values are only available with the client application, they need to be passed to the CLM APIs. To accomplish this, the client application must use the utility class "UserInfoHelper" provided by the CLM APIs. This information needs to be set before any of the event logs or any hibernate actions. Also this information needs to be set only once throughout a thread of execution which caters to a client request. It means, in case of a web application, this information should be set only once during the start of the doPost method.

Common Logging Database

This is the persistence storage that the JDBC appender uses to store the Audit Logs. The Log Locator application of CLM connects to this database to allow the user to browse the logs. This database consists of tables that consists of fields which are used to store the audit message and Object Attributes. A common schema can be deployed and shared across applications. The application name is part of the log message and can be used to pull logs only for a particular application. Admin for a particular application can view logs only for that application. CSM is used to security and admins need privileges to access logs of particular applications. Please refer the security section for details.

JDBC Appender

To persist the generated audit logs the CLM provides an asynchronous JDBC Appender. This appender is asynchoronous and maintains a configurable internal buffer. This buffer is used to store the logs before a parallel thread is spawned which writes these logs into the database. This appender is also capable of extracting the user information set by the client application and uses it while writing the data into the database. Thus, an application that wants to enable the audit logging should also configure this Appender. A sample log4j entry for the CSM APIs is show below.

```
<appender name="CLM_APPENDER" class="gov.nih.nci.logging.api.appender.jdbc.JDBCAppender">
        <param name="application" value="csm" />
        <param name="maxBufferSize" value="1" />
        <param name="dbDriverClass" value="com.mysql.jdbc.Driver" />
        <param name="dbUrl" value="jdbc:mysql://cbiodev104.nci.nih.gov:3306/clmlog" />
        <param name="dbUser" value="user" />
        <param name="dbPwd" value="password" />
        <param name="useFilter" value="true" />
        <layout class="org.apache.log4j.PatternLayout">
                <param name="ConversionPattern" value=":: [%d{ISO8601}] %-5p %c{1}.%M() %x - %m%n"
/>
        </layout>
</appender>
<logger name="CSM.Audit.Logging.Event.Authentication">
        <level value="info" />
        <appender-ref ref="CLM_APPENDER" />
</logger>
<logger name="CSM.Audit.Logging.Event.Authorization">
        <level value="info" />
        <appender-ref ref="CLM_APPENDER" />
</logger>
<logger name="CSM.Audit.Logging.ObjectState.Authoriaztion">
        <level value="info" />
        <appender-ref ref="CLM_APPENDER" />
</logger>
```

Figure 9.2 Example log4j.xml file

Note: CSM is capable of performing both event and object state audit logging only for the operations and data pertaining to CSM. In order to use the similar functionality, the client application can separately download and install CLM. In this case CLM can be used (even without using CSM) to provide event logging and automated object state logging capabilities using the special appender and schema. Also the log locator tool can be used for the purpose of viewing the logs.

Query APIs

To retrieve the generated logs, CLM Query APIs provide a very easy and flexible Search Criteria to query the logs from the CLM log database.

Workflow for CLM Integration

This workflow section outlines the basic steps, both strategic and technical, for successful CLM integration.

1. Decide which services of CLM you want to integrate with an application. If the application needs to maintain an audit log of each and every action that takes place then you can use the CLM's Audit logging capabilities. If the application needs to log object state changes for audit purposes then use CLM's automated object logger capabilities.
2. Read the CLM Guide for Application Developers (this chapter). It provides an overview, workflow, and specific deployment and integration steps.
3. Determine a logging strategy. Based on the logging mode selected configure the appropriate configuration files. Also set up the logging database to capture all the audit logs as discussed in *Deployment Steps* on page 133.

4. Deploy the web-based tool for the purpose of browsing through the logs. Enter the configuration details to point to the database server that holds the logs
5. Integrate the application code using the integration steps for EventLogging, and/or ObjectStatelogging
6. Test and refine CLM integration with your application. Confirm that your audit logging implementation meets requirements.

Deployment Models

CLM APIs

Introduction

The logging API's facilitate the audit and logging needs of a developer at run time. These APIs use Java and log4j for the purpose of logging, so it is assumed that developers are familiar with both.

Software Products

Table 9.1 displays descriptions of software products used for logging.

Software Product	***Description***
JBoss Server	The JBoss/Server is the leading open-source, standards-compliant, J2EE-based application server implemented as pure Java. A majority of caCORE applications use this server to host their applications.
MySQL Database	MySQL is an open source database. Its speed, scalability, and reliability make it a popular choice for web developers. CLM recommends storing authorization data in a MySQL database because it is a light database and is easy to manage and maintain.
Hibernate	Hibernate is an object/relational persistence and query service for Java. CSM requires developers to modify a provided Hibernate configuration file (`hibernate.cfg.xml`) in order to connect to the appropriate application authorization schema.
Log4j	Log4j is a open source logging framework. It is used by an application for the purpose of logging. It provides a configurable mechanism for an application to be able to persist logs onto different media.

Table 9.1 Logging software products

Configuration and SQL Files

File	Description
`ObjectStateLogger-Config.xml`	The XML file containing the configuration data for the Object State Logger. It contains the log level, the object for which the automated logging should be enabled etc.
`log4j.xml`	The log4j configuration file provided by the JBoss server. It contains the details about configuration for the Event Logger and the Object State Logger.
`mysql_log_table.sql`	This Structured Query Language (SQL) script is used to create an instance of the logging database schema that will be used for the purpose of storing log messages.

Table 9.2 Logging configuration and SQL files

Overview to Integrating CLM APIs

This section provides instruction for integrating the CLM APIs with Jboss-based web application. The integration is flexible enough to meet the needs for other deployment scenarios such as stand alone or enterprise applications.

Deployment Steps

Step 1: Create and Prime MySQL

1. Log into the database using an account ID that has permission to create new databases.
2. If the log database is not already created, then run the `CLMSchemaMySQL.sql` script at the database prompt. This creates the CLM database

Step 2: Placing the CLM APIs Jar

1. The CLM application APIs are available as a JAR that must be placed in the classpath of the application. Along with this JAR, there are many supporting JARs on which the CLM API depends. These extra JARs are supplied as part of the CLM distribution and should be added to the folder `<application-web-root>\WEB-INF\lib`.

   Note: Based on the class loader used in JBoss, the `clmwebapp.jar` file might need to be placed in the `WEB-INF/lib` directory instead of the `clm.jar`. In this case the `clm.jar` should be placed in the lib directory of JBoss so that the JDBC appender is available to the log4j service of JBoss.

Step 3: Configure ObjectStateLoggerConfig.xml

If the application plans to use the Object State Logger provided by CLM then the `ObjectStateLoggerConfig.xml` file needs to be configured.

1. The following entries should be configured based on the application's requirements

   a. **ObjectStateLoggerName**: Name of the Logger used in the LoggerConfigurationFile. The name of the logger should match with the `log4j-`

`config.xml` logger value. That is, `log4j-config.xml` should have a logger with ObjectStateLoggerName.

b. **LoggerConfigurationFile**: This is the log4j configuration file for Audit Logging. It can be configured to log messages either to a flat text file or to a database table via a JDBC Appender.

c. **LogLevel**: This is the log level for Audit Logging. It can be one of those five levels provided by the log4j framework.

d. **LoggingEnabled**: This flag indicates whether or not the audit logging is enabled for the client application. It works with the above domainObjectList context. This flag will take either TRUE or FALSE as input. If the value for this tag is FALSE, then it will disable the audit logging for all classes. If it is set to TRUE, then only those classes that are listed in the domainObjectList will be audited.

e. **ObjectStateLoggerMessageFormat**:This is the message format for the log messages. It can be configured to be String or XML. In the case of String, the log message will be in the string format. In the case of XML, the log message will be generated in xml files.

f. **DomainObjectList**: It is the list of DomainObjects whose state changes are to be logged. This is the context to put a list of fully qualified classes names to be audited. Each fully qualified class needs to be placed within the ObjectName name tag within DomainObject.

g. **DomainObject**: Object whose changes are to be logged.

h. **ObjectName**: ObjectName of the domain object. Example: sample.package.ClassName.

i. **IdentifierAttribute**: IdentifierAttribute is the name of the object property that is to be used save in logs object ID column. This value will be matched with query Object ID for retrieval. Example: For object with attributes: personname, street, zip. < IdentifierAttribute>personname</ IdentifierAttribute>. In this example, the attribute 'personname' will be used to store logs with ObjectId = Name. Example: For object with attributes: first, last, street, zip. <IdentifierAttribute>first,last</ IdentifierAttribute>. In this example, the composite attributes value of 'first+last' will used to store logs with ObjectID = 'first+last'.

Figure 9.3 is an example of the `ObjectStateLoggerConfig.xml` file for the CSM APIs integration with CLM.

```
<ObjectStateLoggerConfig>
        <ObjectStateLoggerName>CSM.Audit.Logging.ObjectState.Authorization</ObjectStateLoggerName>
        <LoggerConfigurationFile>log4jConfig.xml</LoggerConfigurationFile>
        <LogLevel>info</LogLevel>
        <LoggingEnabled value="true" />
        <ObjectStateLoggerMessageFormat type="string" />
        <DomainObjectList>
                <DomainObject>

        <ObjectName>gov.nih.nci.security.authorization.domainobjects.Application</ObjectName>
                        <IdentifierAttribute>applicationDescription</IdentifierAttribute>
                </DomainObject>
                <DomainObject>

        <ObjectName>gov.nih.nci.security.authorization.domainobjects.ApplicationContext</ObjectName>
                        <IdentifierAttribute>applicationName</IdentifierAttribute>
                </DomainObject>
                <DomainObject>

        <ObjectName>gov.nih.nci.security.authorization.domainobjects.Group</ObjectName>
                        <IdentifierAttribute>groupName</IdentifierAttribute>
                </DomainObject>
                <DomainObject>

        <ObjectName>gov.nih.nci.security.authorization.domainobjects.Privilege</ObjectName>
                        <IdentifierAttribute>name</IdentifierAttribute>
                </DomainObject>
                <DomainObject>

        <ObjectName>gov.nih.nci.security.authorization.domainobjects.ProtectionElement</ObjectName>
```

Figure 9.3 Example ObjectStateLoggerConfig.xml file

2. Place the `ObjectStateLoggerConfig.xml` in the classpath of the client application. In case of a web application, the path is the folder `WEB-INF/classes`.

Step 4: Configure log4j.xml

1. Modify the log4j configuration file to configure the JDBC Appender provided by the CLM APIs. If using Jboss, the log4j configuration file is called `log4j.xml` and can be found in the conf directory of the JBoss server.
2. An entry for the appender should be made. Set the following properties for the appender entry:
    a. **name**: The name of the appender.
    b. **class**: The fully qualified path of the CLM's JDBC Appender `gov.nih.nci.logging.api.appender.jdbc.JDBCAppender`.
    c. **application**: The name of the application for which the logging is performed.
    d. **maxBufferSize**: The size of the asynchronous buffer before which the message would be logged into the database.
    e. **dbDriverClass**: The name of the jdbc driver to be used to connect to the database.
    f. **dbUrl**: The URL of the server where the server is hosted.
    g. **dbUser**: The user id to be used to connect to the database.

   h. **dbPwd**: The password to be used to connect to the database.
   i. **useFilter**: Denotes whether the logger should use a filter or not.

3. Once the appender has been configured then the logger should be configured. Entries made in the `log4j.xml` file should be based on the logger name used in the `ObjectStateLoggerConfig.xml` file for the object state logger and the logger name used for event logging. For the logger entry in the `log4j.xml` file, set the following parameters:
   a. **name**: The name of the logger.
   b. **level**: The minimum log level that this logger should log.
   c. **appender-ref**: The name of the appender invoked for logging. It should be same as the appender name used in the previous step.

Figure 9.4 is an example of the `log4j.xml` file for the CSM APIs integration with CLM.

```
<appender name="JDBC_MySql" class="gov.nih.nci.logging.api.appender.jdbc.JDBCAppender">
  <param name="application" value="csm" />
  <param name="maxBufferSize" value="1" />
  <param name="dbDriverClass" value="com.mysql.jdbc.Driver" />
  <param name="dbUrl" value="jdbc:mysql://<<Database URL>>" />
  <param name="dbUser" value="<<userID>>" />
  <param name="dbPwd" value="<<Password>>" />
  <param name="useFilter" value="true" />
  <layout class="org.apache.log4j.PatternLayout">
    <param name="ConversionPattern" value=":: [%d{ISO8601}] %-5p %c{1}.%M() %x - %m%n" />
  </layout>
</appender>

<logger name="CSM.Audit.Logging.Event.Authentication">
  <level value="info" />
  <appender-ref ref="JDBC_MySql" />
</logger>
<logger name="CSM.Audit.Logging.Event.Authorization">
  <level value="info" />
  <appender-ref ref="JDBC_MySql" />
</logger>
<logger name="CSM.Audit.Logging.ObjectState.Authoriaztion">
  <level value="info" />
  <appender-ref ref="JDBC_MySql" />
</logger>
```

Figure 9.4 Example log4j.xml file

Integrating with the CLM's Audit Logging Services

Importing and Using the Audit Logging Classes

To use the Audit Logging service, add the highlighted import statements as shown in the following:

```
import org.apache.log4j.Logger;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import
gov.nih.nci.logging.api.logger.hibernate.HibernateSessionF
actoryHelper;
import gov.nih.nci.logging.api.user.UserInfoHelper;
```

Getting a Hibernate Session for Audit Logging

First, obtain a Hibernate session using one of the following methods:

```
Session session = 
HibernateSessionFactoryHelper.getDefaultAuditSession();
```

Or

```
Session session = 
HibernateSessionFactoryHelper.getAuditSession(SessionFacto
ry sessionFactory)
```

Use the first line of code if you want to obtain the session using the default session factory obtained from the default `hibernate.cfg.xml` file.

If there is already a session factory created in the application, use the existing session factory to obtain an audit session by using the second method listed above.

By obtaining the Audit Session, the automated object state logger is engaged and tracks all the object state changes performed using this session.

Note: The CLM object state logger has issues logging if the transaction managers are set in the `hibernate.cfg.xml` file when deployed on JBoss server. In this case the transaction manager properties should be removed from the `hibernate.cfg.xml` file used for CSM APIs to connect to the common authorization schema.

Setting User Information for the Client Application

The Audit Logging service has the capability to log messages that include user information. session id and organization. The code to do this needs to be written only once per a client request thread in the client application as follows:

```
UserInfoHelper.setUserInfo(new String("NAME"), new 
String("sessionId"));
```

Or

```
UserInfoHelper.setUserInfo("john", "sessionId, 
"organizationA");
```

Obtaining the Event Logger

Obtain the event logger as you would obtain a regular log4j logger. The logger name passed is used in the log4j configuration file to enable logging for these messages.

Logging from the Client Application

For event logging, use the log4j logger class in the normal way. Based on the log level configured in the log4j configuration for the appender these messages are logged.

APPENDIX A

UNIFIED MODELING LANGUAGE

The caCORE team bases its software development primarily on the Unified Modeling Language (UML). This appendix is designed to familiarize the reader who has not used UML with its background and notation.

Topics in this appendix include:

Note: Throughout this guide, references to the Unified Modeling Language refer to the approved version 1.3 of the standard.

UML Modeling

The UML is an international standard notation for specifying, visualizing, and documenting the artifacts of an object-oriented software development system. Defined by the Object Management Group, the UML emerged as the result of several complementary systems of software notation and has now become the *de facto* standard for visual modeling. For a brief tutorial on UML, refer to http://bdn.borland.com/article/0,1410,31863,00.html.

The underlying tenet of any object-oriented programming begins with the construction of a model. In its entirety, the UML version 1.3 is composed of nine different types of modeling diagrams that form, in essence, a software blueprint.

Only a subset of the diagrams, that used in caCORE development, is described in this appendix.

- Use-case diagrams
- Class diagrams
- Package diagrams
- Component diagrams
- Sequence diagrams

The caCORE development team applies use-case analysis in the early design stages to informally capture high-level system requirements. Later in the design stage, as classes and their relations to one another begin to emerge, class diagrams help to define the static attributes, functionalities, and relations that must be implemented. As design continues to progress, other types of interaction diagrams are used to capture the dynamic behaviors and cooperative activities the objects must execute. Finally, additional diagrams, such as the package and sequence diagrams can be used to represent pragmatic information such as the physical location of source modules and the allocation of resources.

Each diagram type captures a different view of the system, emphasizing specific aspects of the design such as the class hierarchy, message-passing behaviors between objects, the configuration of physical components, and user interface capabilities.

Note: Not all UML artifacts discussed in this appendix are necessary for using caCORE. They are included in this appendix to provide a more complete overview of UML.

While many good development tools provide support for generating UML diagrams, the Enterprise Architect (EA) software is used throughout caCORE. The resulting documents, originally generated during design and development, provide value throughout the software life cycle as they can rapidly familiarize new users of the system with the logic and structure of the underlying design elements.

Use-case Documents and Diagrams

A good starting point for capturing system requirements is to develop a structured *textual* description, often called a use-case document, of how users will interact with the system. While there is no hard and fast predefined structure for this artifact, use-case documents typically consist of one or more actors, a process, a list of steps, and a set of pre- and post-conditions. In many cases, it describes the post-conditions associated with success as well as failure. An example use-case document is represented in *Figure A.1*.

Find Gene(s) for a given search criteria (keyword)

<u>Usecase ID:100300</u>

<u>Actor</u>

- caBIO Application developer

<u>Starting Condition</u>

The actor establishes reference to the caBIO software

<u>Flow of Events</u>

1. The actor sets the search criteria (Use case ID 101300) using one or more keywords in the criteria.
2. Invoke the search use case (Use case ID 105300) and pass the search criteria instantiated at step 1.
3. A result set (Use case ID 110300) is returned to the actor.

<u>End Condition</u>

The actor has obtained a collection of Genes needed for his application.

Figure A.1 Example use-case document

Using the use-case document as a model, a use-case diagram is then created to confirm the requirements stated in the text-based use-case document.

A use-case diagram, which is language independent and graphically described, uses simple ball and stick figures with labeled ellipses and arrows to show how users or other software agents might interact with the system. The emphasis is on *what* a system does rather than *how* a system works. Each "use-case" (an ellipse) describes a particular activity that an "actor" (a stick figure) performs or triggers. The "communications" between actors and use-cases are depicted by connecting lines or arrows.

The example use-case diagram in *Figure A.2* can be interpreted as follows:

- A caBIO application developer triggers the actions to build a search query, connect to the server, and search the server.
- The caBIO application developer receives the output from the search.

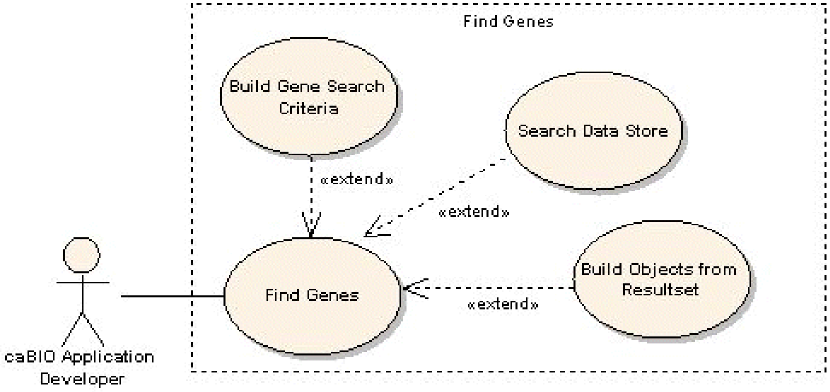

Find Genes
Build Gene Search Criteria
Search Data Store
«extend»
«extend»
Find Genes
«extend»
Build Objects from Resultset
caBIO Application Developer

Figure A.2 Example use-case diagram

Class Diagrams

The system designer utilizes use-case diagrams to identify the classes that must be implemented in the system, their attributes and behaviors, and the relationships and cooperative activities that must be realized. A class diagram is used later in the design process to give an overview of the system, showing the hierarchy of classes and their static relationships at varying levels of detail. *Figure A.3* shows an abbreviated version of a UML Class diagram depicting many of the caBIO domain objects.

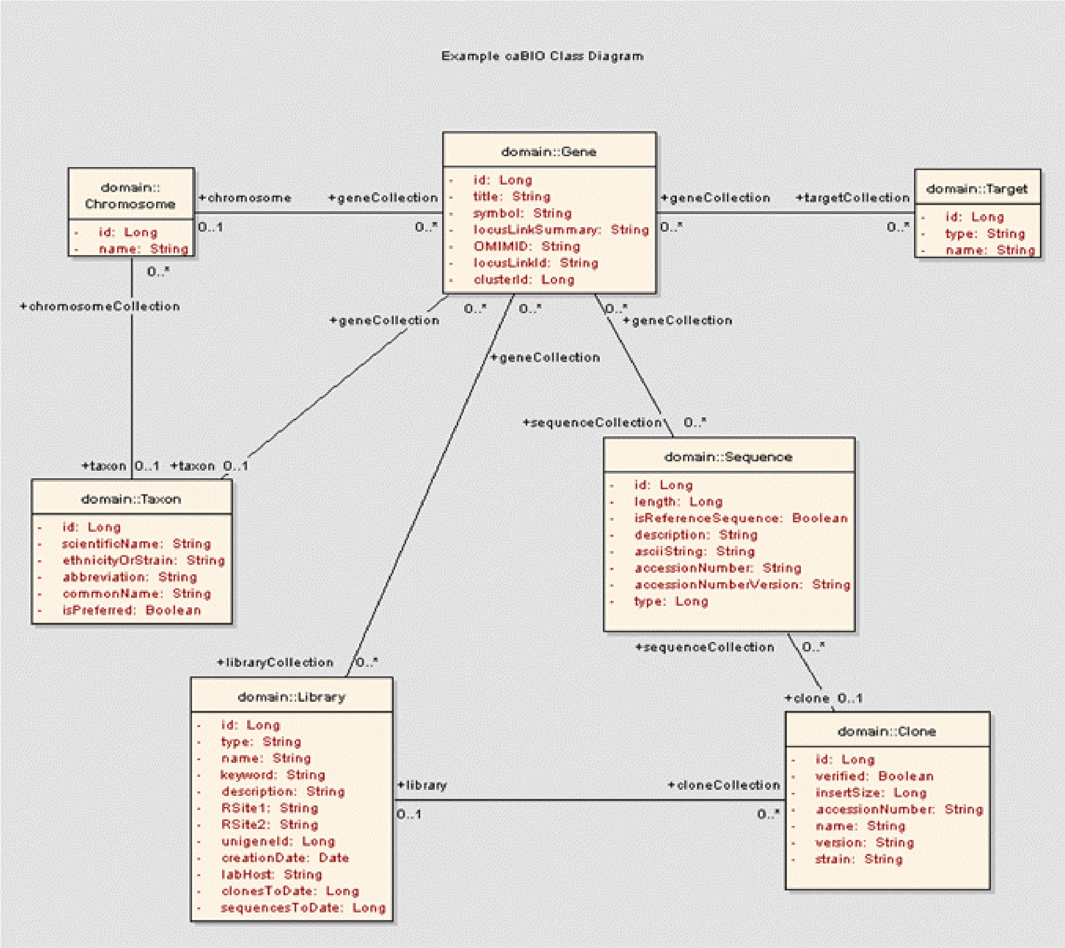


Figure A.3 Example UML Class diagram depicting the caBIO domain objects

Class objects can have a variety of possible relationships to one another, including “is derived from,” “contains,” “uses,” “is associated with,” etc. The UML provides specific notations to designate these different kinds of relations, and enforces a uniform layout of the objects’ attributes and methods — thus reducing the learning curve involved in interpreting new software specifications or learning how to navigate in a new programming environment.

Figure A.4 (a) is a schematic for a UML class representation, the fundamental element of a class diagram. *Figure A.4* (b) is an example of how a simple class might be represented in this scheme. The enclosing box is divided into three sections: The topmost section provides the name of the class, and is often used as the identifier for the class; the middle section contains a list of attributes (structures) for the class; (the attribute in the class diagram maps into a column name in the data model and an attribute within the Java class); the bottom section lists the object's operations (methods). *Figure A.4* (b) specifies the *Gene* class as having a single attribute called *sequence* and a single operation called *getSequence().*

Class
-attribute
+operation()
(a)
Gene
-sequence
+getSequence()
(b)

Figure A.4 (a) Schematic for a UML class (b) A simple class called Gene

Naming Conventions

Naming conventions are important when creating class diagrams. caCORE follows the formatting convention for Java APIs in that a class starts with an uppercase letter and an attribute starts with a lowercase letter. Names contain no underscores. If the name contains two words, then both words are capitalized, with no space between words. If an attribute contains two words, the second word is capitalized with no space between words. Boolean terms (has, is) are used as prefixes to words for test cases.

The operations and attributes of an object are called its features. The features, along with the class name, constitute the signature, or classifier, of the object. The UML provides explicit notation for the permissions assigned to a feature, and UML tools vary with respect to how they represent their private, public, and protected notations for their class diagrams.

The caBIO classes represented in *Figure A.3* show only class names and attributes; the operations are suppressed in that diagram. This is an example of a UML *view*: Details are hidden where they might obscure the bigger picture the diagram is intended to convey. Most UML design tools provide means for selectively suppressing either or both attributes and operation compartments of the class without removing the information from the underlying design model. In *Figure A.3*, the emphasis is on the relationships and attributes that are defined among the objects, rather than on operations.

The following notations (as shown in *Figure A.3* and *Figure A.7*) are used to indicate that a feature is public or private:

- "-" prefix signifies a private feature
- "+" signifies a public feature

In *Figure A.4* for example, the *Gene* object's *sequence* attribute is private and can only be accessed using the public *getSequence () method*.

Relationships Between Classes

Note: Not all figures used in this chapter appear in the demonstration class diagram, *Figure A.3*. They are, however, examples of models that may be found in caCORE.

A quick glance at *Figure A.3* demonstrates relationships between some of the classes. Generally, the relationships occurring among the caBIO objects are of the following types: association, aggregation, generalization, and multiplicity, described as follows:

Association **—** The most primitive of these relationships is association, which represents the ability of one instance to send a message to another instance. Association is depicted by a simple solid line connecting the two classes.

Directionality **—** UML relations can have directionality (sometimes called navigability), as in *Figure A.5*. Here, a *Gene* object is uniquely associated with a *Taxon* object, with an arrow denoting bi-directional navigability. Specifically, the Gene object has access to the Taxon object (i.e., there is a *getTaxon()* method), and the Taxon object has access to the Gene object. (There is a corresponding *getGeneCollection()* method.) Role names also display in *Figure A.3* and *Figure A.5*, clarifying the nature of the association between the two classes. For example, a taxon (rolename identified in *Figure A.5*) is a line item of each Gene object. The (+) indicates public accessibility.

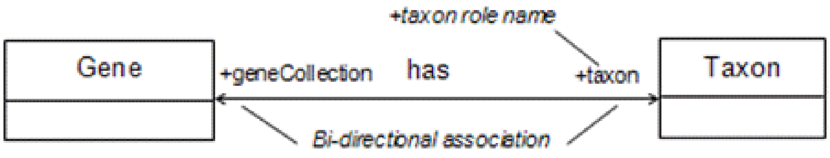


Figure A.5 A one-to-one association with bi-directional navigability

Multiplicity **—** Optionally, a UML relation can have a label providing additional semantic information, as well as numerical ranges such as 1..n at its endpoints, called multiplicity. These cardinality constraints indicate that the relationship is one-to-one, one-to-many, many-to-one, or many-to-many, according to the ranges specified and their placement. Table *A.1* displays the most commonly used multiplicities.

Multiplicities	***Interpretation***
0..1	Zero or one instance. The notation n..m indicates n to m instances.
0..* or *	Zero to many; No limit on the number of instances (including none). An asterisk (*) is used to represent a multiplicity of many.
1	Exactly one instance
1..*	At least one instance to many

Table A.1 Multiplicities table

Figure A.6 depicts a bidirectional many-to-one relationship between *Sequence* objects and *Clone* objects. Each Sequence may have at most one Clone associated with it, while a Clone may be associated with many Sequences. To get information about a Clone from the Sequence object requires calling the *getSequenceClone()* method. Each Clone in turn can return its array of associated Sequence objects using the *getSequences()* method. This bidirectional relationship is shown using a single undirected line between the two objects.

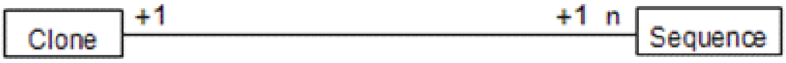


Figure A.6 A bidirectional many-to-one relation

Aggregation — Another relationship exhibited by caCORE objects is aggregation, in which the relationship is between a whole and its parts. This relationship is exactly the same as an association, with the exception that instances cannot have cyclic aggregation relationships (i.e., a part cannot contain its whole). Aggregation is represented by a line with a diamond end next to the class representing the whole, as shown in the *Clone-to-Library* relation of *Figure A.7*. As illustrated, a Library can contain Clones but not vice-versa.

In the UML, the empty diamond of aggregation designates that the whole maintains a *reference* to its part. More specifically, this means that while the Library is composed of Clones, these contained objects may have been created prior to the Library object's creation, and so will not be automatically destroyed when the Library goes out of scope.

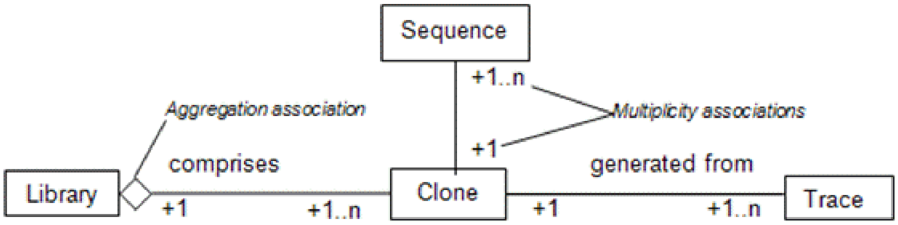


Figure A.7 Aggregation and multiplicity associations

Additionally, *Figure A.7* shows a more complex network of relations. This diagram indicates that:

a. one or more Sequences is associated with a *Clone*

b. the *Clone* is contained in a *Library*, which comprises one or more *Clones*

c. the *Clone* may have one or more *Traces*.

Only the relationship between the Library and the Clone is an aggregation. The others are simple associations.

Generalization — Generalization is an inheritance link indicating that one class is a subclass of another. *Figure A.8* depicts a generalization relationship between the *SequenceVariant* parent class and the *Repeat* and *SNP* classes. Classes participating in generalization relationships form a hierarchy, as depicted here.

In generalization, the more specific element is fully consistent with the more general element (it has all of its properties, members, and relationships) and may contain additional information. Both the *SNP* and *Repeat* objects follow that definition.

The superclass-to-subclass relationship is represented by a connecting line with an empty arrowhead at its end pointing to the superclass, as shown in the *SequenceVariant-to-Repeat* and *SequenceVariant-to-SNP* relations of *Figure A.8*.

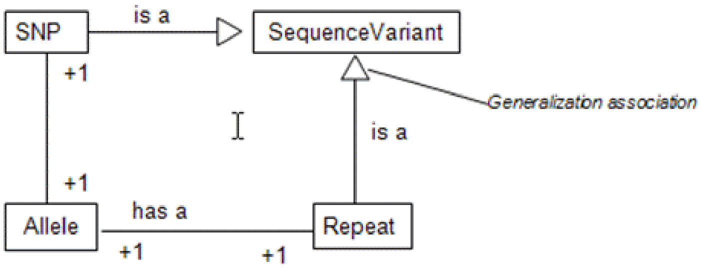


Figure A.8 Generalization relationship

In summary, class diagrams represent the static structure of a set of classes. Class diagrams, along with use-cases, are the starting point when modeling a set of classes. Recall that an object is an instance of a class. Therefore, when the diagram references objects, it is representing dynamic behavior, whereas when it is referencing classes, it is representing the static structure.

Package Diagrams

Large-scale software design is a highly complex activity. As the number of classes grows to satisfy the evolving requirements of an application, the overall architectural design can quickly become obscured by this proliferation of design elements. To simplify complex UML diagrams, classes can be organized into packages representing logically related groupings. Packaging can be applied to any type of UML diagram; a package diagram is any UML diagram composed only of packages.

Most commonly, packaging is used to simplify use-case and class diagrams. The package diagram is not one of the nine standard UML diagrams, but since it provides a convenient way of depicting the organization of software components into packages, it is described here.

A package is depicted as a labeled rectangle with a small tab attached to its upper left corner, somewhat resembling a file folder (*Figure A.9*). This image represents a package diagram generated in EA. “gov” is the top level package; “+nih” is a sub-package to gov, with the “+” indicating that sub-packages to nih exist. The dotted arrows connecting packages as displayed in *Figure A.10* represent dependencies: one package depends on another if changes in one could force changes in the other. This figure is the hierarchical representation of *Figure A.9*.

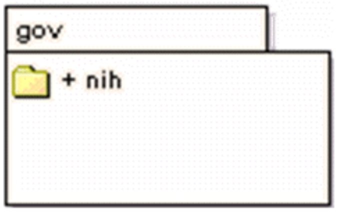


Figure A.9 Package diagram generated in EA

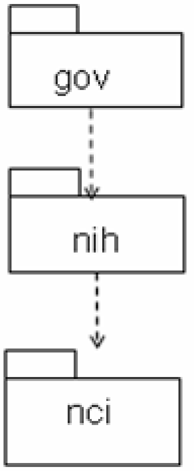


Figure A.10 Sub-package diagram

The concept of a package in a software application is similar but not identical to the notion of a UML package.

The organization of software components into packages is used to increase reusability and to minimize compile-time dependencies. It is highly unusual to reuse a single class, but quite common to reuse a collection of related classes that collaborate to produce some desired functionality. The UML models of the caCORE software that are available on the JavaDocs pages approximately reflect the actual Java package structure but do not have a one-to-one correspondence.

Component Diagrams

A component diagram is a physical analog of a class diagram. Its purpose is to show the organizations and dependencies among various software components comprising the system, including source code components, run time components, or executable components.

In complex systems, the physical implementation of a defined service is provided by a group of classes rather than a single class. A component is an easy way to represent the grouping together of such implementation classes.

A Component diagram consists of the following:

- Component
- Class/Interface/Object
- Relation/Association

A generic component diagram's main icon is a rectangle that has two rectangles overlaid on its left side (*Figure A.11*). The *component* name appears inside the icon. If the *component* is a member of a *package*, you can prefix the *component's* name with the name of the *package*.

Figure A.12 represents a component diagram as it is represented in EA.

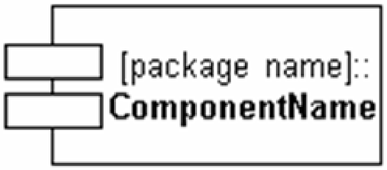


Figure A.11 Generic component diagram

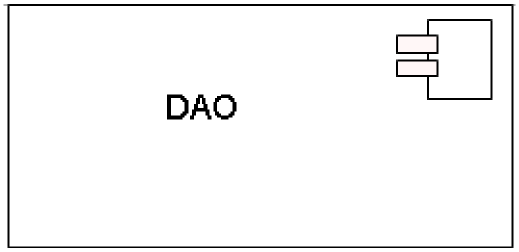


Figure A.12 Component diagram as represented in EA

Component diagrams and class diagrams represent both the static structure and the dynamic behavior of the system. Component diagrams are optional since they are not used for code generation.

Sequence Diagrams

A sequence diagram describes the exchange of messages being passed from object to object over time. The flow of logic within a system is modeled visually, validating the logic of a usage scenario. In a sequence diagram, bottlenecks can be detected within an object-oriented design, and complex classes can be identified.

Figure A.13 is an example of a sequence diagram. The vertical lines in the diagram with the boxes along the top row represent instantiated objects. The vertical dimension displays the sequence of messages in the time order that they occur; the horizontal dimension shows the object instances to which the messages are sent. The diagram is read from left to right, top to bottom, following the sequential execution of events.

This sequence diagram explains the sequence of execution of the toolkit at runtime.The User query from the client traverses the following sequence path before reaching the database.

1. The user uses *search()* method in ApplicationService and queries the server.
2. This call is picked up at HTTPClient as *query()* with *Request* as the input parameter
3. HTTPClient calls the HTTPServer (Interface Proxy for HTTP Tunneling) and sends the same *Request* to *BaseDelegate*
4. *BaseDelegate* calls *ServiceLocator* to find the name of *Data Access Object*.
5. Using this name *BaseDelegate* creates the corresponding DAO factory and passes the *Request* object.
6. In this scenario the ORMDAO is the right DAO to be called.
7. ORMDAOImpl contains specific implementation about the data source and connects to the data source.

Note: Sequence diagrams are optional, since they are not used for code generation.

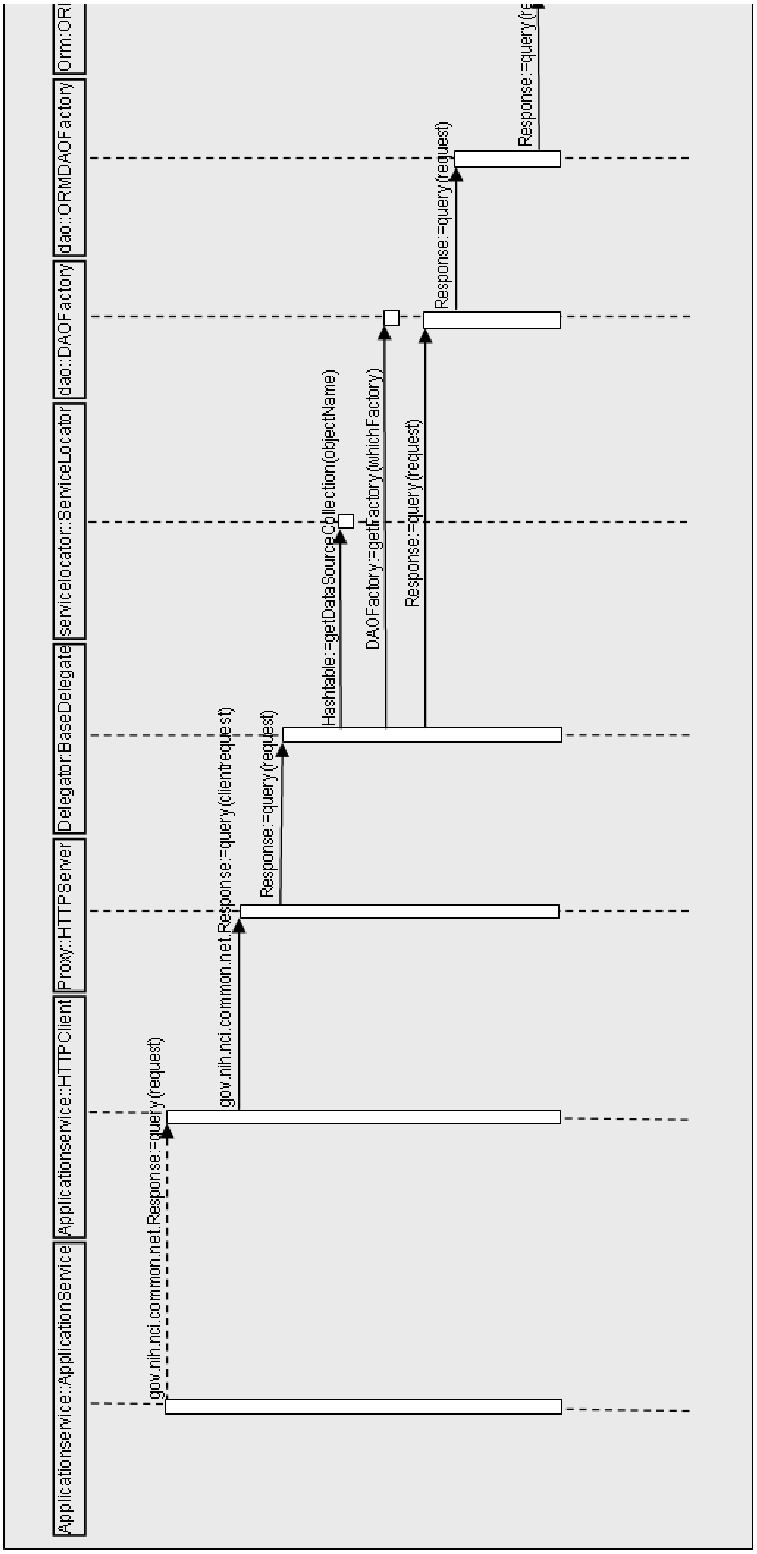

Applicationservice::ApplicationService
Applicationservice::HTTPClient
Proxy::HTTPServer
Delegator:BaseDelegate
servicelocator::ServiceLocator
dao::DAOFactory
dao::ORMDAOFactory
Orm::ORI
gov.nih.nci.common.net.Response::query(request)
gov.nih.nci.common.net.Response::query(clientrequest)
Response::query(request)
Hashtable::getDataSourceCollection(objectName)
DAOFactory::getFactory(whichFactory)
Response::query(request)
Response::query(request)
Response::query(re

Figure A.13 Example of a sequence diagram

APPENDIX B

REFERENCES

Technical Manuals/Articles

1. National Cancer Institute. *caCORE SDK 3.2 Programmer's Guide* http://ncicb.nci.nih.gov/NCICB/infrastructure/cacoresdk#Documentation
2. Java Bean Specification: http://java.sun.com/products/javabeans/docs/spec.html
3. Foundations of Object-Relational Mapping: http://www.chimu.com/publications/objectRelational/
4. Object-Relational Mapping articles and products: http://www.service-architecture.com/object-relational-mapping/
5. Hibernate Reference Documentation: http://www.hibernate.org/5.html
6. Basic O/R Mapping: http://www.hibernate.org/hib_docs/v3/reference/en/html/mapping.html
7. Java Programming: http://java.sun.com/learning/new2java/index.html
8. Jalopy User Manual: http://jalopy.sourceforge.net/existing/manual.html
9. Javadoc tool: http://java.sun.com/j2se/javadoc/
10. JUnit: http://junit.sourceforge.net/
11. Extensible Markup Language: http://www.w3.org/TR/REC-xml/
12. XML Metadata Interchange: http://www.omg.org/technology/documents/formal/xmi.htm
13. Ehcache: http://ehcache.sourceforge.net/documentation/

Scientific Publications

1. Ansher SS and Scharf R (2001). The Cancer Therapy Evaluation Program (CTEP) at the National Cancer Institute: industry collaborations in new agent development. Ann N Y Acad Sci 949:333-40.
2. Boon K, Osorio EC, Greenhut SF, Schaefer CF, Shoemaker J, Polyak K, Morin PJ, Buetow KH, Strausberg RL, De Souza SJ, and Riggins GJ (2002). An anatomy of normal and malignant gene expression. Proc Natl Acad Sci U S A 2002 Jul 15.
3. Buetow KH, Klausner RD, Fine H, Kaplan R, Singer DS, and Strausberg RL (2002). Cancer Molecular Analysis Project: Weaving a rich cancer research tapestry. Cancer Cell 1(4):315-8.
4. Boguski & Schuler (1995). ESTablishing a human transcript map. Nature Genetics 10: 369-71.
5. Clifford R, Edmonson M, Hu Y, Nguyen C, Scherpbier T, and Buetow KH (2000). Expression-based genetic/physical maps of single-nucleotide polymorphisms identified by the Cancer Genome Anatomy Project. Genome Res 10(8):1259-65.
6. Covitz P.A., Hartel F., Schaefer C., De Coronado S., Sahni H., Gustafson S., Buetow K. H. (2003). caCORE: A common infrastructure for cancer informatics. Bioinformatics. *19*: 2404-2412.
7. Dowell RD, Jokerst RM, Day A, Eddy SR, Stein L. The Distributed Annotation System. BMC Bioinformatics 2(1):7.
8. The Gene Ontology Consortium. (2000). Gene ontology: tool for the unification of biology. Nature Genetics 25:25-9.
9. The Gene Ontology Consortium. (2001). Creating the gene ontology resource: design and implementation. Genome Res 11:1425-33.
10. Golbeck J., Fragoso G., Hartel F., Hendler J., Oberthaler J., Parsia B. (2003). The National Cancer Institute's thésaurus and ontology. Journal on Web Semantics. *1*:75-80.
11. Hartel FW and de Coronado S (2002). Information standards within NCI. In: *Cancer Informatics: Essential Technologies for Clinical Trials*. Silva JS, Ball MJ, Chute CG, Douglas JV, Langlotz C, Niland J and Scherlis W, eds. Springer-Verlag.
12. Hartel F.W., Coronado S., Dionne R., Fragoso G. and Golbeck J. (2005). Modeling a description logic vocabulary for cancer research. Journal of Biomedical Informatics, *38*, in press. (http://www.sciencedirect.com/)
13. Pruitt KD, Katz KS, Sicotte H, and Maglott DR (2000). Introducing RefSeq and LocusLink: curated human genome resources at the NCBI. Trends Genet 16(1):44-7.
14. Pruitt KD, and Maglott DR (2001). RefSeq and LocusLink: NCBI gene-centered resources. Nucleic Acids Res 29(1):137-40.
15. Schuler et al. (1996). A gene map of the human genome. Science 274: 540-6.

16. Schuler (1997). Pieces of the puzzle: expressed sequence tags and the catalog of human genes. J Mol Med 75(10):694-8.
17. Strausberg RL (1999). The Cancer Genome Anatomy Project: building a new information and technology platform for cancer research. In: Molecular Pathology of Early Cancer (Srivastava S, Henson DE, Gazdar A, eds. IOS Press, 365-70.
18. Strausberg RL (2001). The Cancer Genome Anatomy Project: new resources for reading the molecular signatures of cancer. J Pathol 195:31-40.
19. Zhang, Schwartz, Wagner, and Miller (2000). A Greedy algorithm for aligning DNA sequences. J Comp Biol 7(1-2):203-14.

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/NCICB/infrastructure/cacore_overview
2. caBIO: http://ncicb.nci.nih.gov/NCICB/infrastructure/cacore_overview/caBIO
3. caDSR: http://ncicb.nci.nih.gov/NCICB/infrastructure/cacore_overview/cadsr
4. EVS: http://ncicb.nci.nih.gov/NCICB/infrastructure/cacore_overview/vocabulary
5. CSM: http://ncicb.nci.nih.gov/NCICB/infrastructure/cacore_overview/csm

Modeling Concepts

1. Enterprise Architect Online Manual: http://www.sparxsystems.com.au/EAUserGuide/index.html
2. OMG Model Driven Architecture (MDA) Guide Version 1.0.1: http://www.omg.org/docs/omg/03-06-01.pdf
3. Object Management Group: http://www.omg.org/

Applications Currently Using caCORE

1. BIOgopher: http://biogopher.nci.nih.gov/BIOgopher/index.jsp
2. BIO Browser: http://www.jonnywray.com/java/index.html
3. caPathway: http://cgap.nci.nih.gov/Pathways

Software Products

1. Hibernate: http://www.hibernate.org/5.html; http://hibernate.org
2. Tomcat: http://jakarta.apache.org/tomcat/
3. Enterprise Architect: http://www.sparxsystems.com.au/
4. Apache WebServices Axis: http://ws.apache.org/axis/
5. MySQL: http://www.mysql.com/
6. Concurrent Versions System (CVS): https://www.cvshome.org/
7. Ant: http://ant.apache.org/
8. JBoss Application Server: http://www.jboss.com/products/jbossas

GLOSSARY

This glossary describes acronyms, objects, tools and other terms referred to in the chapters or appendixes of this guide.

Term	*Definition*
Apache Axis	Open source package that provides SOAP-based web services to users
API	Application Programming Interface
Application Service	This refers to the CSM interface which exposes all the writeable as well as business methods for a particular application
BO	Business Object
C3D	Cancer Centralized Clinical Database
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
CCR	Center of Cancer Research
CDE	Common Data Element
CGAP	Cancer Genome Anatomy Project
CLM	Common Logging Module
CMAP	Cancer Molecular Analysis Project
CS	Classification Scheme
CSI	Classification Scheme Item
CLM	Common Logging Module
CSM	Common Security Module
CTEP	Cancer Therapy Evaluation Program
CVS	Concurrent Versions System
DAO	Data Access Objects
DAS	Distributed Annotation System
DCP	Division of Cancer Prevention

Term	Definition
DDL	Data Definition Language
DEC	Data Element Concept
DL	Description Logic
DOM	Document Object Model
DTD	Document Type Definition
DTS	Distributed Terminology Server
DU	Deployment Unit
EA	Enterprise Architect
EBI	European Bioinformatics Institute
EMF	Eclipse Modeling Framework
EVS	Enterprise Vocabulary Services
FreeMarker	A "template engine"; a generic tool to generate text output (anything from HTML or RTF to auto generated source code) based on templates
GAI	CGAP Genetic Annotation Initiative
GEDP	Gene Expression Data Portal
Hibernate	A high performance object/relational persistence and query service for JavaProvides the ability to develop persistent classes following common object-oriented (OO) design methodologies such as association, inheritance, polymorphism, and composition (http://www.hibernate.org)
HQL	Hibernate Query Language is designed as a "minimal" object-oriented extension to SQL, provides a bridge between the object and relational databases
IDE	Integrated Development Environment
ISO	International Organization for Standardization
JAR	Java Archive
Java Bean	Reusable software components that work with Java
Java Servlet	Server-side Java programs, that web servers can run to generate content in response to client requests
Javadoc	Tool for generating API documentation in HTML format from doc comments in source code (http://java.sun.com/j2se/javadoc/)
JBoss	J2SE application server used as a a presentation layer in caCORE architecture. See also Tomcat.
JDBC	Java Database Connectivity
JDiff	Javadoc doc-let which generates an HTML report of all the packages, classes, constructors, methods, and fields which have been removed, added or changed in any way, including their documentation, when two APIs are compared (http://javadiff.sourceforge.net/)
JET	Java Emitter Templates
JMI	Java Metadata Interface
JSP	Java Server Pages. Web pages with Java embedded in the HTML to incorporate dynamic content in the page
JUnit	A simple framework to write repeatable tests (http://junit.sourceforge.net/)
MDR	Metadata Repository

Term	*Definition*
metadata	Definitional data that provides information about or documentation of other data.
MMHCC	Mouse Models of Human Cancers Consortium
multiplicity	Multiplicity of an association end indicates the number of objects of the class on that end that may be associated with a single object of the class on the other end
MVC	Model-View-Controller, a design pattern
navigability	Navigability defines the visibility of an object to its associated source/target object at the other end of an association. Navigability is the same as directionality.
NCI	National Cancer Institute
NCICB	National Cancer Institute Center for Bioinformatics
NSC	Nomenclature Standards Committee
OMG	Object Management Group
OR	Object Relation
ORM	Object Relational Mapping
PCDATA	Parsed Character DATA
persistence layer	Data storage layer, usually in a relational database system
RDBMS	Relational Database Management System
RUP	Rational Unified Process
SOAP	Simple Object Access Protocol. A lightweight XML-based protocol for the exchange of information in a decentralized, distributed environment
SPORE	Specialized Programs of Research
SQL	Structured Query Language
Tagged value	A UML construct that represents a name-value pair; can be attached to anything in a UML model. Often used by UML modeling tools to store tool-specific information
Tomcat	J2SE application server used as a a presentation layer in caCORE architecture. See also JBoss.
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locators
war	Web archive file
Writeable API	Methods exposed by the CSM to create, update and delete a domain object. These methods are generated using the code generation component.
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

Term	Definition
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
XP	Extreme Programming

INDEX