caCORE
Software Development Kit 3.2

Programmer's Guide

NATIONAL® CANCER INSTITUTE

Center for Bioinformatics

Revised March 28, 2007

CREDITS AND RESOURCES

caCORE SDK 3.2 Contributors			
Development		***Programmer's Guide***	***Program Management***
Anwar Ahmad[3]	Thai Le[1]	Anwar Ahmad[3]	Tara Akhavan[1]
Prerna Aggarwal[3]	Art Lian[4]	Steve Alred[3]	Steve Alred[3]
Gavin Brennan[7]	Ying Long[1]	Michael Connolly[1]	Bill Britton[7]
Vanessa Caldwell[7]	Wei Lu[7]	Dan Dumitru[4]	Peter Covitz[2]
Ben Chapman[7]	Christophe Ludet[3]	Wendy Erickson-Hirons[6]	Charles Griffin[4]
Ram Chilukuri[8]	Harsh Marwaha[3]	Jill Hadfield [2]	Frank Hartel[2]
Michael Connolly[1]	Doug Mason[1]	Shan Jiang[1]	George Komatsoulis[2]
Eric Copen[4]	Kunal Modi[4]	George Komatsoulis[2]	Sichen Liu[2]
Kim Diercksen[7]	Shaziya Muhsin[1]	Christophe Ludet[3]	Krishnakant Shanbhag[2]
Dan Dumitru[4]	Kim Ong[5]	Doug Mason[1]	Denise Warzel[2]
Craig Fee[7]	Satish Patel[4]	Kunal Modi[4]	
Gilberto Fragoso[2]	Ralph Rutherford[7]	Shaziya Muhsin[1]	
Steven Hunter[4]	Nick Schroedl[1]	Satish Patel[4]	
Jane Jiang[3]	Jiang Scott[1]	Tracy Safran[1]	
Shan Jiang[1]	Claire Wolfe[7]	Nick Schroedl[1]	
Sriram Kalyanasundaram[7]	Ye Wu[1]	Krishnakant Shanbhag[2]	
Doug Kanoza[7]	Rob Wynne[9]	Denise Warzel [2]	
Alan Klink[7]	Bob Wysong[7]	Ye Wu[1]	
Vinay Kumar[4]	Nafis Zebarjadi[1]	Nafis Zebarjadi[1]	
Norval Johnson[7]	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] Semantic Bits
[9]Management System Designers, Inc.			

The following NCICB listserv facilities are pertinent to this SDK Programmer's 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
cacoresdk	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/	Common Logging Module (CLM) CLM is a powerful set of auditing and logging tools implemented in a flexible and comprehensive solution.

TABLE OF CONTENTS

CHAPTER

1

USING THE SOFTWARE DEVELOPMENT KIT PROGRAMMER'S GUIDE

This chapter introduces you to the *caCORE Software Development Kit 3.2 Programmer's Guide* and suggests ways you can maximize its use.

Topics in this chapter include:

Introduction

The *caCORE Software Development Kit 3.2 Programmer's Guide* (SDK Guide) is the companion documentation to the caCORE (cancer Common Ontologic Representation Environment [http://ncicb-dev.nci.nih.gov/infrastructure/cacore_overview]) Software Development Kit (SDK). The SDK aids intermediate level Java programmers with some life science background who are interested in using or extending the capabilities of caCORE. The caCORE SDK is a set of development resources that allows you to create, compile, and run caCORE-like software.

This guide includes information and instructions for using the SDK. When the processes outlined in this guide are followed, a Java programmer of moderate skill, starting with a Unified Modeling Language (UML) model, should be able to create and install a caBIG 'Silver' compliant caCORE-like system. (For more information, see *caBIG* on page 7.)

Before continuing, note three points about this *caCORE Software Development Kit Programmer's Guide*:

- Installation and basic test instructions for the SDK are available in an independent document, the *caCORE Software Development Kit 3.2 Installation and*

Basic Test Guide, downloadable at http://ncicb.nci.nih.gov/NCICB/infrastructure/cacoresdk#Documentation.

- This document contains no information on the topic of Java programming or Object-Oriented Programming in the abstract.
- Generally, caCORE-like systems persist their data in relational database management systems (RDBMS), although other data storage and retrieval facilities are also supported. Although it is possible to create an RDBMS schema that mirrors the Object Model of the caCORE-like system, this is not necessarily the most efficient practice. This guide does not cover optimization of relational databases.

Users wanting more information about these topics are referred to the documentation noted or to the substantial literature on these subjects.

Recommended Reading

Following is a list of recommended reading materials and resources that can be useful for familiarizing oneself with concepts contained within this guide.

- Java Programming
- Enterprise Architect Online Manual
- OMG Model Driven Architecture (MDA) Guide Version 1.0.1
- Hibernate

Uniform Resource Locators (URLs) are also included throughout the document to provide more detail on a subject or product.

Organization of this Guide

The *caCORE Software Development Kit 3.2 Programmer's Guide* contains the following chapters:

Chapter 1 Using the Software Development Kit Programmer's Guide — This chapter provides information about using the SDK Guide.

Chapter 2 NCICB caCORE Infrastructure — This chapter provides an overview of the caCORE infrastructure including a discussion on Model Driven Architecture, *n*-tier architecture with open APIs, use of controlled vocabularies, and registered metadata.

Chapter 3 caCORE Software Development Kit Architecture — This chapter provides an overview of the caCORE SDK and its architecture including its process flow from an architectural perspective, components, and software requirements.

Chapter 4 caCORE Software Development Kit Workflow — This chapter summarizes the process workflow for using the caCORE SDK to generate a caCORE-like, semantically-interoperable system.

Chapter 5 Creating the UML Models — This chapter contains all of the necessary procedures to create UML models for the caCORE-like system.

Chapter 6 Performing Semantic Integration — This chapter describes all of the necessary procedures to configure semantic integration in the SDK.

Chapter 7 Registering Metadata — This chapter describes the process of registering and mapping metadata using the UML Loader.

Chapter 8 Generating the caCORE-Like System — This chapter describes the process for generating the code that produces a caCORE-like system, executing tests on the system, and creating manual ORMs.

Chapter 9 Integrating CSM with the SDK —This chapter describes functionality that enables security, session management, and writable APIs for your application. This chapter also demonstrates how to combine the writable APIs with the SDK-generated domain model.

Appendix A Unified Modeling Language — This appendix is designed to familiarize the reader who has not worked with UML with its background and notation for the models described in this guide.

Appendix B Software Configuration Management — This appendix describes the best practices of Software Configuration Management (SCM) used by the caCORE development team.

Appendix C Performance Tuning —This appendix describes a specific Hibernate issue relevant to using CSM with the SDK.

Appendix D References —This appendix provides a list of references used to produce this guide or referred to within the text.

CHAPTER 2

NCICB caCORE Infrastructure

This chapter provides an overview of the caCORE infrastructure.

Topics in this chapter include:

caCORE Infrastructure Overview

NCICB provides biomedical informatics support and integration capabilities to the cancer research community. NCICB has created a core infrastructure called caCORE, a data management framework designed for researchers who need to be able to navigate through a large number of data sources. caCORE is NCICB's platform for data management and semantic integration, built using formal techniques from the software engineering and computer science communities.

caCORE Development Principles

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

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.

The use of MDA and *n*-tier architecture, both standard software engineering practices, allows for easy access of 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.

EVS and caDSR are two of the main components of caCORE, created and deployed by NCICB. Cancer Bioinformatics Infrastructure Objects (caBIO) and the Common Security Model (CSM) are also main components of caCORE. All components, designed using these same four development principles, are described as follows:

- **Cancer Bioinformatics Infrastructure Objects (caBIO)** — A set of JavaBeans with open APIs that can be used to directly access bioinformatics data (http://ncicb.nci.nih.gov/NCICB/infrastructure/cacore_overview/caBIO). 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.
- **Cancer Data Standards Repository (caDSR)** — A metadata registry, based on the ISO/IEC 11179 standard, used to register the descriptive information needed to render cancer research data reusable and interoperable (http://ncicb.nci.nih.gov/NCICB/infrastructure/cacore_overview/cadsr). 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.
- **Enterprise Vocabulary Services (EVS)** — Controlled vocabulary resources that support the life sciences domain, implemented in a description logics framework (http://ncicb.nci.nih.gov/NCICB/infrastructure/cacore_overview/vocabulary). EVS vocabularies provide the semantic 'raw material' from which data elements, classes, and objects are constructed.
- **Common Security Model (CSM)** — A flexible solution for application security and access control (http://ncicb.nci.nih.gov/NCICB/infrastructure/cacore_overview/csm). CSM provides 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

caBIG

The NCICB, in cooperation with various cancer centers and other research institutions has recently launched the cancer Biomedical Informatics Grid (caBIG) (http://cabig.nci.nih.gov/) that is designed to create a large data system using Grid technology. Because of the federated nature of data grids, it was deemed essential that semantic interoperability be integrated into caBIG, with guidelines devised for various levels of compliance ranging from Legacy (no semantic interoperability), through Bronze, Silver and Gold (fully Grid compatible). See caBIG Compatibility Guidelines (http://cabig.nci.nih.gov/guidelines_documentation) for more information.

caBIO as an Example System

To understand the mechanics of creating a software system using the caCORE SDK, it is useful to study an existing system built using its principles and tools. caBIO, an integral part of the NCICB caCORE infrastructure, provides an excellent example of how all of the various parts of the caCORE infrastructure and SDK interact in a caCORE-compatible software system.

Model Driven Architecture

Model Driven Architecture (MDA) is a software development practice that uses a structured modeling language to describe the requirements, objects, and interactions of a data system prior to its construction. When coupled with a design process such as the Rational Unified Process (RUP) and Extreme Programming (XP), it can greatly assist in the production of quality software delivered in a timely fashion. At NCICB, caCORE is modeled using the UML, coupled with a fusion of the RUP and XP.

For more information about UML, see *Appendix A*.

n-tier Architecture and Consistent APIs

The caBIO system uses an architecture that separates the application into a series of tiers (*Figure 2.1*). A typical client-server system is a two-tier system (the client and the server that returns the data). While simple, it ties the client very tightly to the details of the implementation model. To isolate the client from the implementation details, a data system can be built with one or more layers of ‘middle ware’, software whose purpose is to act as a bridge between the server and the client. If changes are made to the server, the middle ware is modified so that the client sees a consistent interface (API).

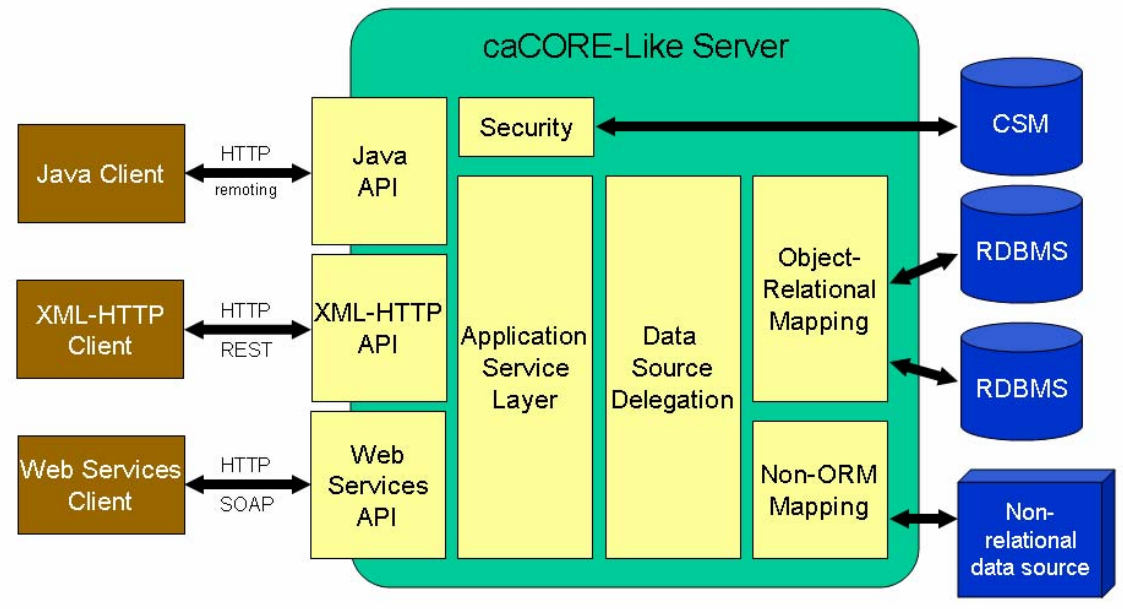


Figure 2.1 caCORE architecture

Metadata and Controlled Vocabularies

Metadata

The use of controlled vocabularies and registration of data in caCORE through EVS and caDSR helps resolve the issue of identifying in an unambiguous manner the meaning of each object and attribute in an API. Generally, metadata is 'data about data'. That is, it is a *definition* of an attribute rather than its *value* and this is the case for caCORE.

Two examples:

- The *value* of the attribute 'zipCode', might be '20852' while its metadata (*definition*) is 'a 5 or 9 digit number used by the United States Postal Service to divide geographical regions into delivery zones'. By registering metadata (using terms in an electronically-accessible controlled vocabulary) in a repository, caCORE provides a means to select appropriate information resources and to aggregate information from multiple sources.
- An object model that describes an Agent, in this setting, is a chemotherapeutic agent. An excerpt from the caBIO model describing the Agent class is shown in *Figure 2.2*.

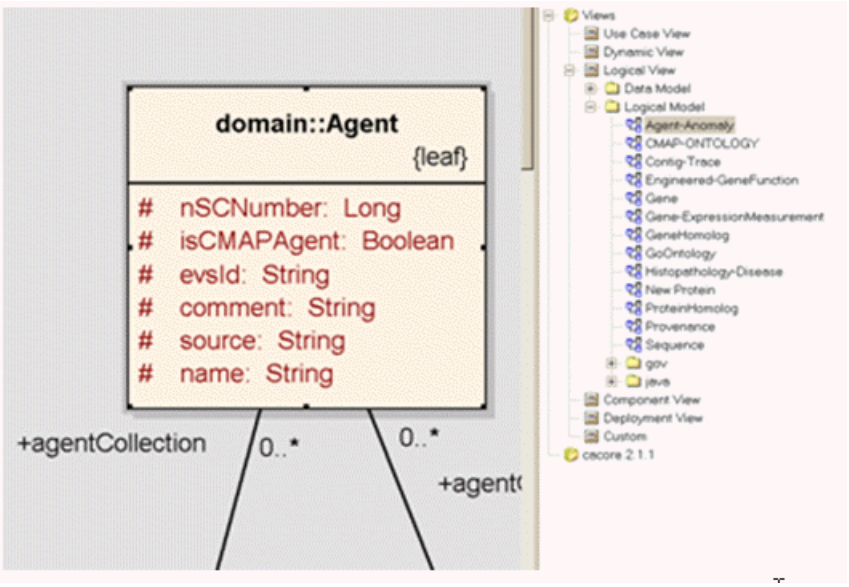


Figure 2.2 Agent Class from caBIO class diagram

The Agent class has several attributes including ‘name’, ‘nSCNumber’, etc. *Table 2.1* displays a specific instance of this class with its attributes and values, and demonstrates two possible sets of metadata: one from the perspective of the National Cancer Institute (NCI) and the other from the perspective of the Central Intelligence Agency (CIA).

Class or Attribute Name	***Value***	***NCI Metadata***	***CIA Metadata***
Agent		A chemical compound administered to a human being to treat an existing disease or condition, or prevent the onset of a disease or condition	A sworn intelligence agent; a spy
nSCNumber	007	Identifier given to a chemical compound by the US Food and Drug Administration (FDA) Nomenclature Standards Committee (NSC)	Identifier given to an intelligence agent by the National Security Council (NSC)
Name	Taxol	Name of a chemical compound given by the NCI Cancer Therapeutics Evaluation Program (CTEP)	Code name given to intelligence agents by the Central Intelligence Agency (CIA)

Table 2.1 Metadata examples

As can be seen in the table, the same values are reasonable whether Agent is described as a chemotherapeutic agent (NCI) or a spy (CIA); the metadata, on the other hand, allows us to distinguish between two sets of completely valid information.

Controlled Vocabularies

As noted, for maximum interoperability, the metadata should be derived from terms with unambiguous meanings. This prevents misunderstanding based on differences in the

use of terms or phrases between different specialties or geographic regions. In a data system context, this can be accomplished by having all parties use the same dictionary of terms. When these dictionaries reside in a central location and are managed according to defined rules, they are known as 'controlled vocabularies'. These vocabularies come from a variety of sources and can cover a wide range of topics. Furthermore, they can be organized into ontologies; these hierarchical structures exhibiting well defined relationships make it easy to compute certain relationships between data.

Registration of Metadata in the caDSR

Metadata Repositories

For metadata to be useful, it must be accessible to applications at runtime. For this reason, the NCICB developed the caDSR to store metadata, based on the ISO/IEC 11179 metamodel. This model describes a wide range of characteristics of data elements including definitions, permissible values, data types, units of measure, minimum and maximum lengths, etc. *Figure 2.3* shows an example of a Common Data Element or CDE (in this case, the attribute 'nSCNumber' from *Table 2.1* as it is represented with a non-enumerated, generic value domain of datatype "String" in the caDSR).

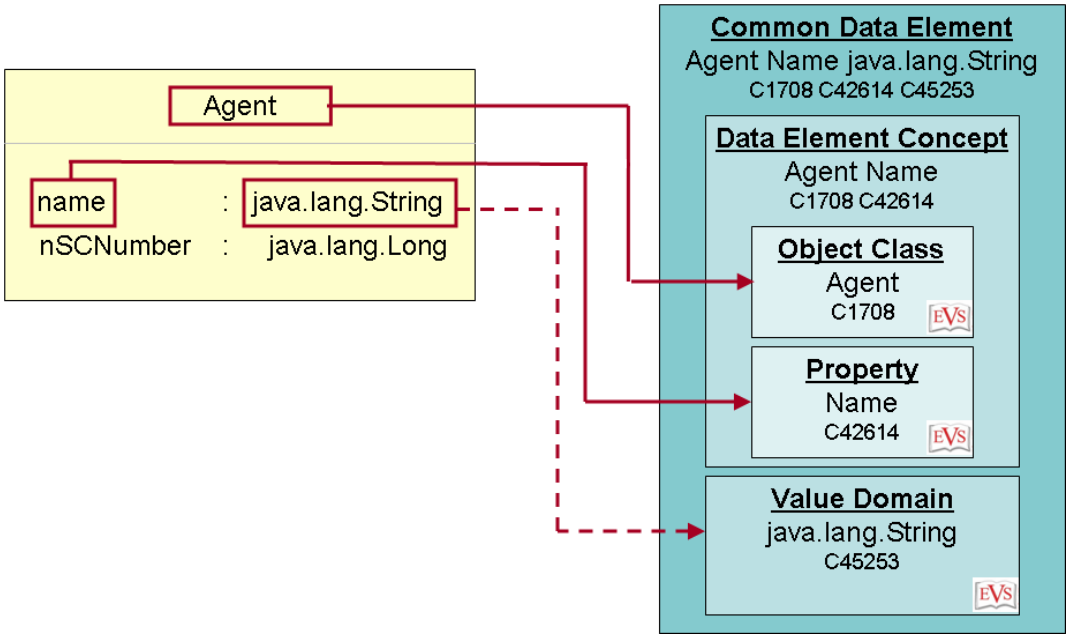


Figure 2.3 A CDE as represented in the caDSR (ISO/IEC 11179 model)

In the ISO/IEC 11179 model, a **Data Element** consists of two parts as shown in *Figure 2.4*:

1. a **Data Element Concept** that provides the conceptual definition of the data element.
2. a **Value Domain** that describes specific acceptable values for that data element. Value domains can be either 1) enumerated with an explicit list of permissible values, or 2) non-enumerated, restricting the values to a description, specification or rule. Attributes of the Value Domain include data characteristics such as the data type, representation class, and unit of measure.

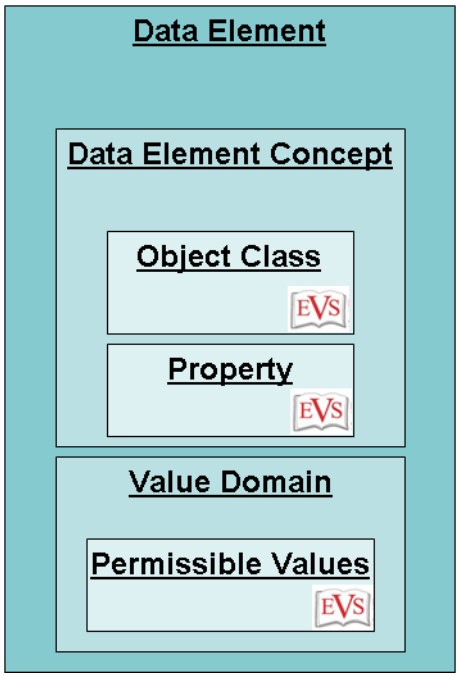


Figure 2.4 Representation of a Data Element

The parts of the caDSR implementation of the ISO/IEC 11179 model, **Object Class**, **Property**, and **Value Domain** are described using controlled vocabulary terms maintained by the EVS. Thus the caDSR provides a link between a data element (such as an attribute in an object model) and definitions in a controlled vocabulary.

A Data Element Concept is represented by a combination of at least two EVS concepts—a primary concept for Object Class and Property, each of which may have qualifiers that are also EVS terms. Similarly, the Value Domain has at least a representation that reflects the operational characteristics of the form in which the value is being recorded. The representation could be ‘Currency’, ‘Number’, ‘Code’, etc. It is intended to convey information in addition to the datatype. If the value domain is enumerated, the list of values and the meaning associated with each may be based on one or more concepts from EVS.

UML Models and the caDSR

The previous section describes metadata in the caDSR in the abstract. For most users of this SDK, the more relevant information is the means by which attributes of an object model (specifically a UML model) are stored in the caDSR. *Figure 2.5* shows the mapping of an attribute from a UML model into the components of the caDSR described above.

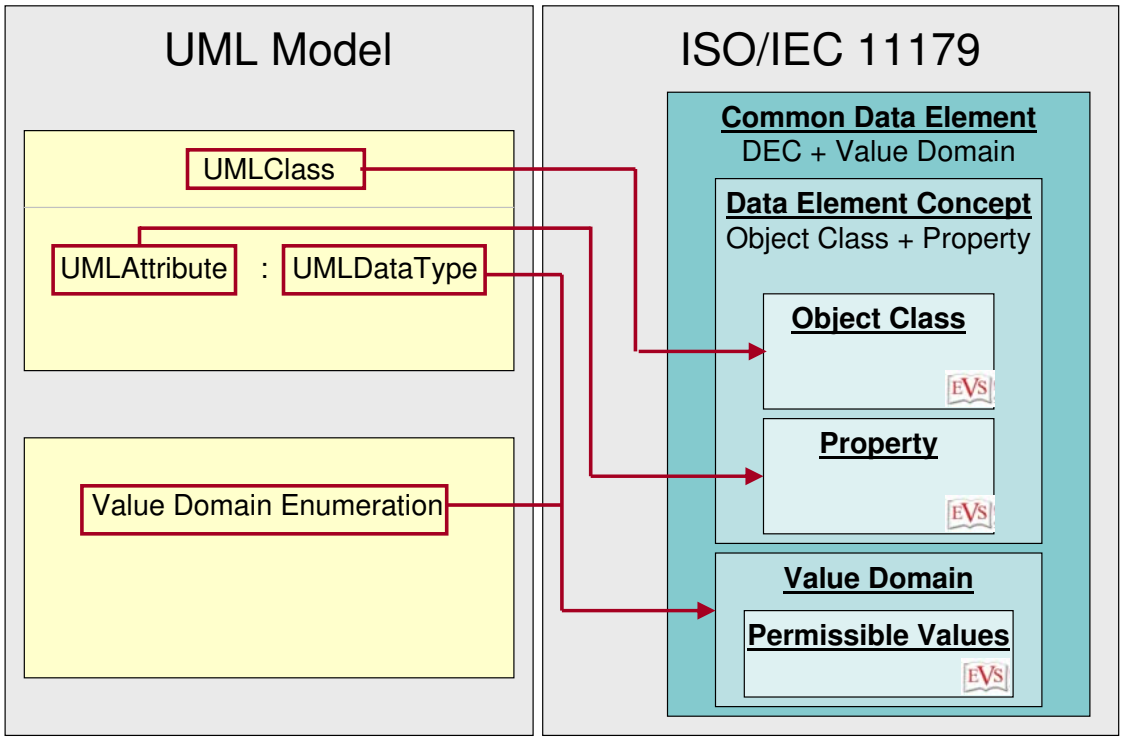


Figure 2.5 Mapping a UML model into an ISO/IEC 11179 compliant caDSR CDE

In the caDSR implementation, a data element corresponds to a semantically-enhanced UML attribute. A Common Data Element's semantics are based on a Data Element Concept (DEC) and a Value Domain as shown in *Figure 2.5*. A DEC is composed of the UML class concatenated with one of its attributes. The UML class is mapped to the caDSR 'Object Class' and the UML attribute is mapped to the caDSR 'Property'. The caDSR Object Class and properties are concepts derived from EVS. Combined with the Value Domain (if enumerated as described in *Registration of Metadata in the caDSR* on page 10), this gives an unambiguous mapping of an attribute in a UML model to terms in a controlled vocabulary. This mapping or transformation of UML Models into caDSR metadata is performed by the UML Loader and is described in more detail in *Chapter 8*. The complete process for mapping UML model elements to controlled vocabulary concepts is described in *Chapter 7*.

Examples of caCORE-Like Systems

The following applications are caCORE-like and have been built using the caCORE principles described here.

- BIOcrawler (http://ncicb.nci.nih.gov/download/index.jsp)
- BIOgopher (http://biogopher.nci.nih.gov/BIOgopher/index.jsp)
- BIO Browser (http://www.jonnywray.com/java/index.html)
- caArray (http://caarray.nci.nih.gov)
- Cancer Molecular Analysis Project (CMAP) (http://cmap.nci.nih.gov)
- Cancer Models Database (http://cancermodels.nci.nih.gov)

- Cancer Centralized Clinical Database (C3D) (http://ncicbsupport.nci.nih.gov/sw/content/C3D.html)

Finalizing the Development Process

To summarize the SDK development process: 1) object models and data models are created and exported to XMI; 2) DDL scripts are generated from the data models; 3) the models are annotated with immutable concept codes from EVS; 4) the metadata is registered in caDSR, thereby enabling semantic interoperability; 5) the Java source code is generated for a data access API, using the XMI file generated in the SDK process.

For a complete discussion of the SDK process workflow, see *Chapter 4*.

Software Configuration Management

Appendix B introduces basic Software Configuration Management (SCM) concepts, describing a set of management processes that we recommend you have in place if you plan to distribute your caCORE-like software or deploy it outside of your site. The caCORE SDK development team has followed this defined set of SCM processes centered around caCORE open source tools. Refer to *Appendix B* for additional information and links to resources about SCM to help you manage your own software environment.

CHAPTER

3

caCORE Software Development Kit Architecture

This chapter provides an overview of the caCORE SDK architecture.

Topics in this chapter include:

caCORE 3.2 SDK Minimal System Requirements

Minimal system requirements for the caCORE 3.2 SDK consist of:

- Internet connection
- Tested platforms

The caCORE 3.2 SDK has been tested on the platforms shown in Table *3.1*.

	Linux Server	***Solaris***	***Windows***
Model	HP Proliant ML 330	Sunfire 480R	Dell GX 270
CPU	1 x Intel® Xeon™ Processor 2.80GHz	2 x 1050MHz	1 x Intel® Pentium™ Processor 2.80GHz
Memory	4 GB	4 GB	1 GB

Table 3.1 Platform Testing Environment

	Linux Server	Solaris	Windows
Local Disk	System 2 x 36GB (RAID 1) Data = 2 x 146 (RAID 1)	System 2 x 72GB	System 1 x 36GB
	Red Hat Linux ES 3 (RPM 2.4.21-20.0.1)	Solaris 8	Windows XP/2000 Professional

Table 3.1 Platform Testing Environment (Continued)

Note: The Semantic Integration Workbench (SIW) has been tested on Windows 2000, Linux, Mac OSX, and Solaris 8.

caCORE SDK Package

The caCORE SDK includes the following components:

- Sample UML object/data model to use with the development kit
  - `cacoretoolkit.eap`
- XML Metadata Interchange (XMI) Version of the sample model
  - `cabioExampleDomainModel.xmi`
- Framework packages
  - `gov.nih.nci.system`
  - `gov.nih.nci.common`
  - `org.hibernate`
- Configuration files to enable you to customize your installation to meet your specific database, server, and other network needs. Users may need to modify other configuration and property files, such as os.[linux|unix|windows].xml.
  - `deploy.properties`
  - `download.properties`
- Ant `buildfile.xml file`
- Code generator package
  - `gov.nih.nci.codegen.core`
  - `gov.nih.nci.codegen.framework`
- Java JET templates for generating caCORE-like APIs
- Demo package with sample Java clients and examples of how to leverage the code generation framework (for advanced users)

caCORE SDK Software and Technology Requirements

The required and optional software to use the caCORE SDK are listed in Table *3.2* and Table *3.3*. The software name, version, description, URL, and whether it is included in the distribution are indicated. The included (**Incl.**) column indicates (with a **Yes**) if the software is packaged with the SDK. **No** indicates that you must supply the software.

Required software not packaged with the caCORE SDK:

- Java Software Development Kit (SDK); downloaded from Sun Microsystems
- to run the Semantic Integration Workbench, Java Web Start must be installed on your machine. It can be downloaded from Sun Microsystems
- a UML modeling tool (Enterprise Architecture was used to create models and screen shots for this guide)

Hyperlinks are included in Table *3.2* for your reference to appropriate sources.

Software Name	Version	Description	URL	Incl
Java 2 Platform Standard Edition (J2SE) 5.0 Development Kit (JDK 5.0)	jdk1.4.2	The J2SE Software Development Kit (SDK) supports creating J2SE applications	http://java.sun.com/j2se/1.4.2/download.html	No
UML 1.3 Modeling Tool that produces XMI 1.1 output format	EA 4.50.744 or higher	We recommend using Enterprise Architect (EA)	http://www.sparxsystems.com.au	No
ant.jar	1.6.5	Apache Ant is a Java-based build tool	http://ant.apache.org/bindownload.cgi	Yes
activation.jar		The classes that make up the JavaBeans Activation Framework (JAF) standard extension are contained in the included Java Archive (JAR) file, "activation.jar"	http://java.sun.com/products/javabeans/glasgow/jaf.html	Yes
antlr-2.7.6.jar	2.7.6	Query parser used by Hibernate 3	http://www.antlr.org/download.html	Yes
axis-ant.jar	1.4	Ant tasks for building axis.	http://ws.apache.org/axis/releases.html	Yes
ant-testutil.jar	1.6.5	Part of the Ant framework. Needed to run the JUnit task from within Ant.	Not available as a standalone binary download. You would have to download the ANT sources and then build it.	Yes
asm.jar	1.5.3	ASM is a Java bytecode manipulation framework. It can be used to dynamically generate stub classes or other proxy classes, directly in binary form, or to dynamically modify classes at load time, i.e., just before they are loaded into the Java Virtual Machine.	http://asm.objectweb.org/index.html	Yes
asm-attrs.jar	1.5.3	Part of the ASM bytecode manipulation framework.		Yes
axis.jar	1.4	Apache Axis is an implementation of the SOAP (Simple Object Access Protocol)	http://ws.apache.org/axis/releases.html	Yes
castor-1.0.2.jar	1.0.2		http://www.castor.org/download.html	Yes

Table 3.2 Required Software and Technology for the Development Kit

Software Name	Version	Description	URL	Incl
cglib-2.1.3.jar	2.1.3	Dynamic Java byte code generator	http://sourceforge.net/project/showfiles.php?group_id=56933	Yes
codegen.jar		Classes required for Java Emitter Template (JET) compilation.	http://www.eclipse.org/	Yes
commons-collections-3.2.jar	3.2	Apache Jakarta Commons utilities	http://apache.bestwebcover.com/java-repository/commons-collections/jars/	Yes
commons-dbcp-1.2.1.jar	1.2.1	The Jakarta Commons DBCP Component provides database connection pooling.	http://archive.apache.org/dist/java-repository/commons-dbcp/jars/?C=S;O=A	Yes
commons-discovery-0.2.jar	0.2	Apache Jakarta Commons discovery utilities	http://jakarta.apache.org/commons/discovery/	Yes
commons-lang-2.1.jar	2.1	Provides a helper utilities for the java.lang API.	http://linux.cs.lewisu.edu/apache/java-repository/commons-lang/jars/?C=N;O=D	Yes
commons-logging-1.1.jar	1.1	Provides a helper utilities logging.	http://public.planetmirror.com/pub/maven/commons-logging/jars/	Yes
commons-logging.jar		Apache Jakarta Commons logging utilities	http://jakarta.apache.org/commons/logging/	Yes
commons-pool-1.3.jar	1.3	The Jakarta Commons Pool Component provides a generic object pooling API.	http://apache.intissite.com/java-repository/commons-pool/jars/	Yes
classes12.jar		Oracle JDBC drivers. Classes for use with JDK 1.2 and JDK 1.3.	JDBC driver classes, except classes for NLS support in Oracle Object and Collection types.	Yes
clm.jar		Common Logging Module (CLM), which provides a separate service under caCORE for audit and logging capabilities.		Yes
csmapi.jar		Common Security Module (CSM) APIs.		Yes
dtsrpcclient.jar		Server API extensions to EVS, which provide the capability for users to retrieve vocabulary and edit history data from the DTS database.		Yes
datafile.jar	1.3.2	Java data file read/write utility that provides a convenient set of interfaces for reading and writing data to and from files in widely accepted format such as comma separated values (CSV), fixed width, tab separated, and others.	http://datafile.sourceforge.net/	Yes

Table 3.2 Required Software and Technology for the Development Kit (Continued)

Software Name	Version	Description	URL	Incl
db2java.jar		Contains classes to support connections to DB2 databases.	http://www-306.ibm.com/software/data/db2/udb/	Yes
dom4j-1.6.1.jar		Contains classes that allow read, write, navigate, create and modify capability to XML documents.	http://public.planetmirror.com/pub/maven/dom4j/jars/	Yes
ehcache-1.2.2.jar	1.2.2	Ehcache is a pure Java, in-process cache.	http://smokeping.planetmirror.com/pub/maven/ehcache/jars/	Yes
freemarker.jar	2.3	FreeMarker is a "template engine"; a generic tool to generate text output (anything from HTML or RTF to auto generated source code) based on templates.	http://freemarker.sourceforge.net/freemarkerdownload.html	Yes
hibernate3.jar	3.1.3	Hibernate is used for the server-side object-relational mapping (ORM).	http://www.hibernate.org	Yes
	3.0.5	Hibernate jar file used by CSM; this should replace the 3.1.3 jar ONLY when security is enabled. For more information, see *Appendix C*.		
jakarta-oro-2.0.8.jar	2.0.8	The Jakarta-ORO Java classes are a set of text-processing Java classes that provide Perl5 compatible regular expressions, AWK-like regular expressions, glob expressions, and utility classes for performing substitutions, splits, filtering filenames, etc.	http://jakarta.apache.org/site/bin-index.cgi	Yes
jalopy-1.0b11.jar	1.0b11	Source code formatter.	http://public.planetmirror.com/pub/maven/jalopy/jars/	Yes
jalopy-ant-0.6.2.jar	0.62	Ant task for building jalopy.	http://public.planetmirror.com/pub/maven/jalopy/jars/	Yes
jaxen-core.jar		The jaxen project is a Java XPath Engine. jaxen is a universal object model walker, capable of evaluating XPath expressions across multiple models.	http://jaxen.org/releases.html	Yes
jaxen-jdom.jar		The jaxen project is a Java XPath Engine. jaxen is a universal object model walker, capable of evaluating XPath expressions across multiple models.	http://jaxen.org/releases.html	Yes
jaxrpc.jar	1.1	Java API for XML-based RPC.		Yes
jboss-client.jar		Contains the JBoss EJB container proxy and stub client classes.		Yes

Table 3.2 Required Software and Technology for the Development Kit (Continued)

Software Name	Version	Description	URL	Incl
jdom.jar	1.0	Java-based solution for accessing, manipulating, and outputting XML data from Java code.	http://www.jdom.org/downloads/index.html	Yes
jdtcore.jar		Eclipse Tomcat Plugin.		Yes
jetc-task.jar		Ant task for translating JET templates outside of Eclipse.	http://download.eclipse.org/tools/emf/scripts/docs.php?doc=tutorials/jet2/jet_tutorial2.html	Yes
jmi.jar		JMI is a standards-based, platform independent, vendor-neutral specification for modeling, creating, storing, accessing, querying, and interchanging metadata using UML, XML, and Java.	http://mdr.netbeans.org/download/daily.html	Yes
jmiutils.jar			http://mdr.netbeans.org/download/daily.html	Yes
jta.jar		JTA specifies standard Java interfaces between a transaction manager and the parties involved in a distributed transaction system.	http://java.sun.com/products/jta/	Yes
junit-3.8.1.jar		JUnit is a regression testing framework that is used by the developer who implements unit tests in Java.	http://www.junit.org/index.htm	Yes
log4j-1.2.13.jar	1.2.13	Log4j is an open source tool developed for putting log statements into your application. With log4j you can enable logging at runtime without modifying the application binary.	http://logging.apache.org/log4j/docs/download.html	Yes
log4j.properties		Log4J.		Yes
mail.jar	1.2	JavaMail API.		Yes
mdrant.jar		Ant tasks for building MDR.		Yes
mdrapi.jar		MDR implements the OMG's MOF (Meta Object Facility) standard based metadata repository and integrates it into the NetBeans Tools Platform. It contains an implementation of an MOF repository including a persistent storage mechanism for storing the metadata. The interface of the MOF repository is based on (and fully compliant with) JMI (Java Metadata Interface - JSR-40).	http://mdr.netbeans.org/download/daily.html	Yes

Table 3.2 Required Software and Technology for the Development Kit (Continued)

Software Name	Version	Description	URL	Incl
mof.jar			http://mdr.netbeans.org/download/daily.html	Yes
mysql-connector-java-3.1.13-bin.jar		Contains classes that support a JDBC connection		Yes
nbmdr.jar		http://jaxen.codehaus.org/	http://mdr.netbeans.org/download/daily.html	Yes
openide-util.jar		Contains low level basic support classes on which MDR depends.	http://mdr.netbeans.org/download/daily.html	Yes
osgi.jar			http://www.osgi.org/osgi_technology/download_specs.asp?section=2	Yes
ojdbc14.jar		Oracle JDBC Drivers. Classes for use with JDK 1.4. It contains the JDBC driver classes, except classes for NLS support in Oracle Object and Collection types.		Yes
p6spy.jar		Open source framework for applications that intercept and optionally modify data-base statements.	http://www.p6spy.com	Yes
resources.jar				Yes
runtime.jar				Yes
saaj.jar	1.2			Yes
saxpath.jar	1.0	SAXPath is an event-based API for XPath parsers that parse XPath expressions.		Yes
serializer.jar		Contains the serializer classes of Xalan-Java2.		Yes
servlet.jar				Yes
uml-1.3.jar				Yes
wsdl4j-1.5.1.jar		Web Services Description Lan-guage support for Java.		Yes
xercesImpl.jar	2.7.1	Xerces Java XML parser.	http://xml.apache.org/xerces-j/	Yes
xml-apis.jar	2.0.2	XSLT processor for transforming XML documents into HTML, text, or other XML document types.	http://xml.apache.org/xalan-j/	Yes
xmlrpc.jar		Apache XML-RPC is a Java implementation of XML-RPC, a popular protocol that uses XML over HTTP to implement remote procedure calls.	http://www.apache.org/	Yes

Table 3.2 Required Software and Technology for the Development Kit (Continued)

Software Name	Version	Description	URL	Incl
xalan.jar		An XSLT (Extensible Stylesheet Language Transformation) processor for transforming XML documents into HTML, text, or other XML document types.	http://xalan.apache	

Table 3.2 Required Software and Technology for the Development Kit (Continued)

Additional Software

The caCORE SDK requires a Java 2 container in which the server component can run. Several different open source and commercial applications are available; the software in Table *3.3* have been tested and are known to work with the SDK.

Software Name	Version	URL	Notes
Apache Tomcat	4.1.x	http://tomcat.apache.org	The SDK can install Tomcat as part of the build process.
JBoss Application Server	4.0.2	http://www.jboss.org	See the JBoss documentation for instructions on installation and configuration.

Table 3.3 Additional software and technology for the Development Kit

The caCORE SDK typically also requires a relational database management system (RDBMS) for data persistence. The SDK provides support for the RDBMS software in Table *3.4*.

Software Name	Version	URL	Notes
MySQL	4.1.x	http://dev.mysql.com/downloads/mysql/4.1.html	The SDK can install MySQL as part of the build process.
Oracle 9i	9i	http://www.oracle.com/technology/software/products/oracle9i/index.html	Oracle is commercial software. Release 9i is currently the only release supported by the SDK.
IBM DB2	8.2	http://www-306.ibm.com/software/data/db2/	DB2 is commercial software.

Table 3.4 RDBMS software

Note: Drivers for MySQL, Oracle 9i and DB2 are included with the SDK. If you are using a different version of Oracle or DB2, you must obtain the appropriate drivers. JDBC drivers can be downloaded from the Sun Developer Network at http://developers.sun.com/product/jdbc/drivers/index.html, or from the individual vendors' sites (for example, the Oracle 8i driver `classes12.zip` can be downloaded from http://www.oracle.com/technology/software/tech/java/sqlj_jdbc/index.html). These drivers should be placed in the `{project_home}/lib` directory and the appropriate directory in your J2SE con-

tainer (e.g., Tomcat, JBoss) to enable connection to the appropriate database. In addition, some manual modification of the Hibernate configuration files may be necessary.

Additional optional software to use with the caCORE SDK is listed in Table *3.5*, including the software name, version, description and URL columns. The included (**Incl**.) column indicates (with a **Yes**) if the software is packaged with the SDK. **No** indicates that you must supply the software. A hyperlink is included for your reference to appropriate sources.

Software Name	*Version*	*Description*	*URL*	*Incl.*
Eclipse IDE	3.0 or higher	An open platform for tool integration which provides developers with flexibility and control over their software technology.	http://www.eclipse.org	No
jeteditor-eclipse plugin	0.0.1-alpha-2004-07-22	JET-Editor is an Eclipse-based editor for Java Emitter Template (JET) technology.	http://jet-editor.sourceforge.net/	Yes

Table 3.5 Optional software and technology for the SDK

Documentation and Style Tools

The following tools are part of the SDK framework and are useful for documentation and styling.

- **Javadoc** – Execute the Ant task `doc` to generate Javadocs for your beans. Your javadocs will be generated to the `{home_directory}/output/{project_name}/doc` directory. For more information on Javadoc see http://java.sun.com/j2se/javadoc/.
- **Jalopy** – Execute the Ant task `format` to make your code well formatted. The default indentation format is used in the SDK. This task is configurable to enforce coding standards that you wish to adhere to for your project. See http://jalopy.sourceforge.net/manual.html for information on how to customize this task.

SDK Installation

Complete instructions for installing and performing basic tests for the SDK are found in the *caCORE Software Development Kit 3.2 Installation and Basic Test Guide*, available from http://ncicb.nci.nih.gov/NCICB/infrastructure/cacoresdk#Documentation.

CHAPTER

4

caCORE SDK Process Workflow

This chapter summarizes the process workflow for using the caCORE SDK to generate a caCORE-like, semantically interoperable system.

Topics in this chapter include:

Overview of the SDK Process Workflow

The caCORE SDK facilitates the creation of a caCORE-like service-oriented architecture as shown in *Figure 4.1* and described in *caCORE 3.2 SDK Minimal System Requirements* on page 15 in *Chapter 3*. Based on a model driven architecture, leveraging UML and using the UML model and system properties as inputs, the SDK applies Java JET templates to model elements, then produces a fully functional caCORE-like system.

The SDK was developed to promote semantic interoperability and expedite *n*-tier application development in the research community. Using a process supported by EVS APIs, the model owner can use the SDK to produce a model annotated with EVS concept codes. The SDK process takes the EVS-annotated model in XMI and registers the metadata in caDSR, using the UML Loader described in *Chapter 8*.

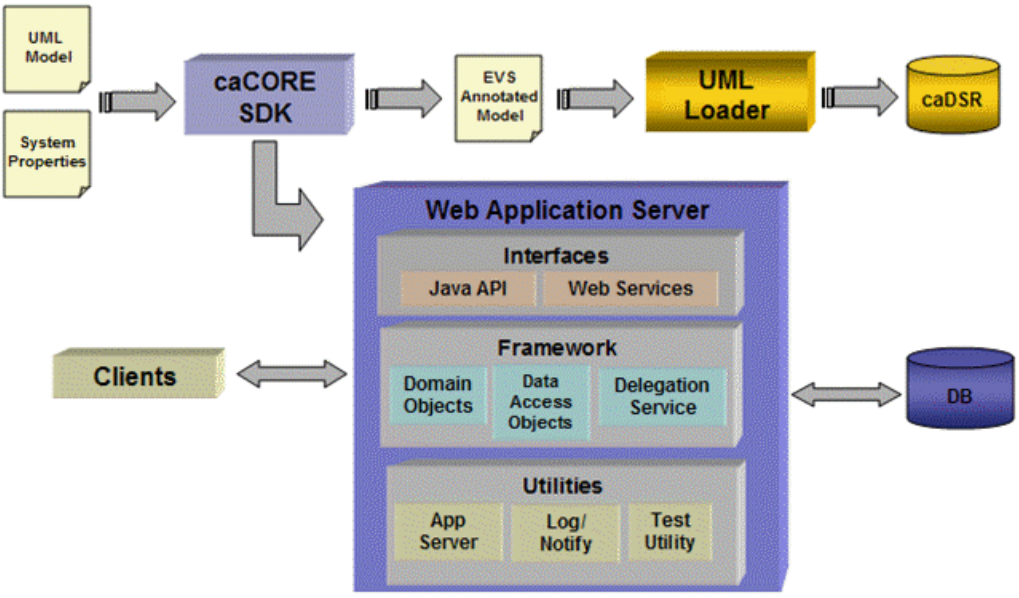


Figure 4.1 SDK process flow

caCORE SDK Components and Their Functions

The caCORE SDK consists of several components:

- Semantic Integration Workbench (SIW)
- UML Loader
- Code Generator

These tools are used to automate creation of the infrastructure described above.

Semantic Integration Workbench

The Semantic Integration Workbench (SIW) is designed to facilitate and streamline the process of semantic integration—the steps whereby data element metadata are mapped to EVS concept codes. This prepares the UML domain models for being loaded into the caDSR and helps developers generate the fully descriptive metadata components described above. The SIW takes an XMI representation of a UML model class diagram, processes the classes and attributes, and then searches EVS for concepts that match the elements of the UML model. The output is a report that lists concept(s) that match each element. The model owner can then select the appropriate concept from a list of candidates, or manually enter an alternative when the SIW finds no satisfactory matches. Then, using the report, the SIW annotates the XMI representation of the UML model with the appropriate concepts for eventual loading into the caDSR.

UML Loader

The UML Loader does the work of registering the metadata in the caDSR. The input to this tool is the semantically-annotated XMI representation of the UML model class diagram that was generated by the developer. Since this XMI is fully annotated with concepts corresponding to Object Classes and Properties (and the data types as defined by the UML attribute itself), the loader has sufficient information to populate most of the

required fields of the caDSR. After loading, the CDEs representing UML attributes are curated to add certain properties, such as valid values of enumerated value domains.

Note: The UML loader is not distributed with this version of the SDK. However, if you create properly annotated XMI representations of your UML models, your models will be loaded into the caDSR by NCICB staff.

Code Generator

The Code Generator actually creates the caCORE-compatible software system. It takes the UML model (including the object model and data model) and generates JavaBeans that are used in the caCORE-like application. It also generates the Object Managers and Data Access Objects that are used by the JavaBeans to retrieve information from the relational databases that are the source of the data itself.

The Code Generator also generates an XML Schema and corresponding Castor XML Mapping files. These files are useful while marshalling and unmarshalling domain objects to and from XML, as well as validating the generated XML. Finally, the Code Generator also creates a Web Services Deployment Descriptor (WSDD) file, which is useful for enabling Web Services access to the generated domain objects.

caCORE SDK Process Flow Details

This section summarizes the process of creating a caCORE-like system using UML and the SDK. This complex interrelated set of activities must be performed in a specific sequence to achieve success. If the process is not followed as suggested, the goal of semantic integration will not be realized. Detailed steps for performing the procedures are included under four consecutive process segments, identified by number in *Figure 4.2* and summarized below.

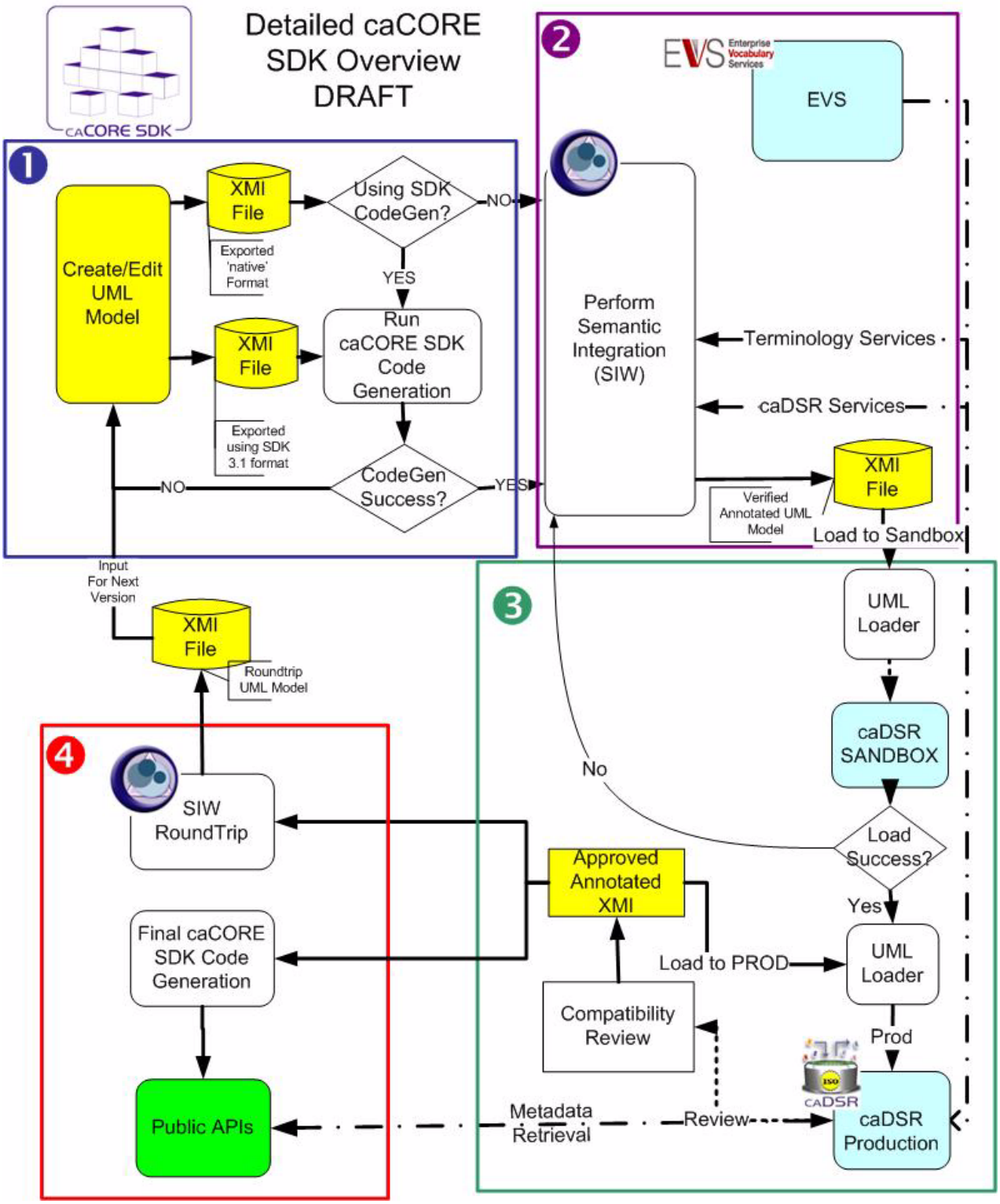


Figure 4.2 caCORE SDK workflow yielding semantically-integrated APIs; caCORE infrastructure components = light blue, caCORE SDK components = white; artifacts (documents) = yellow; generated software system = green

Step-by Step Workflow

1. Design the system and draw the model. *See Chapter 5.*
    a. Create use-case artifacts (optional). See *Creating Use-case Artifacts* on page 37.
    b. Create class diagrams (required). See *Creating a Class Diagram* on page 39.
    c. Create data model and mapping diagrams (recommended, but optional if you have an existing database schema). See *Creating a Data Model* on page 52 and *Creating Manual ORMs* on page 107.
    d. Create sequence diagrams (optional). See *Creating a Sequence Diagram* on page 66.
    e. Generate XMI file from UML model (required). See *Generating XMI* on page 66.
    f. Generate DDL from data model (optional). See *Generating Data Definition Language* on page 67.
2. Annotate the model. *See Chapter 7.*
    a. Run the Semantic Integration Workbench to perform semantic integration. See *Semantic Integration Workbench* on page 92.
    b. The names of entities (class, attribute) in the UML Model are semantically annotated by EVS in an iterative process with the model owner.
    c. The model owner reviews the Annotated XMI file and, if accepted, completes the submission template and sends a request to NCICB. (ncicb@pop.nci.nih.gov).
3. Register the Metadata. See *Chapter 8*.
    a. The UML model is loaded to the caDSR Sandbox environment.
    b. If the model loads successfully, it is submitted to load to caDSR Production for final curation and a compatibility review.
4. Generate Code and Deploy the System. See *Chapter 9*.
    a. Once the model has passed the compatibility review, the model owner can use the SIW to run RoundTrip, which will insert the caDSR metadata identifiers for the model into the XMI as tagged values and ready the model for reuse or the next version.
    b. Model owners can also produce the final public APIs using the SDK code generator.

If a change is required in a UML entity (e.g., class, attribute, or relationship) as a result of running the semantic interoperability steps (2 and 3), it may be necessary to repeat certain parts of the process. For example, if, during curation of caDSR metadata, the decision is made to change an attribute to reuse an existing CDE, the model will have to be updated and re-run through the Semantic Integration Workbench.

5. After the first four steps are complete, users can optionally enable CSM, Session Management, and Simple Writable APIs. See *Chapter 10*.

a. Use the generated domain objects to produce simple writable APIs for the application (optional).

b. Integrate with CSM authentication, authorization, and user provisioning (optional).

End Result: A caCORE-Like System

The end result of using this SDK is a caCORE-like system, tailored to your specific needs, that allows you to run a Java program to query your persistence layer (the actual data storage layer, which is generally a relational database system). The generated system and artifacts are essentially the same as a caCORE system, meaning that the code generation and ORM artifacts generated during the process described in this chapter are using "out-of-the-box" SDK generation components. With the exception of the UML Loader, all of the components and generated artifacts described are available for your use or reference. You can create a new model or enhance the example models by following the procedures in this chapter.

CHAPTER

5

CREATING THE UML MODELS

This chapter contains all of the necessary procedures to create a UML model for the caCORE-like system.

Topics in this chapter include:

Prerequisites

Before proceeding with this chapter, it is essential that you have completed the steps in the *caCORE SDK 3.2 Installation and Basic Test Guide* available at http://ncicb.nci.nih.gov/NCICB/infrastructure/cacoresdk#Documentation. Doing so ensures that your system is configured correctly. To enhance the example models or create new models, you must install Enterprise Architect or another UML 1.3 compliant modeling tool that produces XMI 1.1 output format. For more information, see the Installation and Basic Test Guide.

Note: All of the examples and screenshots included in this chapter are Windows specific. If you are using a different platform, then modify the information as appropriate for your system.

Introduction

The processes described in this chapter use the international standard modeling notation, UML, for specifying, visualizing and documenting modeling diagrams (artifacts) of an object-oriented modeling system. The caCORE team bases its software development, as well as this caCORE SDK, primarily on UML. *Appendix A* is included in this guide to familiarize the reader who has not worked with UML with its background and notation. As you follow the steps in this chapter that refer to UML models, you may want to refer to *Appendix A* for more information.

Enterprise Architect (EA) (http://www.sparxsystems.com.au/) was used to create the example UML models included with the SDK and the screen captures of the UML modeling process shown throughout this chapter. It is not a requirement of the SDK that you use EA, but the modeling tool you use must be capable of exporting the UML model in a format that is XMI 1.1 compatible and is a valid Metadata Repository (MDR) XMI file. The XMI produced by EA is not a valid NetBeans Metadata Repository MDR XMI file since it contains some specific EA characteristics. The SDK includes an XMI preprocessor which can be invoked by calling an Ant task, fix-xmi, to make some minor modifications to the structure of a given XMI file before semantic connection and code generation begin. If you use a modeling tool other than EA, you will have to make sure you have a valid MDR XMI file.

EA was chosen for the following reasons:

- EA is an object-oriented tool supporting full life-cycle development
- EA is a flexible, complete, and powerful UML modeling tool
- EA facilitates the system development, project management, and business analysis process

Modeling Constraints

The caCORE SDK places no constraints on the structure of the class diagrams, so you are free to concentrate on creating a model that captures the objects in a scientific or real-world domain with no thought to code generation. However, while the SDK framework does not constrain the contents of models, the SDK transformers (the tools that perform semantic connection and generate source code) do place constraints on the models themselves. You must be aware of the following constraints placed on models by the SDK transformers:

Constraint 1: Allowable UML Elements—Only UML class elements are recognized by SDK tools. Classes may contain both attributes and operations, however operations will be disregarded by the SDK tools.

Constraint 2: Attribute Types—Each attribute must have a type assigned to it, and the type must be a Java primitive data type or one of the Java wrapper objects, with the class defined within the model. For Java class types, you must add default class declarations such as `java.lang.String` as shown in Table *5.9* on page 42 (for example, `java.lang.String`, `java.util.Date`, etc.). In accordance with object-oriented design principles, attributes of a class that are of a complex type should be modeled as associations. The only exception is if you create an enumeration of acceptable values as a caDSR Value Domain stereotyped class in your UML Model as described in *Chap-*

ter 7 on on page 133. For a UML attribute to use this value domain, the model owner adds a tagged value: 'CADSR Local Value Domain' / 'My Value Domain', as described in *Pointing a UML Attribute to a Value Domain* on page 136. Alternatively, for the purpose of building enumerated values for common data elements, Value Domains can be added to caDSR during the curation stage.

Constraint 3: Allowable Relationships—The SDK tools recognize association and generalization (inheritance) relationships. In addition, the UML Loader records aggregations and compositions when registering metadata, but for the purposes of generating common data elements, these are treated the same as simple associations.

Constraint 4: Association End Role Names—Both ends of each association must be given a role name.

Constraint 5: Association End Multiplicity—The multiplicity of each association end must be specified at both ends.

Constraint 6: Association End Navigability (Directionality)—The navigability of each association end must be specified at both ends.

The characteristics of associations described in Constraints 4, 5 and 6 are used by the Code Generator to determine where collection classes should be used, whether get/set methods should be generated and, if so, what they should be named. In order for the Code Generator to function properly, and as a general guideline to enhance the semantic richness of the model, it is suggested that you use the following naming convention: the *role name* should be set to the name of the associated class and should begin with a lower-case alphabetic character. If the multiplicity at that end of the association is greater than 1, the word “Collection” should be appended to the role name.

Constraint 7: “Logical Model” Package—ALL classes and attributes must have textual descriptions to facilitate semantic interoperability.

- A definition for each class must be entered into Tagged Values using "documentation" as the tag (the term is case-sensitive). The "documentation" tagged value is used during semantic integration by the Semantic Connector/UML Loader.
- A definition for each attribute must be entered into Tagged Values using "description" as the tag (the term is case-sensitive). The "description" tagged value is used during semantic integration by the Semantic Connector/UML Loader.

Notes: The method of adding tagged values differs for each UML modeling tool; refer to the documentation for your tool to determine how to add these tagged values.

Constraint 8: id Attribute—For the SDK to properly generate the java bean’s `equals(Object obj)` and `hashCode()` methods, every domain object must have a mandatory “id” attribute. This attribute must be called “id”.

Constraint 9: Java Limitations—Because the SDK produces a Java-based system, class and attributes names are subject to the conventions defined by the Java language. UML models should not use Java reserved words (e.g. int, long, class, interface, void, etc.) as class or attribute names. Also, model elements must not use hyphens, angle brackets or other reserved characters that will result in Java compilation errors.

Naming Best Practices

To ensure semantic interoperability, you should pay close attention to the class and attribute naming conventions established by the Sun Microsystems Java Bean Specification. Decisions as to how to name UML entities affect the creation of common data elements in caDSR by the UML Loader, as well as the generation of the system code by the Code Generator. The following best practices, while not an exhaustive list, provide some broad outlines to be considered during modeling.

- Adopt a consistent naming convention that is used throughout your model and, if possible, across your organization.
  As an example, the Sun Microsystems Java Bean Specification establishes the use of camel case for class and attribute names:
  - Classes

    --One word class names are capitalized (for example, Taxon, Location).

    --Multiple word class names contain multiple words, with no space between words. The first and subsequent words are all capitalized. (for example, SequenceVariant or TaxonCollection). Consult the specification for details.
  - Attributes

    --One-word attribute names should be all lowercase (for example, name or taxons).

    --Multiple word attribute names contain multiple words, with no space between words. The first word is lowercase, while the first letter of second and subsequent words should be capitalized (for example, camelCase [the name of this convention] or geneTitle).

  In addition to enhancing readability, the camel case convention enhances the interoperability process. The Semantic Connector uses capitalization and underscores to separate multi-word attributes. For more information, see *Chapter 7*.
- To the extent possible and reasonable, avoid abbreviations and acronyms (for example, use 'DatabaseReference' instead of 'DbRef').
- Avoid the use of 'jargon' terms where standard terms exist (for example, use 'microarray' not 'chip').
- Do not repeat class names in attributes. For example, in the 'Gene' class use attribute 'name' not 'geneName'. Using the latter format causes the UML Loader to unnecessarily repeat the concept code for 'Gene' twice in a single CDE, which is semantically undesirable. For more information, *see UML Loader* on page 139.
- Do not use Java reserved words as class or attribute names, as this prevents the Code Generator from compiling the generated system code.

Creating Use-Case Artifacts

Producing use-case artifacts is an optional but recommended step in the software development life cycle. The caCORE development team uses use-case analysis to capture high-level system requirements. The first artifact created is the use-case docu-

ment. A use-case document provides structured textual descriptions of how an actor will interact with the system. *Figure 5.1* displays an example Use-case document.

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

Usecase ID:100300

Actor

- caBIO Application developer

Starting Condition

The actor establishes reference to the caBIO software

Flow of Events

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.

End Condition

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

Figure 5.1 Example use-case document

Using the use-case document as a model, a use-case diagram is then created. A use-case diagram, which is language independent and *graphically* descriptive, signifies what a system does from the perspective of an external viewer. An example use-case diagram created from the use-case document is illustrated in *Figure 5.2*.

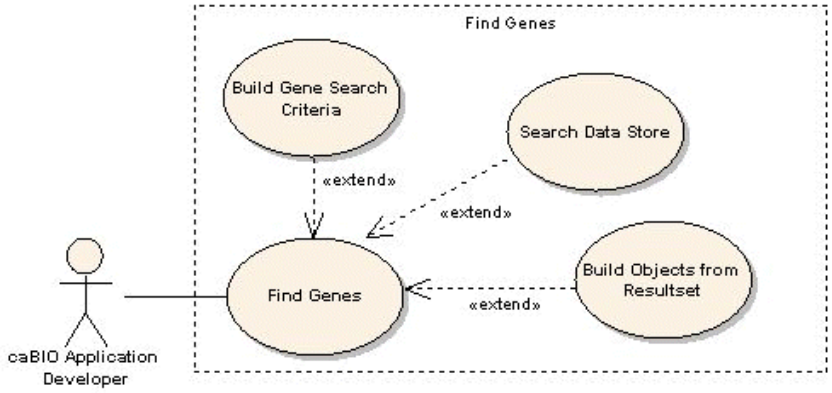


Figure 5.2 Example use-case diagram

See *Use-Case Documents and Diagrams* on page 190 for more information on use-case artifacts. Step-by-step procedures are not included in this guide to produce use-case artifacts because they are not required to use the SDK.

Creating a Class Diagram

Class diagrams are created to define the static attributes, functionalities, and relationships that must be implemented in the software. Software developers who know UML design the system's object models. When designing a class diagram, they use the information from the use-cases while thinking about the code that must be generated. For users interested in a caCORE-like infrastructure, class diagrams also form the basis for creation and registration of semantically unambiguous caDSR metadata.

Note: If you are planning on saving your EA model in `.eap` format in Concurrent Versions System (CVS), make sure you check your file into CVS as a binary file. Otherwise, when you check it back out, your file will not load into EA. Alternatively, use of the built-in version control capabilities of EA avoids this problem.

Opening the caBIO Example Model

Perform the following steps to open the caBIO example model using EA.

1. Open Enterprise Architect and select **Open a Model File**. The Select Enterprise Architect Project to Open dialog box displays.
2. Select the desired project name from the list (for the example, select `models\cabio.EAP`) and click **Open**. The Project View displays. (If the Project View is not displayed, select **View > Project Browser** from the main menu bar.)
3. In the Project View, navigate to the package **Views > Logical View > Logical Model**, and double-click on the Logical Model class diagram. The class diagram displays as shown in *Figure 5.3*. See *Class Diagrams* on page 192 for more information about class diagrams.

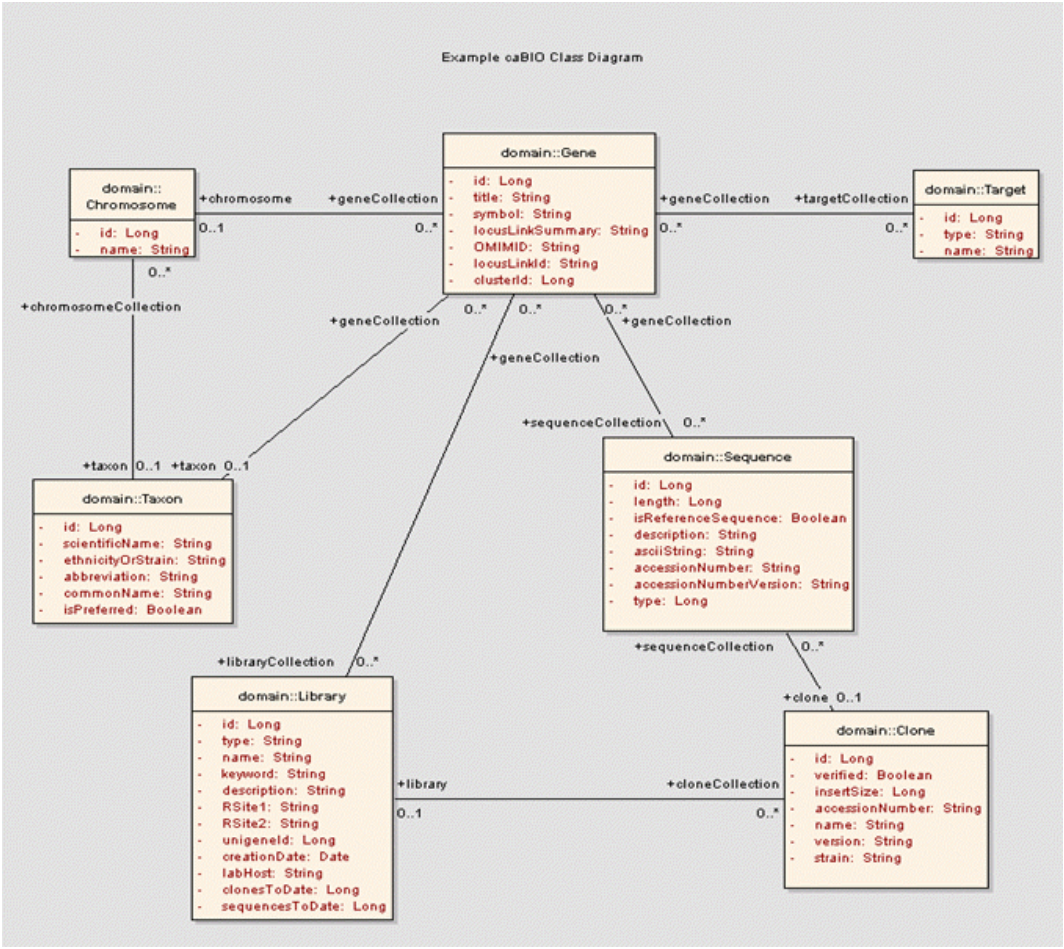


Figure 5.3 Example caBIO Class Diagram

The caBIO model shown in *Figure 5.3* contains seven of the core objects, a subset of the complete caCORE object model, from the `gov.nih.nci.cabio.domain` package. You must create a new project (for example, `cabio.eap`)as described in the following section to create your own project in EA or you can enhance the example caBIO model by skipping the *Creating a New Project* section that follows this section and continuing with the rest of this chapter beginning with *Creating a New Element (Class)* on page 39.

Creating a New Project

This section provides procedures to produce a new project using Enterprise Architect (EA).

Perform the following steps to create a new model using EA.

1. Open Enterprise Architect and select **Create a New Model** from the Model Management window or **File > New Project**... from the main menu. The Create New Enterprise Architect Project dialog box opens.
2. The Model Project field allows you to select a template from which your model can be created. The caCORE SDK includes a model project, `{home_directory}\models\caCORESDKTemplate.EAP`, that contains Java primitive "wrapper" types for use in semantic integration and code genera-

tion. Enter the location and name of this file here (for example, `c:\cacore-toolkit\models\caCORESDKTemplate.EAP`)

3. Enter the New Project name and directory (for example, `C:\Program Files\Sparx Systems\EA\cabio.eap`) and click Create Project as shown in *Figure 5.4*. Your model name appears under Recent Models on the EA Start Page.

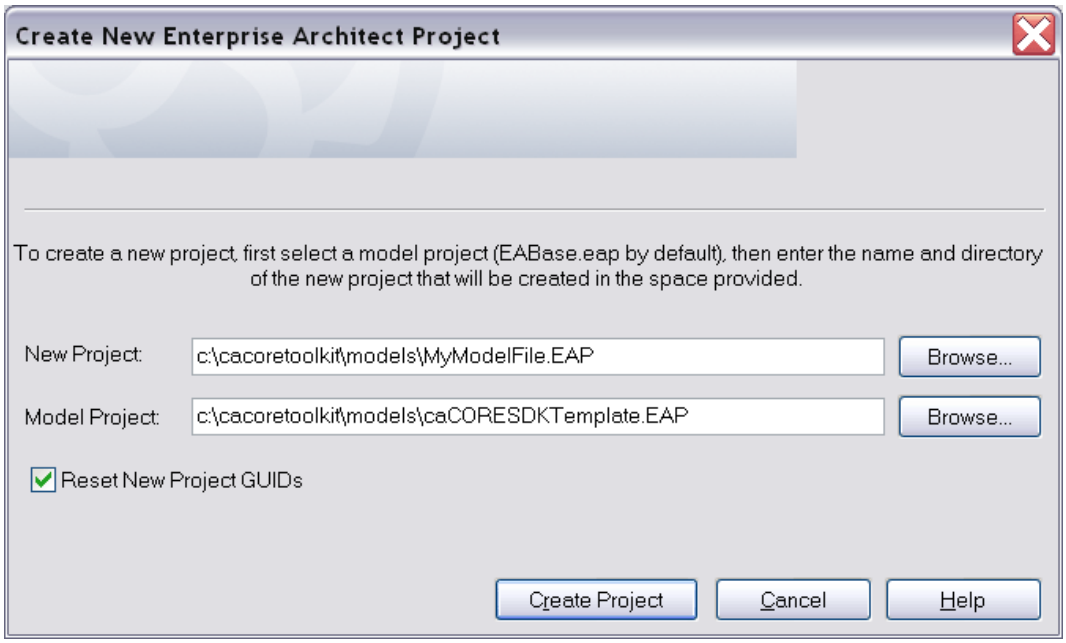


Figure 5.4 Create a New EA Model

4. Select **View > Project Browser** from the main menu bar of EA. The Project View displays.
5. Click the **Logical View** plus sign. The Data Model and Logical Model folders display.

6. Click the **Data Model** plus sign and **Logical Model** plus sign. The Project View displays as shown in *Figure 5.5*.

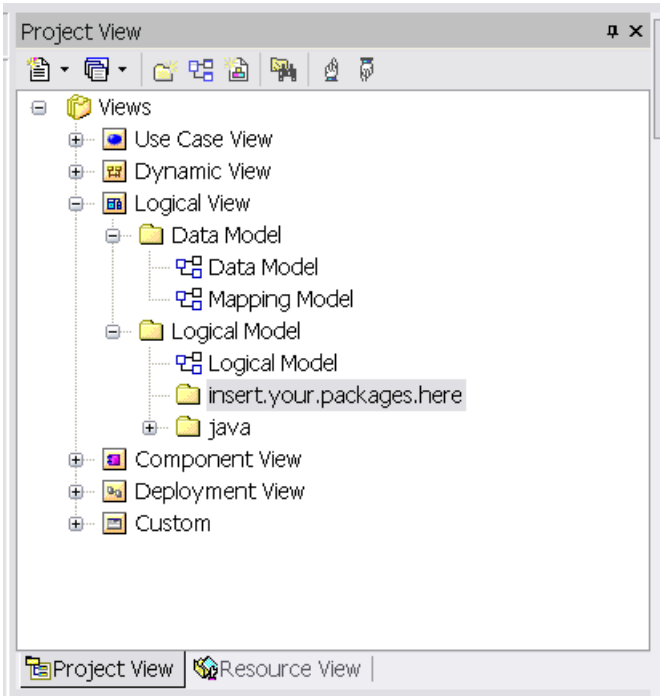


Figure 5.5 Logical View in EA

Creating a New Element (Class)

The following sections provide step-by-step procedures to produce a class diagram using Enterprise Architect (EA). You can enhance the example model provided or create a new model.

For more information on class diagrams, see *Class Diagrams* on page 192.

Before creating a new element, you may want to review *Modeling Constraints* on page 32.

Note: **Constraints 2 through 6** simply require that the model be complete. Furthermore, these constraints apply only to those UML classes that are actually selected for code generation and semantic connection purposes. The code generation transformer places no constraints on model elements that are not selected.

Perform the following steps to create a new class and attributes using EA.

1. In the Project View, right-click Logical Model folder, select **Add > Add Element**. The Insert New Element dialog box displays.

   Note: For the SDK code generation and semantic connector processes to work, it is very important that you create your classes under the Logical Model package, since you can also create classes under the Logical View in EA.

2. Enter the information shown in *Figure 5.6* and listed in the bullets below *Figure 5.6*.

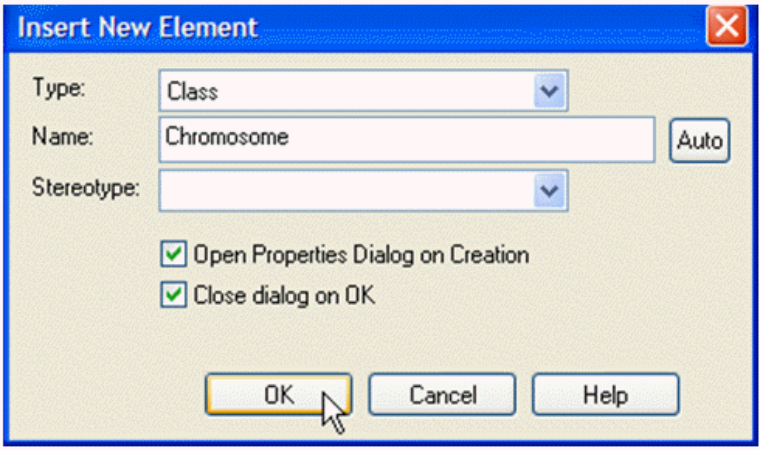


Figure 5.6 Insert New EA Element

- Leave the **Type** selected as *Class*.
- Enter ***{Chromosome}*** (or the name of another class) as the **Name** to be consistent with your model.
- Select both check boxes.
- Click **OK**.

Note: **Constraint 1 - UML Class Elements Only.** Only UML Class elements may be used when inserting a new element as shown in *Figure 5.6*. When creating UML class diagrams to describe a particular domain of objects, it is common to use both class elements and interface elements. Interface elements are used to describe the behavior of a class of objects, and so they should contain only operations. Class elements may contain both attributes and operations.

The purpose of this example is to produce a data access API, so we are mostly concerned with the names and types of the attributes of each class. The behavior of each class is the same: there are operations to retrieve and modify the value of each attribute. Therefore, interfaces are not needed.

For more information about modeling constraints, see *Modeling Constraints* on page 32.

3. From the Project View select the class just added (for example, **Chromosome**) and select the **Element > Tagged Values** from the main menu bar. You can also display the **Tagged Values** tab by using the shortcut key CTRL+SHIFT+6. The **Tagged Values** dialog displays.
4. From the Tagged Values dialog, click the new tag icon and enter ***documentation*** in one field and enter the UMLdescription for the class.

Note: **Constraint 7** – A UMLdescription for the class must be entered in tagged values. The "documentation" tagged value is used during semantic integration by the Semantic Connector/UML Loader.

5. Right-click {your class} and select **Properties** to display the Class properties window. Select the **Detail** tab and click **Attributes** as shown in *Figure 5.7*.

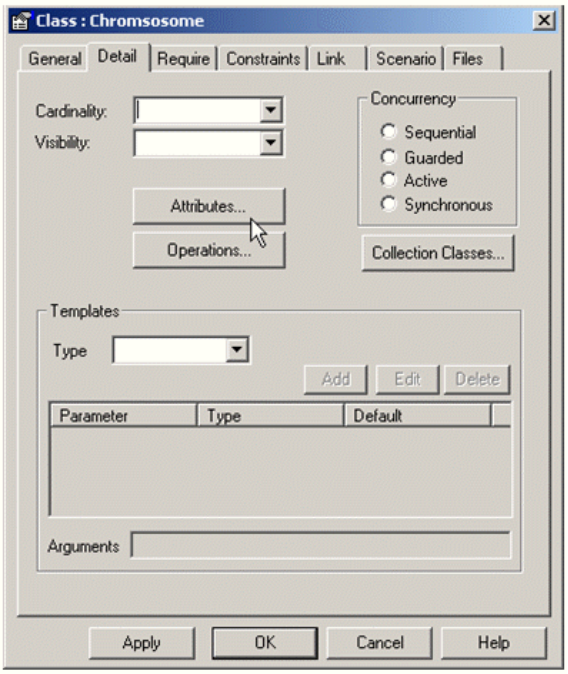


Figure 5.7 Class Detail Dialog

6. The {Class} Attributes dialog box displays with the **General** tab selected. Enter the attribute information as shown in *Figure 5.8* and listed in the bullets below *Figure 5.8*.

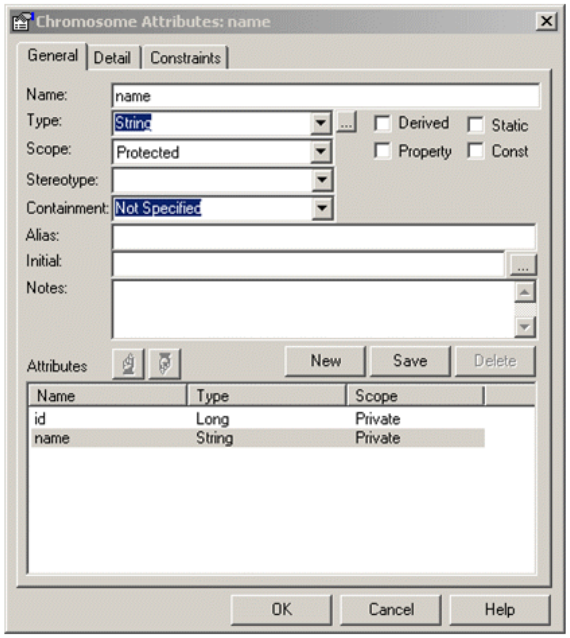


Figure 5.8 Creating an Attribute

- Enter the **Name** of the attribute. Ensure that you adhere to naming conventions described in *Naming Best Practices* on page 34.

- In the **Type** list, type your specified data type for primitive data types or click the Browse button (...) to select data types as shown in *Figure 5.9*.
- In the **Scope** list, select the appropriate scope (**Public**, **Protected**, **Private**, or **Package**).
- Enter any other pertinent information about the attribute such as an alias or specific notes about the attribute.
- Click **Save**. The **Name**, **Type**, and **Scope** are displayed at the bottom of {Class} Attributes dialog.
- Click **New** if you want to add more attributes; enter additional information as described above.
- Click **OK** when you are finished adding attributes for the class.
- From the **Project View**, click the newly created class, and then click the plus sign to display the newly added attributes.

Note: **Constraint 2 – Attribute Types.** Attribute types must be classes defined within the model or primitive data types. It is strongly recommended that you use data types defined within your model rather than primitive data types. Primitive data types will work, but you will need to refer to the Hibernate documentation for handling null values within your database. When creating an attribute as shown in *Figure 5.8*, you must specify the attribute's name and its type. In most tools, the modeler has the option of simply entering a string value or opening another dialog box in which another class that is already defined in the model can be selected. The SDK used by this example requires that object types be valid Java types and that primitive data types be the names of Java primitive types. For example, to specify that an attribute is an integer, use the object "`java.lang.Integer`" type or the primitive "`int`" type. *Figure 5.9* displays the added `java.lang.*` data type objects. You must add data type objects for any Java types you use.

For more information about modeling constraints, see *Modeling Constraints* on page 32.

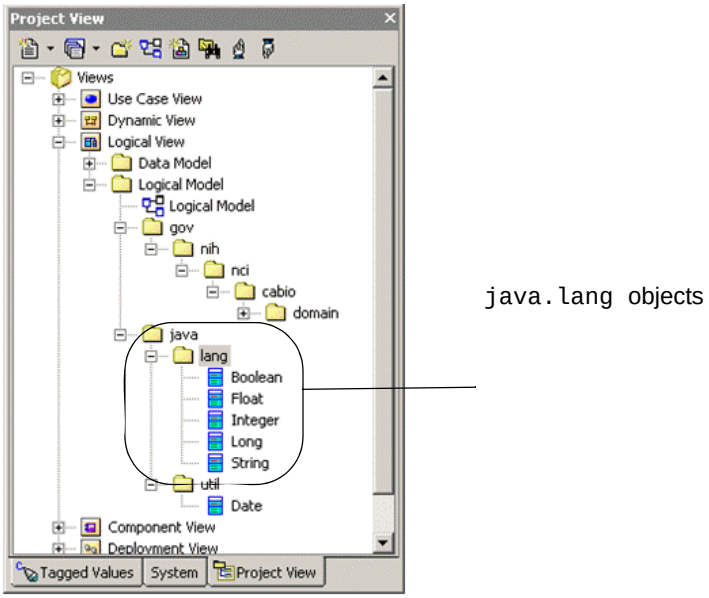


Figure 5.9 Data Type Objects

7. From the Project View select the attribute just added (for example, Name) and select **Element > Tagged Values** from the main menu bar. You can also display the **Tagged Values** tab by using the shortcut key CTRL+SHIFT+6. The Tagged Values dialog box displays.
8. From the Tagged Values dialog box, click the new tag icon and enter ***description*** in one field and enter the UMLDescription for the attribute.

Notes: **Constraint 7** – A UMLdescription for the attribute must be entered in Tagged Values. The "description" tagged value is used during semantic integration by the Semantic Connector/UML Loader.

Valid values lists, if entered, are not supported by the Semantic Connector and UML Loader at this time.

Creating Additional Classes

Create additional classes by following the procedures in the *Creating a New Element (Class)* on page 39 until all of your classes and their corresponding attributes have been added under the Logical Model in the Project View.

Creating a Logical Model Object Diagram

Perform the following steps to create a Logical Model object diagram.

1. From the Project View, double-click on the **Logical Model** icon and then select the **Logical Model** tab at the bottom of the EA page.
2. Drag and drop each class from the **Project View** to the **Logical Model** diagram. Paste the class to the diagram as a simple link as shown in *Figure 5.10*. Click **OK**.

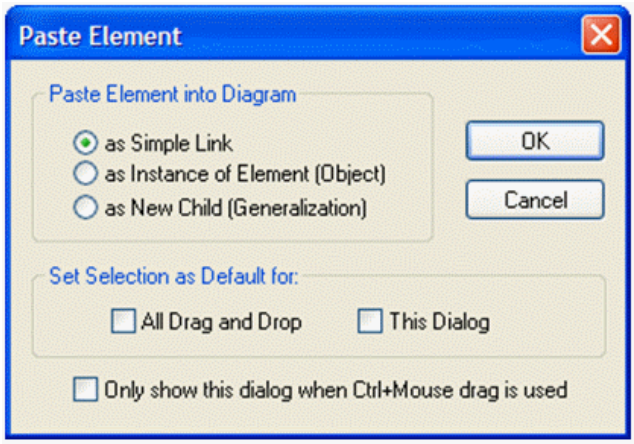


Figure 5.10 Paste Class as Simple Link

Creating Relationships between Classes

For a detailed review of relationships between classes, see *Relationships Between Classes* on page 194.

Perform the following steps to create relationships between classes.

1. From the menu bar, select **Link > Association** as shown in *Figure 5.11*.

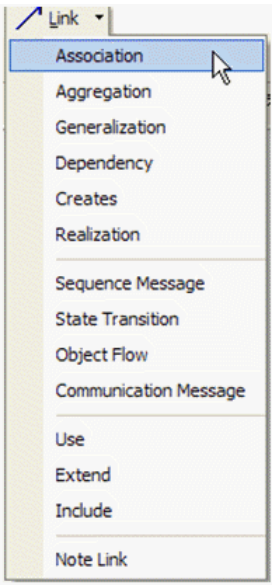


Figure 5.11 Types of Links

Note: The example uses the Association Link. For your own implementation, choose the appropriate type of relationship between classes. Only association and inheritance relationships are mapped to caDSR metadata. For more information on relationships, see *Relationships Between Classes* on page 194.

2. Your cursor becomes a hand in the Logical Model. Click and hold one class (this is your source), then drag and drop to the second class (this is your target) to create the desired link. The type of connector selected displays on the logical model. An association is shown between Gene and Chromosome in *Figure 5.12*.

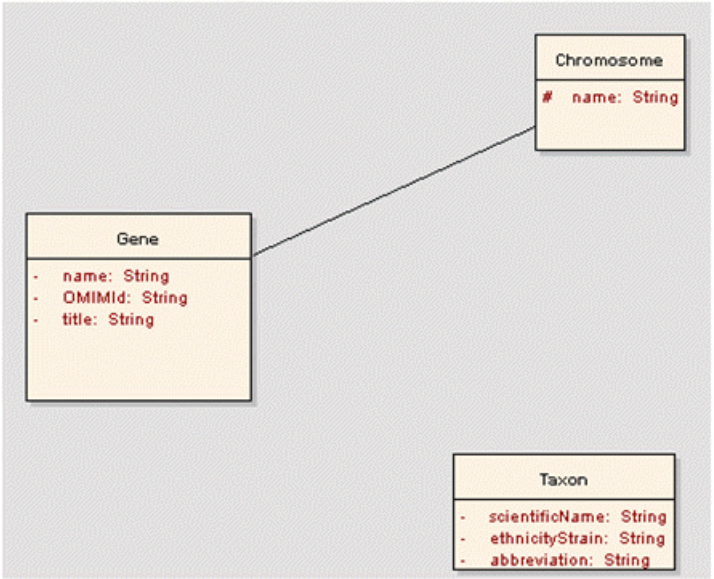


Figure 5.12 Association between Gene and Chromosome classes

3. Right-click on the link created and select **Association Properties**. The Association Properties dialog box displays as shown in *Figure 5.13*. For general information on associations, see *Relationships Between Classes* on page 194.

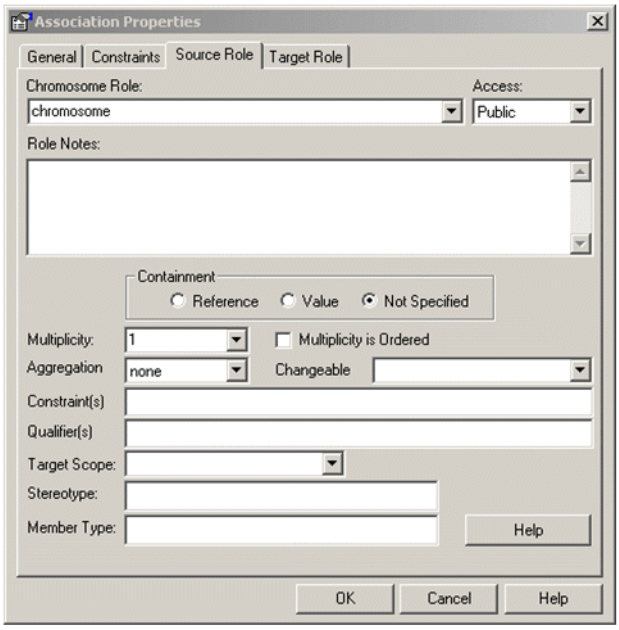


Figure 5.13 Association Properties Window

4. Click the **Source Role** tab as shown in Figure 36. Enter the role name in the **{Class} Role** field (for example, for the association between Chromosome and Gene, the source role name is chromosome).

Note: **Constraint 4–Association End Role Name**. An association between two classes indicates that objects of those classes are related, and the UML depiction must describe the meaning of that relationship. Each association end must be given a role name; it indicates the role that the object on the corresponding end plays in relation to the object on the other end of the association.

(Naming conventions for role names are very important because the role names are used to create method names. For more information, see *Naming Best Practices* on page 34.)

For example, a gene is usually found on a single chromosome. In the example caBIO model as shown in *Figure 5.14*, the Chromosome-end of the association between Gene and Chromosome has the name "chromosome" and the Gene-end has the name "geneCollection".

You can also add additional information about each association end and the association as shown in *Figure 5.13*. If additional information is provided, it can be used by the SDK to produce documentation in the generated source code. See *Relationships Between Classes* on page 194 for more information on associations.

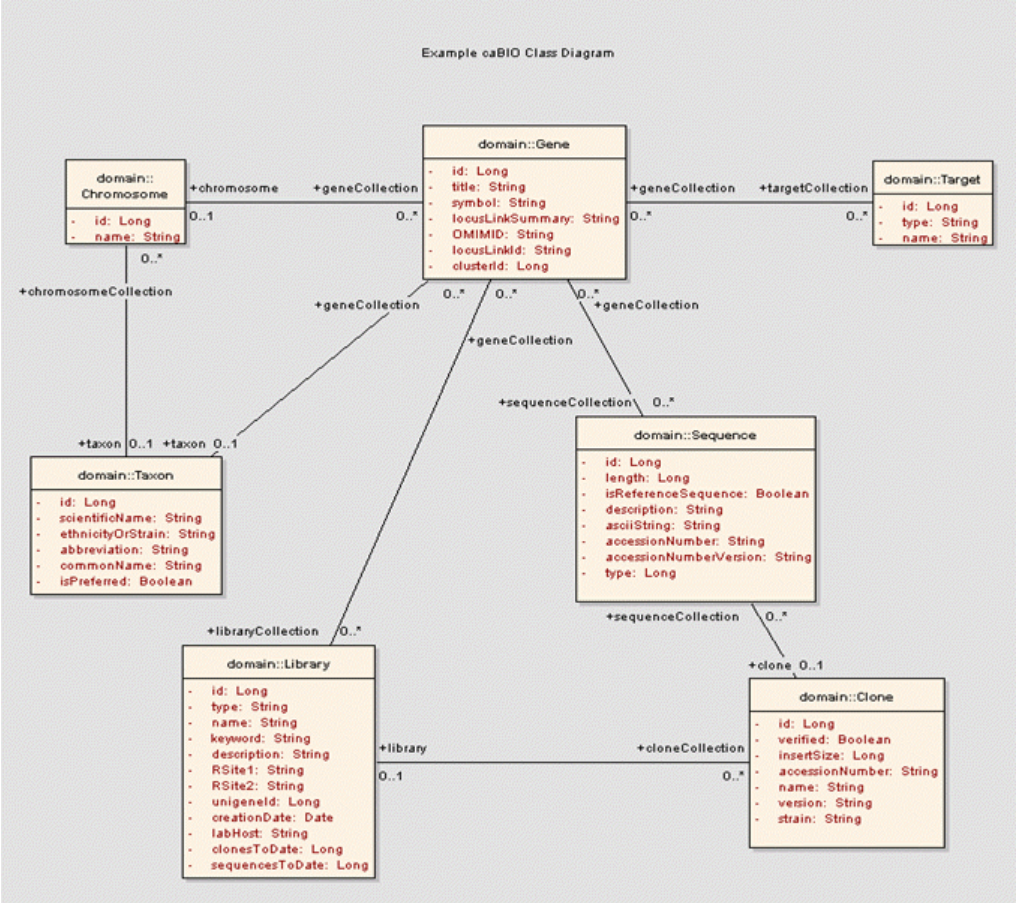


Figure 5.14 Example caBIO Class Diagram

5. From the **Multiplicity** drop-down list in *Figure 5.13*, select the appropriate multiplicity (for example, chromosome has a multiplicity of 0..1). Click **OK**.

Note: **Constraint 5 – Association End Multiplicity**. The multiplicity of each association end must be specified. The concept of multiplicity is a component of the concept of cardinality. The multiplicity of an association end indicates the number of objects of that class type that may be associated with a single object of the class on the other end of the relationship. See *Multiplicity* on page 194 for common multiplicity values.

6. Right-click on the link created and select **Association Properties**. The Association Properties dialog box displays.
7. Select the **Target Role** tab. Enter the role name in the **{Class} Role** field (for example, geneCollection). From the **Multiplicity** list, select the multiplicity of choice as shown in *Figure 5.13*. Click **OK**.

Note: Follow the information and constraints included in step 4 on page 45 and step 5 on page 46 as specified for the **Source Role** tab.

8. Right-click on the link created and select **Association Properties**. The Association Properties dialog box displays with the **General** tab selected as shown in *Figure 5.15*.

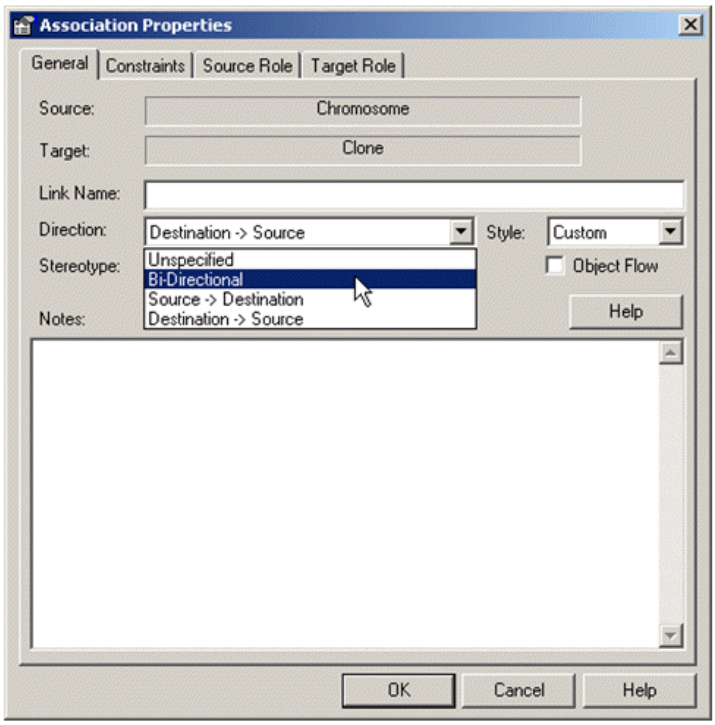


Figure 5.15 Association Properties Dialog

9. Select the **Direction** to indicate the navigability as shown in *Figure 5.15*. Click **OK**.

Note: **Constraint 6–Association End Navigability.** The navigability of each association end, which indicates the direction(s) in which an association may be traversed from one object to another, must be specified at the association ends.

In the caBIO model as shown in *Figure 5.14*, the association between Gene and Chromosome is navigable in both directions. For example, if one has a Gene object, one can ask for the Chromosome on which it is found. And if one has a Chromosome object, one can ask for all Gene objects that are found on it. See *Relationships Between Classes* on page 194 for more information on navigability (directionality).

It is possible for associations to have more than two ends (that is, they may be of degree *n,* where $n > 2$). This document does not address such associations, as they are not found in the SDK.

10. Drag and move the role names and multiplicities to ensure readability.
11. Create associations between your other classes.

As a reminder, for a detailed review of relationships between classes, see *Relationships Between Classes* on page 194.

Creating a Data Model

Extensions to UML to model relational databases are collectively known as the UML Data Modeling Profile. The UML Data Modeling Profile is not a ratified standard but is widely accepted since it allows you to model database tables, columns, keys, triggers, constraints, and other relational database features. Besides modeling database tables, you can generate scripts to create the tables for your databases.

- Creating a data model is optional, but it allows you to automatically create your Object Relational Mapping (ORM). If you do not have an existing database, then creating a data model is the recommended procedure.
- If you have an existing database, it is sometimes possible to import your database schema into the modeling tool you are using. This capability does exist in EA and was used to import the caCORE database schema. Creating your data model this way can save considerable development time.
- If you do not create a data model, then you must create a manual ORM as described in *Using Custom OR Mappings* on page 172.

ORM provides the mapping from an object to data. The benefits of ORM are as follows:

- Exposes data as objects.
- Facilitates the creation, restoration, persistence, and deletion of objects in a relational database.
- Provides model driven data access.
- Permits transformation of data to different formats (XML).
- Allows for the development of code regardless of the data source (Oracle, SQL Server, DB2, MySQL, and so forth).

ORM is an advanced topic and a thorough discussion of it is outside the scope of this document. For more information, go to http://www.chimu.com/publications/objectRelational/ or http://www.service-architecture.com/object-relational-mapping/. The caCORE SDK uses Hibernate (http://hibernate.org) to provide the object relational mapping. The object and its attributes are mapped to corresponding tables and fields in the database.

Opening an Example Data Model

From the Enterprise Architect (EA) **Project View**, navigate to the package **Views > Logical View > Data Model**, and double-click on the **Data Model** diagram. The diagram shows the physical data model for the caBIO example as displayed in *Figure 5.16*.

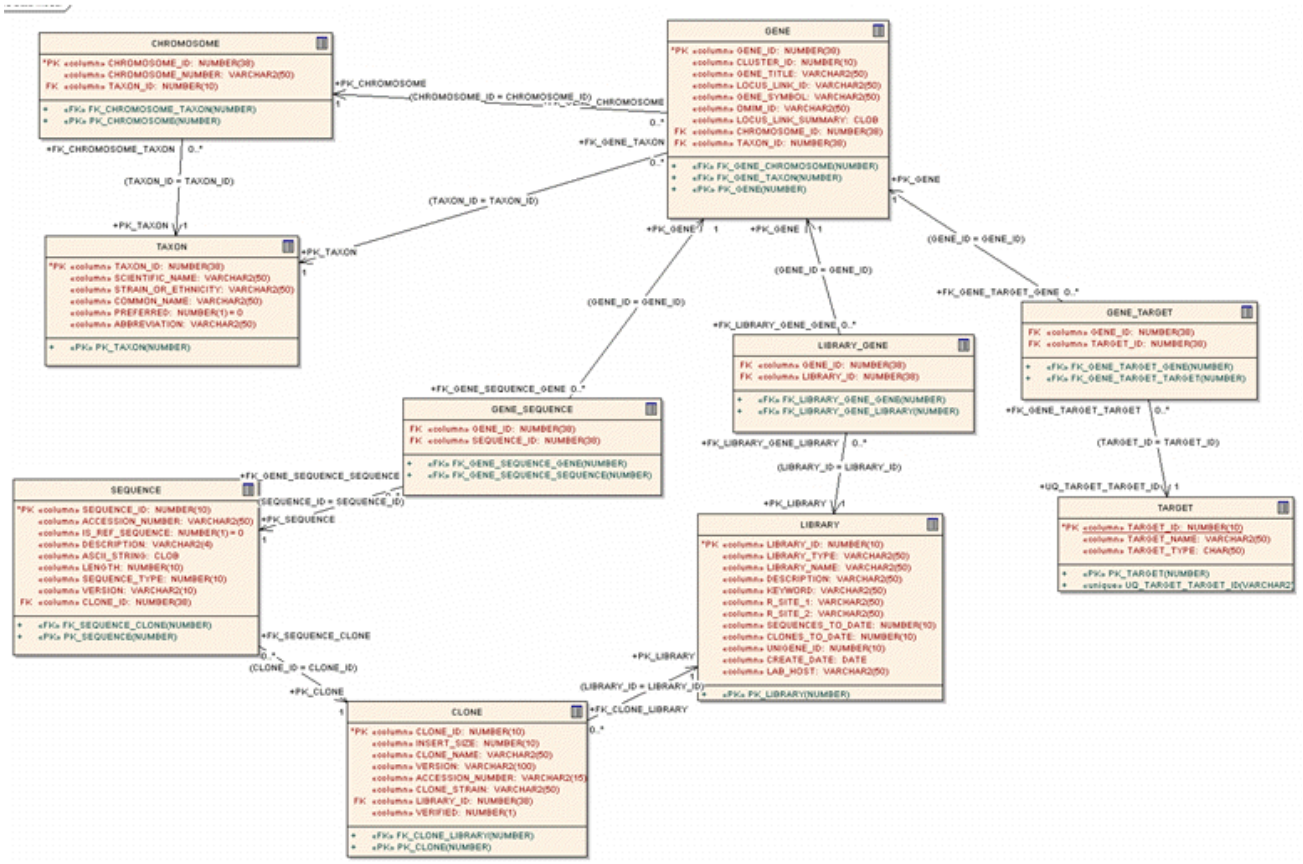


Figure 5.16 Example caBIO Data Model

Data model diagrams are similar to class diagrams since they both show classes and associations among classes. The difference is that the classes in data model diagrams are of stereotype “table”, so EA displays them differently and supports different operations on them.

Note: The example caBIO model does not exhibit inheritance.

Creating a New Data Model

This section explains how to create a data model that specifies how objects should be stored in a relational database. Such a specification is called object relational mapping (ORM). This section describes a single approach that provides efficient access/persistence for the majority of possible object models. For more involved relational systems, designing the database separately and then mapping to the object model may result in better system performance.

This approach requires mapping an object model to a data model. The following mappings must be constructed:

- Class-to-Table
- Attribute-to-Column
- Association-to-Relation(s)
  Creating Class to Table Mappings

Perform the following two steps to create the Class-to-Table mappings.

1. Create a table in the data model diagram for each class in the class diagram.
2. Create dependencies from tables to classes.

Creating Data Models (Tables)

Create a table in the data model diagram for each class in the object model. Each table, as shown in *Figure 5.16* on page 49, has a name that is similar to the name of its corresponding class, as shown in *Figure 5.14* on page 46. For example, the Gene object model class maps to the GENE data model table. Using a consistent naming strategy is good practice because it makes the model easier to understand. Such a naming strategy is not necessary, but using the same name that is in the object model with capital letters for the data model is recommended by the caCORE team.

Perform the following steps to create a data model table using EA.

1. Right-click the **Data Model** folder and select **Insert > New Element**. The Insert New Element dialog box displays.
2. Enter the information as shown in *Figure 5.17* and described in the bulleted items below *Figure 5.17*.

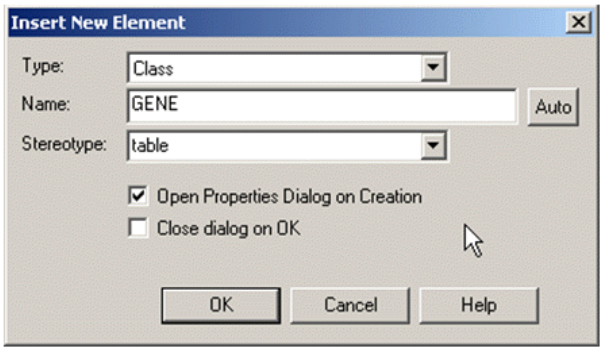


Figure 5.17 Insert Data Model

- Leave the **Type** selected as Class.
- Enter ***{GENE}*** (or another required name) as the **Name** to be consistent with your model. The caCORE team recommends that you enter the same name that was used in the object model, but capitalize all the letters for the data model.
- In the **Stereotype** list, select **table**.
- Select both check boxes.
- Click **OK**.

3. The Class properties window displays since the **Open Properties Dialog** on Creation was checked in *Figure 5.17*. (If it does not display, right-click **{your table}** and select **Properties**).
4. From the **General** tab, select **MySql** from the Database list as shown in *Figure 5.18*. Click **Apply**.

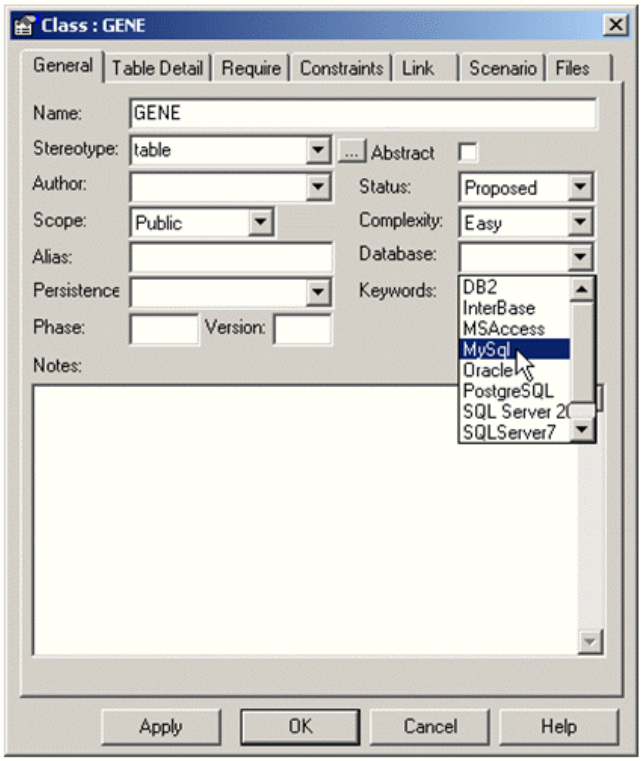


Figure 5.18 Select a Database

Creating Dependencies

Next, create dependencies from tables to classes. Mappings between tables and classes need to be defined explicitly using dependency associations. Dependency associations are another kind of association in UML and are represented graphically as dashed arrows.

From the Enterprise Architect (EA) Project View, navigate to the mapping diagram from the **Views > Logical View > Data Model** package. The mapping diagram displays tables on the left and classes on the right as shown in *Figure 5.19*. Each table is connected to a class by a dependency association. The dependency goes from table to class because (in this approach) the structure of the database depends on the structure of the object model.

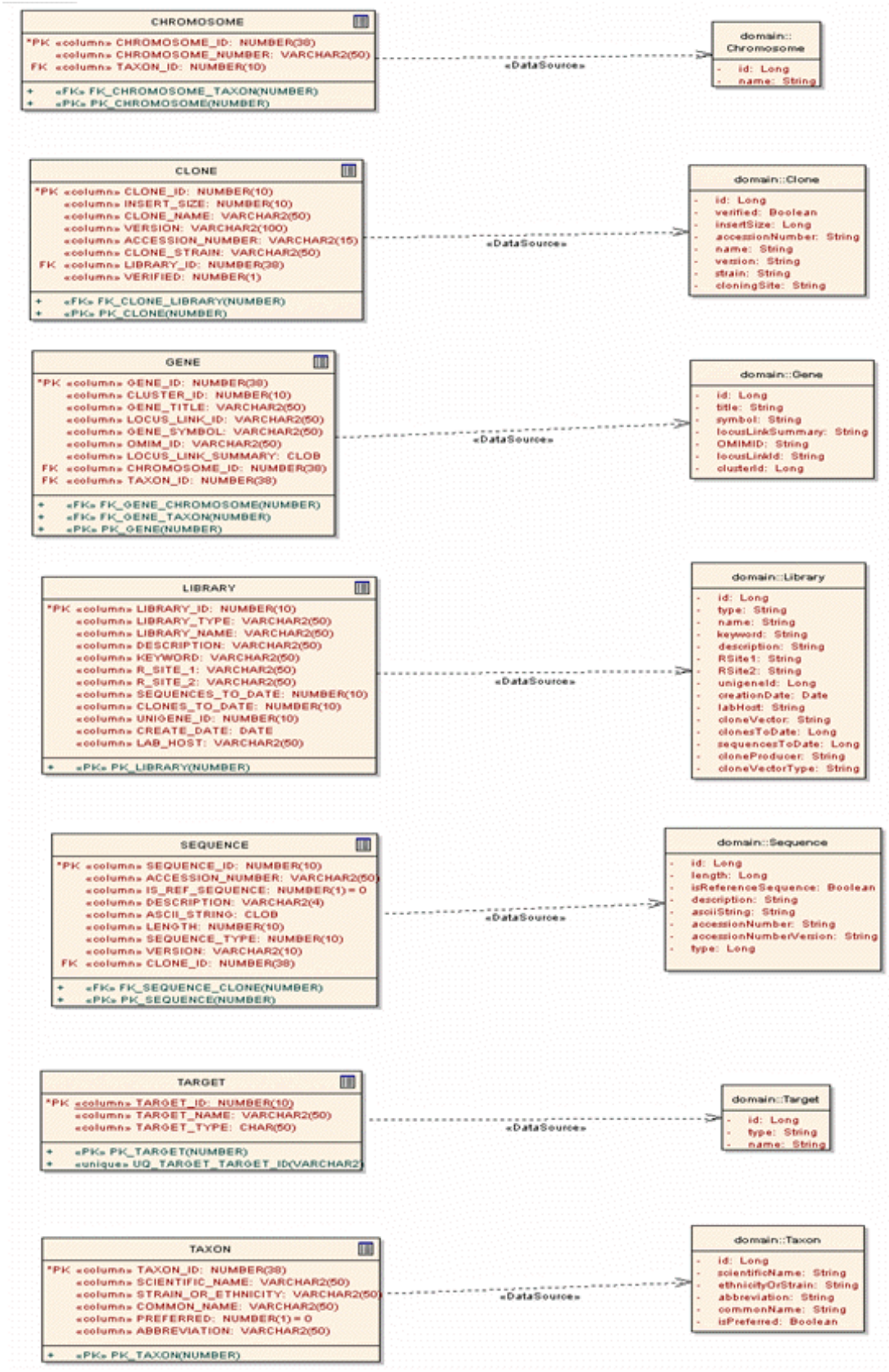


Figure 5.19 Table to Class Mapping diagram

Notice that each dependency has a **<<Data Source>>** label. This kind of label is called a stereotype in UML. Stereotypes provide a way of extending UML to include non-standard type constructs. The DataSource stereotype, in effect, creates a special class of dependencies. Each dependency labeled with **<<DataSource>>** belongs to that class. Most important is that the SDK transformers used in this example need the dependencies that map between tables and classes to be stereotyped this way so that they can be differentiated from other dependencies that could exist in the model.

Perform the following steps to create dependencies from a table to a class.

1. From **Project View > Logical View,** right-click Data Model and select **New Diagram**. The New Diagram dialog displays.
2. In the New Diagram dialog, enter the **Name** for the new mapping diagram, leave the **Structural Type** selected as Class and click **OK**.
3. Select the {mapping diagram} tab just created from the bottom of the EA page.
4. Drag and drop a domain object from the **Project View > Logical View > Logical Model** and its corresponding data model object from **Project View > Logical View > Data Model** into the diagram.
5. Select **Link > Dependency** from the EA tools bar, then drag and drop from the table to the object to create a dependency link.
6. Right-click on the dependency link and select **Dependency Properties** to display the Dependency Properties dialog box.
7. Leave the **Source**, **Target**, and **Direction** fields as is and enter ***`DataSource`*** in the Stereotype field as shown in *Figure 5.20*. Click **OK**.

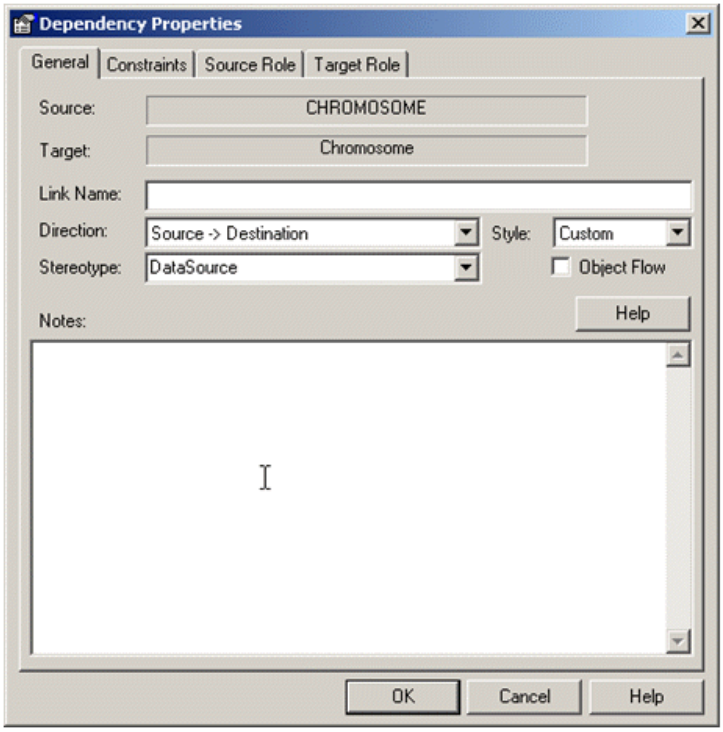


Figure 5.20 Dependency Properties Dialog

8. Create dependency links for each table and class pair.

Create Attribute to Column Mapping

This section describes how to map logical model class attributes to data model table columns.

Perform the following steps to map an attribute to a column.

1. From Project View in EA, right-click on a {**table**} and select **Attributes**. The {table} Attributes dialog box displays.
2. From the **General** tab in the {table} Attributes dialog box, click **New**. Enter the attribute information as shown in *Figure 5.21* and described in a bulleted list below *Figure 5.21*.

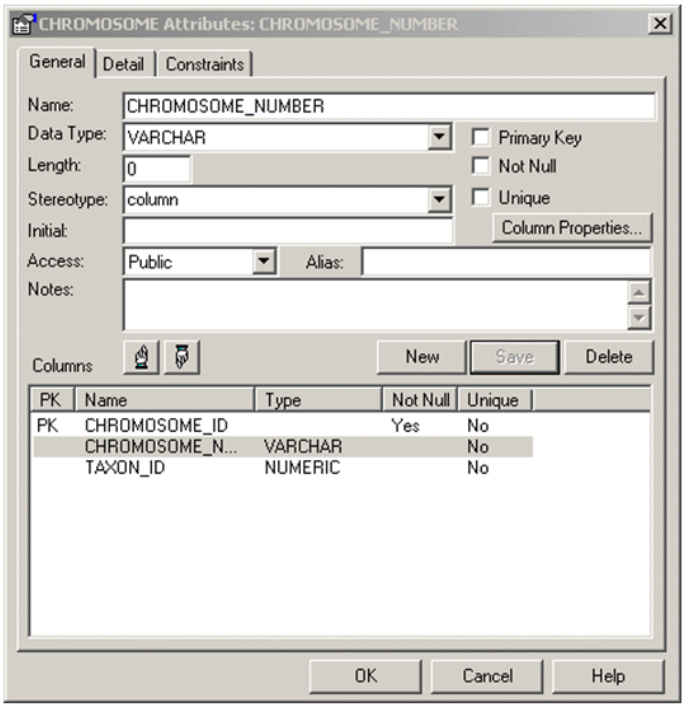


Figure 5.21 Table Attributes Dialog Box

   - Enter the **Name** of the attribute. Naming conventions for the attribute names dictate all capitalized letters and an underscore between names.
   - Select the appropriate **Data Type** from the list. Note that you must select the type of database as shown in *Figure 5.18* to populate the data type list.
   - Select the **Primary Key**, **Not Null**, and **Unique** check boxes as appropriate.
   - Click **Save**. The attribute information displays as shown in *Figure 5.21*.
3. From the Project View select the attribute just added (for example, **GENE_ID**) and select the **Tagged Values** tab from the bottom of the dialog box. You can also display the **Tagged Values** tab by using the shortcut key CTRL+SHIFT+6.

Note: Once the column is created, you must explicitly indicate what class attribute maps to it.[1] You must label the column with the fully qualified name of the associated class attribute. A UML tagged value is used for this purpose. A tagged value is a UML con-

1. While it would be convenient if the same approach could be used to map attributes to columns as was used to map classes to tables, most UML modeling tools do not support creating dependencies (or any other associations) among attributes. Therefore, mapped-attributes tags are used.

struct that represents a name-value pair and can be attached to anything in a UML model. Tagged values provide a way of adding arbitrary (non-standard) information to a UML model.[1] The SDK transformers in this example use tagged values to map attributes to columns.[2]

4. Enter the tag/value pairs as shown in *Figure 5.22* by clicking the new tag icon and described in a bulleted list below *Figure 5.22*.

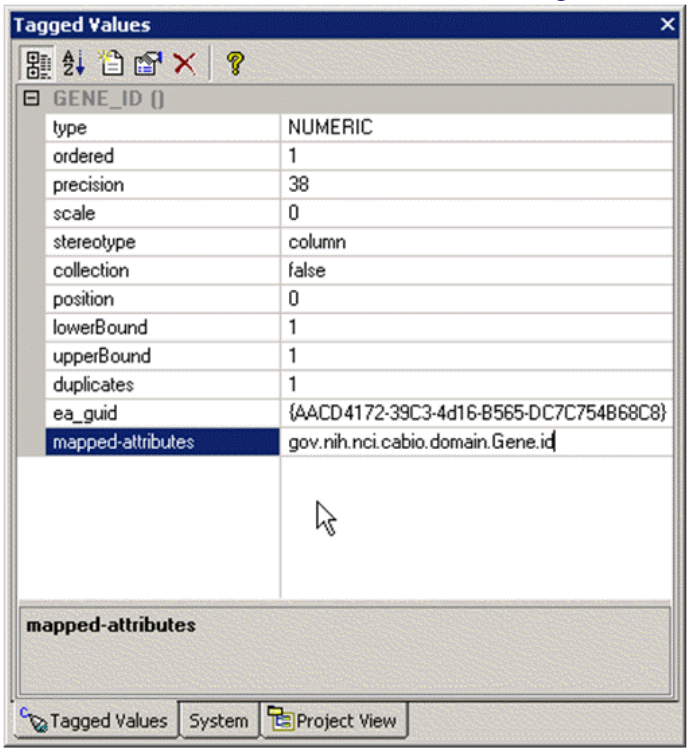


Figure 5.22 Mapped-attributes Tagged Value

   - Enter ***`mapped-attributes`*** in the first field.
   - Enter the fully-qualified name of the associated class attribute in the second field. For example, the `gov.nih.nci.cabio.domain.Gene.id` attribute maps to the GENE_ID column in the caBIO model as shown in *Figure 5.22*.

   Note: The name "**mapped-attributes**" (notice the plural) is used because it is possible that more than one attribute, possibly from different classes, could be mapped to the same column. In such a case, the value of the mapped-attributes tagged value should be a semi-colon separated list of fully qualified attribute names.

5. Repeat the above steps to add additional columns and map the column to its corresponding attribute.
6. When all the information has been added click **OK**.

1. A tagged value is often used by UML modeling tools to store tool-specific information.
2. Database columns of datatype CLOB in a data model can only be mapped to attributes of datatype `java.lang.String` in an object model.

Creating Association to Relation(s) Mapping

This section describes how to map associations that are defined in the object model to relations that need to be defined in the data model (relational model).

As previously discussed in *Creating Relationships between Classes* in Step 5 on page 46, a multiplicity must be specified for each end of an association in a class diagram. An association should be classified by the multiplicity that is specified at each association end. This classification is referred to as the cardinality of the association. The following cardinalities are possible in an object model:

- **one-to-one** - the upper bound of multiplicity range on both ends is one
- **one-to-many** - the upper bound on one side is one and the other end is unbounded
- **many-to-many** - neither end is bounded

Physical relational models (data models) support only the one-to-one and one-to-many cardinalities. Therefore, many-to-many associations must be mapped to two one-to-many associations (relations).

In a relational model, a relation between two records exists when the value of a field in one record matches the value of a field in another record. These matching fields are called keys. When designing a database schema where relations will exist between tables, the designer usually explicitly defines which columns represent key fields.

The most common (and best) way to define a relation from one record to another is to specify that the field of one record will contain a key value that is the unique identifier of another record. A column that contains such values is called a *foreign key* column. A column that contains unique identifier values is called a primary key column.

To map an object model to a relational model, you must specify how the associations between classes are mapped to the foreign key/primary key relations between tables. This process is straightforward and is described by cardinality.

Creating One-to-One Mappings

Perform the following steps to create a one-to-one mapping between tables.

1. Define primary keys in both tables (follow the step-by-step procedures in *Creating Unique Primary Keys* on page 58).
2. Define a foreign key in one of the tables (follow the step-by-step procedures in *Creating Foreign Keys* on page 59).
3. Place a unique constraint on the value of the foreign key column.
4. If the association is mandatory (that is, if the lower bound of the multiplicity range on one or both sides is one), then a NOT NULL constraint should be placed on the foreign key column.

Note: You could implement one-to-one associations using a primary key relation, where the primary key of one record matches the primary key of another record. However, the approach used here is what is expected by the SDK transformers used in this example.

Creating One-to-Many Mappings

Perform the following steps to create a one-to-many mapping between tables.

1. Define primary keys in both tables (follow the step-by-step procedures in *Creating Unique Primary Keys* on page 58).
2. Define a foreign key in the table that represents the class on the many-side of the association (follow the step-by-step procedures in *Creating Foreign Keys* on page 59).

Creating Many-to-Many Mappings

Perform the following steps to create a many-to-many mapping between tables.

1. Define primary keys in both tables (follow the step-by-step procedures in *Creating Unique Primary Keys* on page 58).
2. Define a third table (this is called the correlation table).
3. Define two foreign keys, one for each primary key (follow the step-by-step procedures in *Creating Foreign Keys* on page 59).

Creating Inheritance Mappings

Perform the following steps to implement inheritance mappings between tables.

1. Create one Class element for the Parent (super-class) (follow the step-by-step procedures in *Creating a New Element (Class)* on page 39).
2. Create one Class element for each of the Child (sub-class) classes (follow the step-by-step procedures in *Creating a New Element (Class)* on page 39). Note: Do not create an "id" attribute in any of the Child classes.
3. Create one table for the Parent (super-class) class (follow the step-by-step procedures in *Creating Data Models (Tables)* on page 50).
4. Create one table for each of the Child (sub-class) classes (follow the step-by-step procedures in *Creating Data Models (Tables)* on page 50).
5. Follow the step-by-step procedures in *Creating Data Models (Tables)* on page 50.
6. Define a primary key (PK) in the Parent (super-class) table (follow the step-by-step procedures in *Creating Unique Primary Keys* on page 58).
7. Define a primary key (PK) in the Child (sub-class) table (follow the step-by-step procedures in *Creating Unique Primary Keys* on page 58).
8. The primary key of the child table should also be the foreign key to the parent table.

   Note: This implies that all children have the same primary and foreign key definition and relationship. However, the value of the primary key should never be duplicated in any of the child tables.
9. Add a "mapped-attributes" tagged value to the primary key defined in the Child table, such as `fully.qualified.name.Child.id` (e.g., `gov.nih.nci.cabio.domain.Gene.id`).

   Note: This step must be performed even though the Child table does not have an "id" attribute.

Creating Unique Primary Keys

Perform the following steps to create a unique primary key for your table.

1. From the Property View menu, right-click your **{table}** and select **Properties**.
2. Select the **Table Detail** tab and click **Columns/Attributes** as shown in *Figure 5.23*.

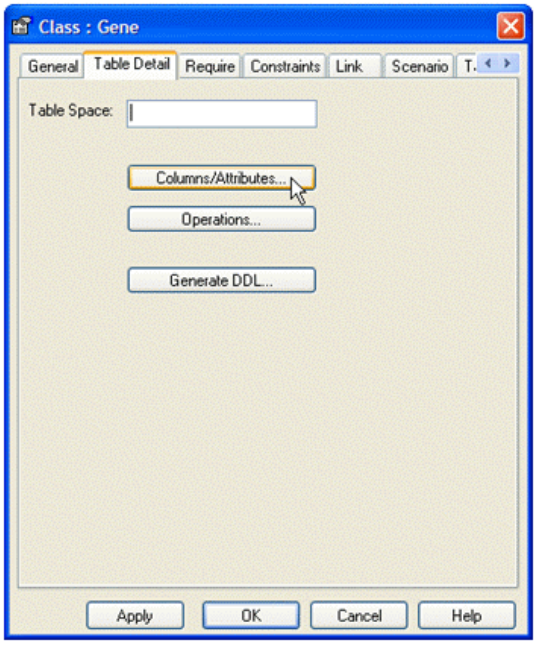


Figure 5.23 Table Detail Dialog Box

3. The {table} Columns dialog box displays as shown in *Figure 5.24*. Select the **{attribute name}** to be your primary key (for example, GENE_ID). Because primary keys are meant to uniquely and unambiguously identify rows, they must be both unique and non-null. Click the **Primary Key, Not Null**, and **Unique** check boxes and click **Save**.

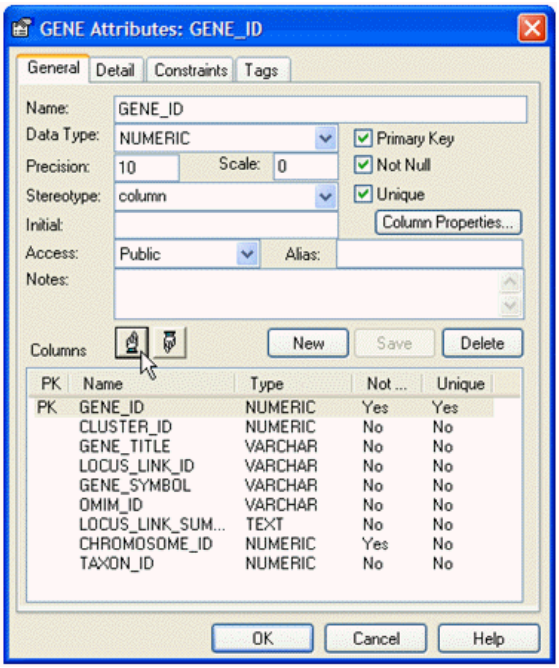


Figure 5.24 Table Column Dialog Box

Note: Each table must contain a unique primary key that should be annotated with the database specific data type (for example, NUMERIC, VARCHAR, TEXT, and so forth).

4. Click **OK** to set your selections.

Creating Foreign Keys

A foreign key (FK) is a collection of columns (attributes) that enforce a relationship to a Primary Key in another table.

Perform the following steps to create foreign keys for your table.

1. From the Data Model diagram, right-click on the link between two tables (for example, **GENE** and **TAXON**) and select **Foreign Keys**.
2. The Foreign Key Constraint dialog box displays similar to *Figure 5.25*. The **Name** (of the foreign key) is automatically created. Edit this name as required.
3. Click on the column of the source table and the column of the target table that you want to link. In the example shown in Figure 48, click **TAXON_ID** from the source and click **TAXON _ID** from the target and click **Save**.

A Foreign Key is created between the source and target tables as shown by the pointer in *Figure 5.25*.

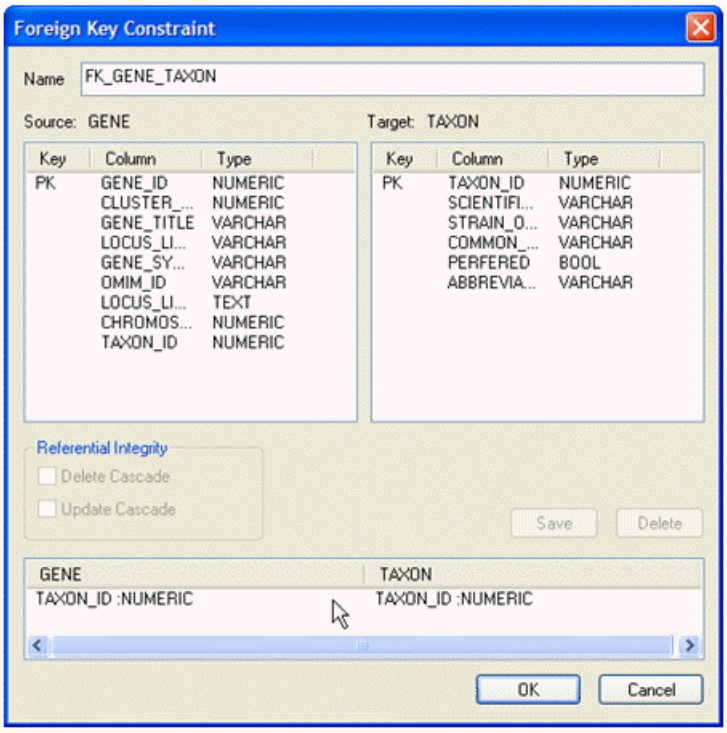


Figure 5.25 Foreign Key Constraint Dialog Box

4. Click **OK**.
5. Repeat the above steps for each required foreign key.

Note: When establishing an associative link between two tables, the table from where the link is drawn is labeled as the source and the table to which the link is drawn is labeled as the target. When creating a foreign key, if the column you are using as the foreign key is not the primary key of the target table you will receive an error. If this happens, you need to delete and redraw the association in the reverse direction. Also, be aware that DB2 will not allow key names to be longer than 18 characters.

Creating Correlation Tables

A correlation table is required when there is a many-to-many relationship between two tables. There are two types of tables in a typical data model: 1) an object table and 2) a correlation table linking two object tables. An object table corresponds directly to an object in the object model. In our example, the Gene object requires a GENE table (as shown in *Figure 5.26*) and the Sequence object requires a corresponding SEQUENCE table (as shown in *Figure 5.27*).

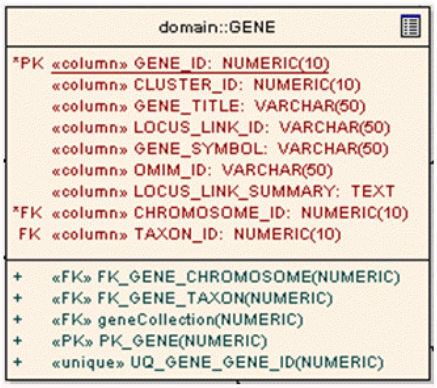


Figure 5.26 GENE data model object table

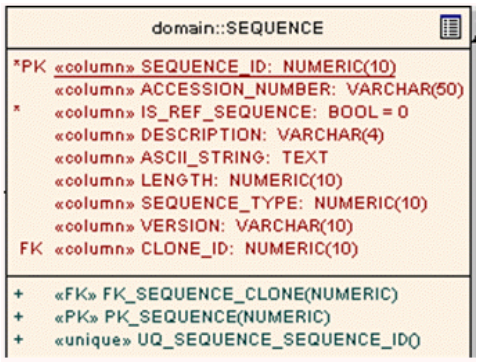


Figure 5.27 SEQUENCE data model object table

A correlation table is required when where there is a many-to-many relationship between two tables. In our example, GENE and SEQUENCE have a many-to-many relationship, so a correlation table GENE_SEQUENCE is required as shown in *Figure 5.28*

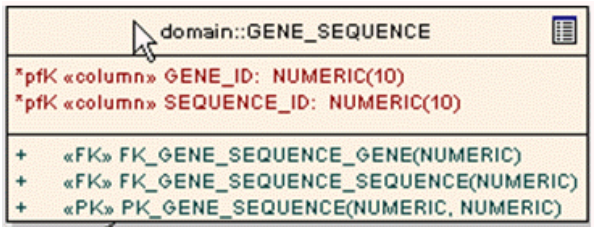


Figure 5.28 GENE_SEQUENCE correlation table

Perform the following steps to create correlation tables.

1. Create a correlation table and name it with the name of the two tables you will be linking (for example, create GENE_SEQUENCE as shown in *Figure 5.28*).
2. Add two columns, one for each primary key you need to link (for example, add GENE_ID and SEQUENCE_ID as shown in *Figure 5.28*).
3. Create two foreign keys linking the correlation table to the primary tables as described in *Creating Foreign Keys* on page 59.

Explicitly Mapping Associations to Relations

Once the primary keys, foreign keys, and correlation tables have been created, you must create the following four tagged values to explicitly map associations to relations (each is described below):

1. implements-association
2. correlation-table
3. implements-association
4. inverse-of

Adding Tagged Value implements-association

First, an "implements-association" tagged value must be added to each foreign key column.

Perform the following steps to enter the tagged values.

1. From the **Project View**, select the required field (for example, the **foreign key** column) and display the Tagged Values dialog box by using the shortcut key CTRL+SHIFT+6.
2. Enter the tag/value pairs by clicking the new tag icon and entering the following information:
   - Enter ***`implements-association`*** in the first field.
   - Enter the fully qualified name of the association end that is implemented by the foreign key in the second field. For example, in the caBIO object model as shown in *Figure 5.14*, there is a one-to-many association from Taxon to Chromosome. Therefore, the CHROMOSOME table contains a TAXON_ID foreign key column. The "implements-association" tagged value for that column is `gov.nih.nci.cabio.domain.Chromosome.taxon`.

Adding Tagged Value correlation-table

Second, a "correlation-table" tagged value must be added to each many-to-many association that is defined in the object model.

Perform the following steps to enter the tagged values.

1. In the class diagram, select the link between two objects that have a many-to-many relationship and display the Tagged Values dialog box by using the shortcut key CTRL+SHIFT+6.
2. Enter the tag/value pairs by clicking the new tag icon and entering the following information.
   - Enter ***`correlation-table`*** in the first field.
   - Enter the fully qualified name of the correlation table that was used to decompose the association. For example, in the caBIO object model, there is a many-to-many association between Gene and Sequence. That association has a "correlation-table" tagged value which is "GENE_SEQUENCE", the name of the correlation table.

Adding Tagged Value implements-association

Third, each foreign key in each correlation table must be given an "implements-association" tagged value that indicates what association end it implements.

Perform the following steps to enter the tagged values.

1. From the Project View, select the required field and display the Tagged Values dialog box by using the shortcut key CTRL+SHIFT+6.
2. Enter the tag/value pairs by clicking the new tag icon and entering the following information.
   - Enter ***`implements-association`*** in the first field.

- Enter what association end it implements. For example, the GENE_ID column in GENE_SEQUENCE has an "implements-association" tagged value which is "gov.nih.nci.cabio.domain.Sequence.geneCollection".

Adding Tagged Value inverse-of

Finally, for each bi-directional, many-to-many association, one association end must be specified as the "inverse-of" end. To do this, simply create an "inverse-of" tagged value on one of the foreign key columns of each of the correlation tables and set its value to the fully qualified name of the other association end.

Perform the following steps to enter the tagged values.

1. From the Project View, select the required field and display the Tagged Values dialog box by using the shortcut key CTRL+SHIFT+6.
2. Enter the tag/value pairs by clicking the new tag icon and entering the following information.
   - Enter ***inverse-of*** in the first field for one of the foreign key columns.
   - Enter the fully qualified name of the other association end. For example, the GENE_ID column in GENE_SEQUENCE has the value `gov.nih.nci.cabio.domain.Gene.sequenceCollection`.

Creating a Sequence Diagram

Creating sequence diagrams is an optional step in creating a caCORE-like system since they are not used for code generation. A sequence diagram displays object interactions in terms of an exchange of messages arranged in a time sequence. It models the flow of logic within your system visually, validating the logic of a usage scenario. Using a sequence diagram, bottlenecks can be detected within your object-oriented design and complex classes can be identified. See *Sequence Diagrams* on page 198 for more information. Step-by-step procedures are not included in this guide to produce sequence diagrams because they are not required to use the SDK.

Generating XMI

This section provides the procedures to export the UML model diagram into XML Metadata Interchange (XMI) format. The caCORE SDK uses a UML version 1.3 model as a basis for generating source code and other artifacts.[1]

To use a UML model, the model must be stored in an XMI format. XMI is a standard interchange format for UML models, and many UML modeling tools (including EA), can export models as (more or less) valid XMI[2].

Perform the following steps to export the UML model into XMI using EA.

1. From Project View, right-click your **Logical View** package, and select **Import/Export > Export package to XML file**. The Export Package to XML dialog box displays.

1. In actual practice, the code generator can use any instance of a Meta-Object Facility (MOF) model.
2. The XMI file must be stored in a format that the NetBeans Metadata Repository (MDR) XMI reader can parse.

2. Enter the following information as shown in *Figure 5.29* and listed in bulleted items below *Figure 5.29*.

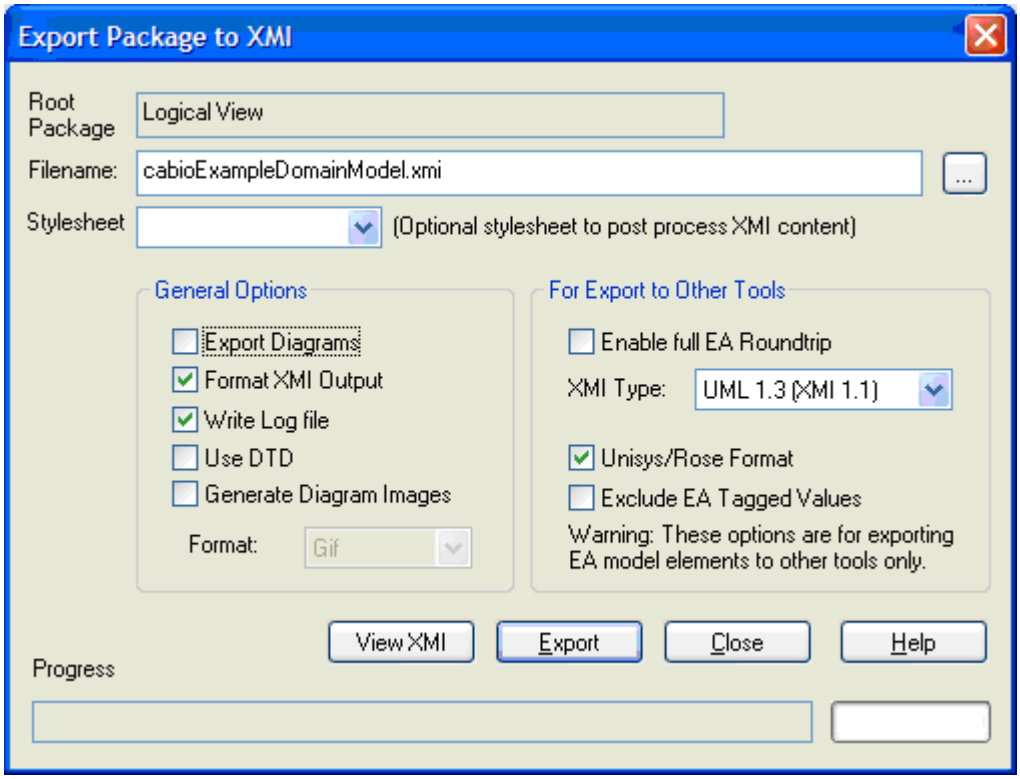


Figure 5.29 Export Package to XML Options

   - **Filename** – Enter the filename for the XMI file. This file must contain no spaces and must be located in `{home_directory}/models/xmi` directory. An example filename in Windows is `C:\cacoretoolkit\models\xmi\cabioExampleModel.xmi`.
   - **General Options** – Select **Format XML Output** and **Write Log File**.
   - **For Export to Other Tools** – Select **Unisys/Rose Format** only.

3. Click **Export**. The status displays in the Progress window while processing with a completion message when done.

Notes: For version 3.2, the caCORE SDK code generation process requires the model to be exported in a different format than that used for exporting to caAdapter and SIW. The code generator still uses the format used in 3.1. However, caAdapter and SIW use the native EA Roundtrip file export, making it easier to work with the model in these two tools and enabling reimport into EA for modifications. This is accomplished with a new tool, the XMI Handler. It's architecture allows it to accept 'plug-ins' so that additional XMI file formats can be accepted. In the next release, code generator will make use of this new capability, eliminating the need to export the model twice if using caCORE SDK to generate code. *Figure 5.29* and *Figure 6.18* show the settings required for caCORE SDK code generation. *Figure 6.5* and *Figure 7.9* show the settings used for caAdapter and SIW.

Also note, the XMI produced by EA is not a valid NetBeans Metadata Repository MDR XMI file since it contains some specific EA characteristics. The SDK includes an XMI preprocessor which can be invoked by calling an Ant task, fix-xmi, to make some minor

modifications to the structure of a given XMI file before semantic connection and code generation begin. If you use a modeling tool other than EA, you will have to make sure you have a valid MDR XMI file.

Generating Data Definition Language

Data Definition Language (DDL) produces Structured Query Language (SQL) commands that can be used to build the underlying database. It hides the implementation details of the database schemes from the users. Many UML modeling tools, including EA, can create DDL scripts from their models. If you did not create a data model, then you do not perform this step (because not creating a data model implies that you already have a database).

Perform the following steps to create Data Definition Language (DDL) scripts using EA.

1. From Project View, right-click on the directory containing the data model, select **Code Engineering > Generate DDL**. The Generate DDL dialog box displays similar to *Figure 5.30*.
2. Enter the information as shown in *Figure 5.30* and listed in the bullets below *Figure 5.30*.

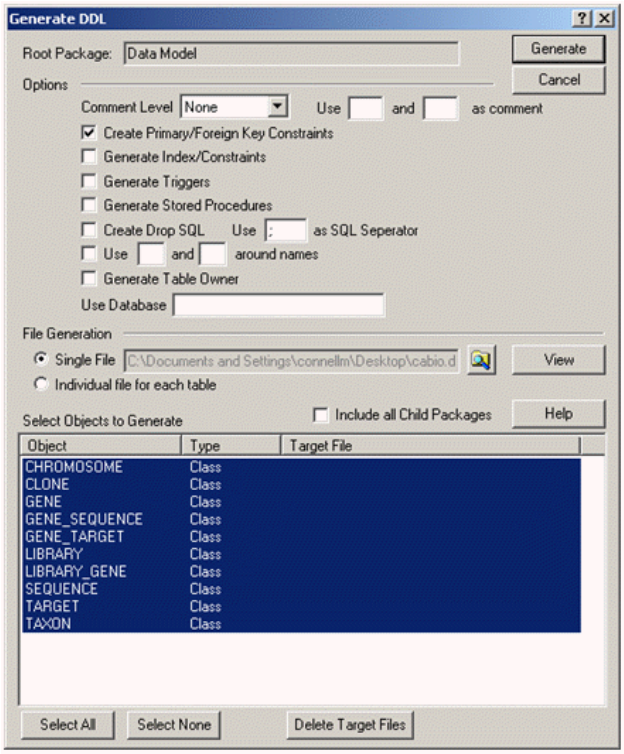


Figure 5.30 DDL Generation Dialog

- Under **Options**, select **Create Primary/Foreign Key Constraints**.
- Select **Single File**, and specify the filename and output directory.
- Select all objects to generate by clicking **Select All**.
- Click **Generate**.

3. The Batch generation dialog indicates Generation complete. Click **Close**.

CHAPTER

6

caADAPTER MODEL MAPPING SERVICE

This chapter describes how to install and use caAdapter, a tool that facilitates object to database mapping.

Topics in this chapter include:

Overview

The caAdapter model mapping service for the caCORE 3.2 SDK takes advantage of the caAdapter mapping infrastructure to facilitate object to database mapping. The model mapping service requires an XMI file (with full EA roundtrip capability) that includes a data model and object model as inputs. The service module loads all models into the tool. Object to database mapping can be done by dragging object model elements and dropping them onto the target data model elements. Once mapping is complete, caAdapter adds SDK-required tagged values into a new XMI file. After reporting the newly tagged XMI into EA and exporting an XMI 1.1-compatible XMI file, the caCORE SDK can perform all code generation tasks.

caAdapter Minimal System Requirements

Minimal system requirements for the caAdapter consist of:

- Internet connection
- Tested platforms

The caAdapter has been tested on the platforms shown in Table *6.1*.

	Windows Server	***Linux Server***
Model	DELL Optiplex GX270	HP Proliant ML 330
CPU	1 x Intel® Pentium™ 2.8 GHz	1 x Intel® Xeon™ Processor 2.80 GHz
Memory	1.3 GB	4 GB
Local Disk	System = 40 GB	System = 2 x 36 GB (RAID 1) Data = 2 x 146 (RAID 1)
Network	100mb / full duplex	100mb / full duplex
OS	Windows 2000 Professional	Red Hat Linux ES 3
Resolution (Recommended)	1280 x 1024 1024 x 768	1600x1200 1400x1050 1280x960 1280x864
Resolution (NOT Recommended)	800 x 600 640 x 480	800 x 600 640 x 480

Table 6.1 Platform testing environment for caAdapter

Java is not included with caAdapter and must be downloaded and installed. If you are installing source code, you must also download and install Ant, which is not included with caAdapter. Table *6.2* contains information for downloading the versions of java and Ant that conform with the NCICB technology stack.

Software Name	***Version***	***Description/URL***	***Example Directory***
Java 2 Platform Enterprise Edition (J2EE) or Standard Edition (J2SE)	1.5.0_06	The J2SE Software Development Kit (SDK) supports creating J2SE applications http://java.sun.com/j2se/	If your root directory in Windows is C:\, then install to the C:\jdk1.5.0_06 Java home directory
Ant	1.6.2	Apache Ant is a Java-based build tool http://ant.apache.org/	If your root directory in Windows is C:\, then install to C:\apache-ant-1.6.2 Ant home directory

Table 6.2 Required software for all caAdapter distributions

Perform the following steps in Windows to verify the JAVA_HOME and ANT_HOME environment variables are set and add them to your PATH. Right click on **My Computer**, select **Properties**, select the **Advanced** tab, and click **Environmental Variables**.

1. JAVA_HOME and ANT_HOME must be listed in the User variables or System variables list box. To add either, click **New** below one of the boxes.
2. In the New Variable dialog box, add the Variable name and Variable value for your home directory.

   Examples:

   ```
   Variable = JAVA_HOME; Variable Value = C:\jdk1.5.0_06
   Variable = ANT_HOME; Variable Value = C:\apache-ant-1.6.2
   ```
3. Find the PATH environment variable, double click it or click the **Edit** button and add%JAVA_HOME%\bin to the end of its value. Add%ANT_HOME%\bin to the end of its value. Click **OK**.
4. To verify that the PATH statement listed in the Environment Variables box includes JAVA_HOME and ANT_HOME, open a command window (Start > Command Prompt). Enter `path` at the prompt and press enter to display your path. For example, `C:\jdk1.5.0_06` and `C:\apache-ant-1.6.2\bin` should be at the end of your path. If you were successful, you can run java and ant anywhere in your system.

Downloading caAdapter

To use caAdapter, download the package provided on the NCICB web site (*Figure 6.1*).

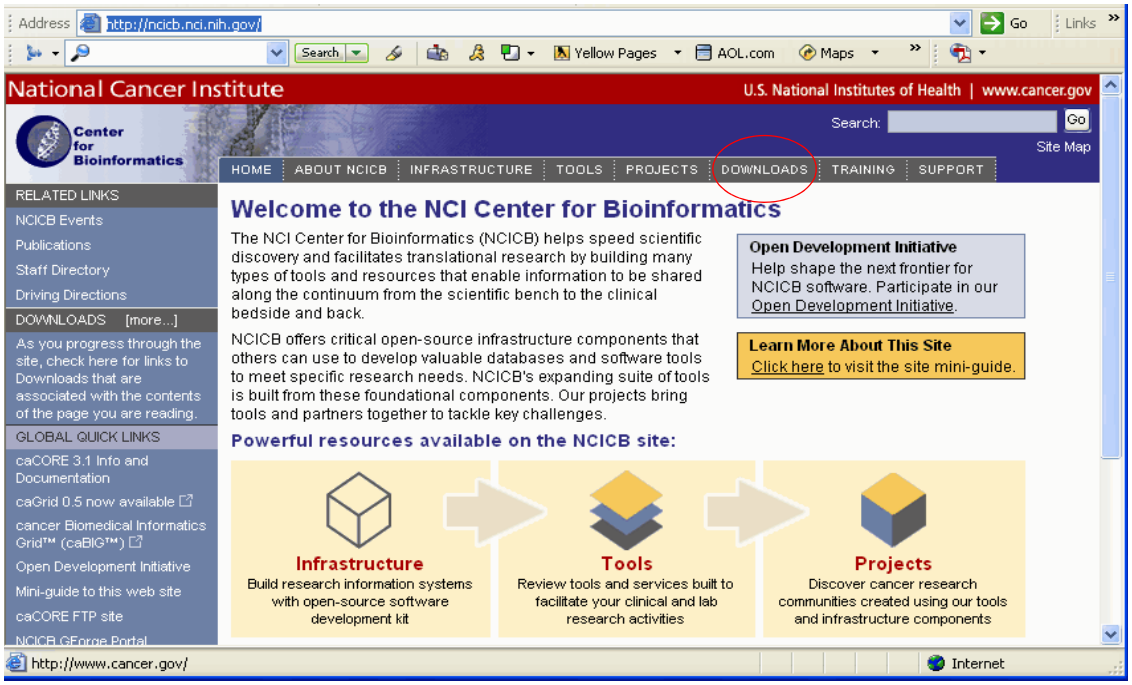


Figure 6.1 Downloads section on the NCICB website

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. Select caAdapter and extract the contents of the downloadable archive to a directory on your hard drive (for example, `c:\caAdapter` on Windows or `/usr/local/caAdapter` on Linux). Select the appropriate distribution as listed below and save it to a temporary directory on your computer (for example, C:\temp in Windows). Each type of distribution has files with (w) or without (wo) Hierarchical Message Definition (HMD) files.The extracted directories and files are listed in Table *6.3*.

Directories and Files	*Description*	*Component*
Binary Distribution	The caAdapter binary distribution file contains the binary code for caAdapter and HL7 JavaSIG software, Javadocs for caAdapter and HL7 JavaSIG code, Release Notes, HL7 message schemas, example messages, and licenses.	caadapterv3.2_bin_w.zip: Binary file with HL7 MIF files Caadapterv3.2_bin_wo.zip: Binary file without HL7 MIF files
Source Distribution	The caAdapter source distribution file contains the source java code for caAdapter and HL7 JavaSIG software, Javadocs for caAdapter and HL7 JavaSIG code, Release Notes, readme.txt, HL7 message schemas, example messages, and licenses.	caadapterv3.2_src_w.zip: Source file with HL7 MIF files caadapterv3.2_src_wo.zip: Source file without HL7 MIF files
Windows Distribution	The caAdapter windows distribution file contains binary code for caAdapter and HL7 JavaSIG software, Javadocs for caAdapter and HL7 JavaSIG code, Release Notes, readme.txt, HL7 message schemas, example messages, and licenses.	caadapterv3.2_w.ms: Windows file with HL7 MIF files caadapterv3.2_wo.msi: Windows file without HL7 MIF files

Table 6.3 Extracted directories and files in the caAdapter package

Installing caAdapter

To install caAdapter, complete the appropriate steps for your distribution.

Installing the Source and Binary Distributions

Extract the contents of the caAdapter source or binary distribution zip file to your root directory. For example, if your root directory in Windows is C:\, then your caAdapter home directory becomes C:\caadapter.

Note: The caadapter directory is created for you.

Table *6.4* contains the directory structure after installation.

Installing the Windows Distribution

Double-click on the .msi file and follow the instructions provided to complete installation. Table *6.4* contains the directory structure after installation.

Directory	*Contents*
build	Binaries (.class files) (only for source distribution and is created at runtime)
etc	Important supplementary files
images	Images used by the GUI
lib	Java libraries and dependencies
license	License and legal information
schema	HL7 v3 Schema files
src	Source code (.java files) (only for source distribution)
workingspace	Default directory where you can save project files. It contains log files and HL7 v3 XML instances. It also contains an examples directory with sample data.

Table 6.4 Directory Structure of caAdapter

Verifying Installation

Perform the appropriate processes for your distribution to ensure the installation was successful.

Verifying Binary Installation

Perform the following steps to launch the caAdapter Mapping Tool.

1. In a Command Prompt window, enter `cd {home_directory}` to go to your home directory (for example, in Windows C:\caadapter).
2. Enter `java -jar caadapter_ui.jar.`
3. The caAdapter Mapping Tool displays (*Figure 6.2*).

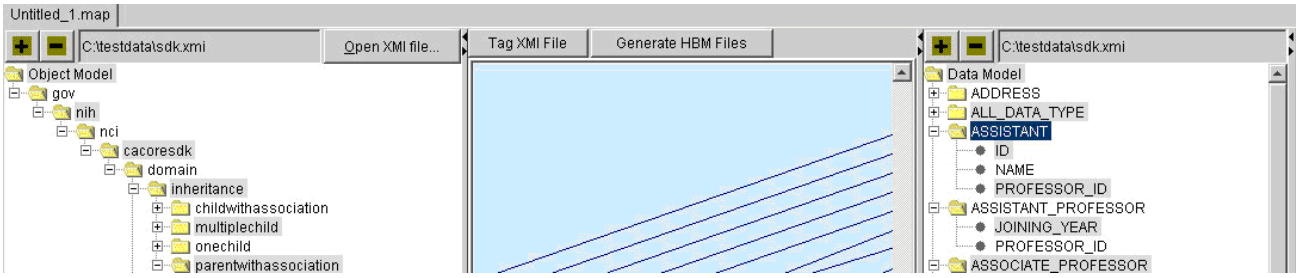


Figure 6.2 caAdapter Mapping Tool

Verifying Source Installation

Perform the following steps to launch the caAdapter Mapping Tool.

1. In a Command Prompt window, enter `cd {home_directory}` to go to your home directory (for example, in Windows C:\caadapter).
2. Enter `ant compile.`

3. Enter `ant launchui`.
4. The caAdapter Mapping Tool displays (*Figure 6.2*).

Verifying Windows Installation

Perform the following steps to launch the caAdapter Mapping Tool.

1. Navigate to the caAdapter shortcut from the Start menu and click caAdapter.
2. The caAdapter Mapping Tool displays (*Figure 6.2*).

Using caAdapter

The caAdapter model mapping service provides the following functionalities:

- Parse and load data model and object model from an XMI file
- Drag and drop mapping between an object model and a data model
- Add SDK required tags and tag values into an XMI file
- Generate Hibernate mapping file

Figure 6.3 describes the overall flow of how the caAdapter model mapping service is integrated with other components. First, an object model and data model are developed. An XMI file is then exported from Enterprise Architect (EA) and the caAdapter model mapping service loads the XMI file. Objects are mapped to tables and attributes to columns by dragging and dropping. Once completed, caAdapter directly generates a set of Hibernate (hbm) mapping files; caCORE compliant tag values can also be added into the original XMI file. The tagged XMI file can then be reimported into EA and an XMI file generated that can be used by the caCORE SDK.

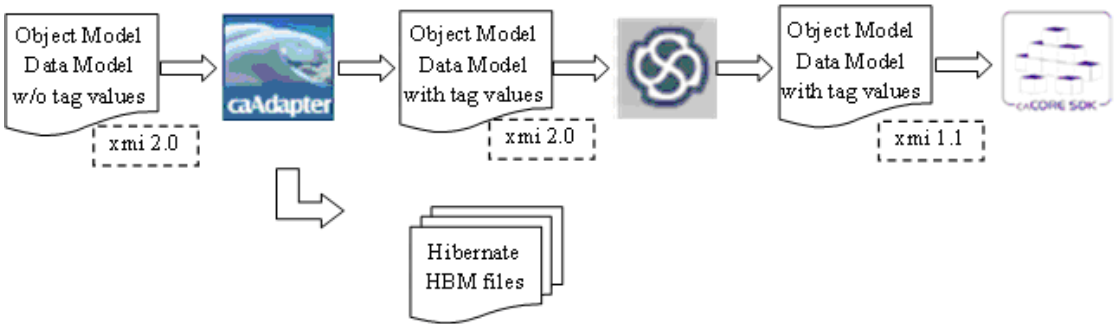


Figure 6.3 caAdapter process overview

The following sections describe each of these steps in detail.

Export XMI File from EA

Before conducting mapping between an object model and a data model through the caAdapter model mapping service, an XMI file needs to be generated from EA by using the following steps.

1. In EA, open the .eap file (Model) and right click on the Logical View element. Select **Import/Export** from the pop-up menu and then select **Export package to XMI file** on the sub menu (*Figure 6.4*).

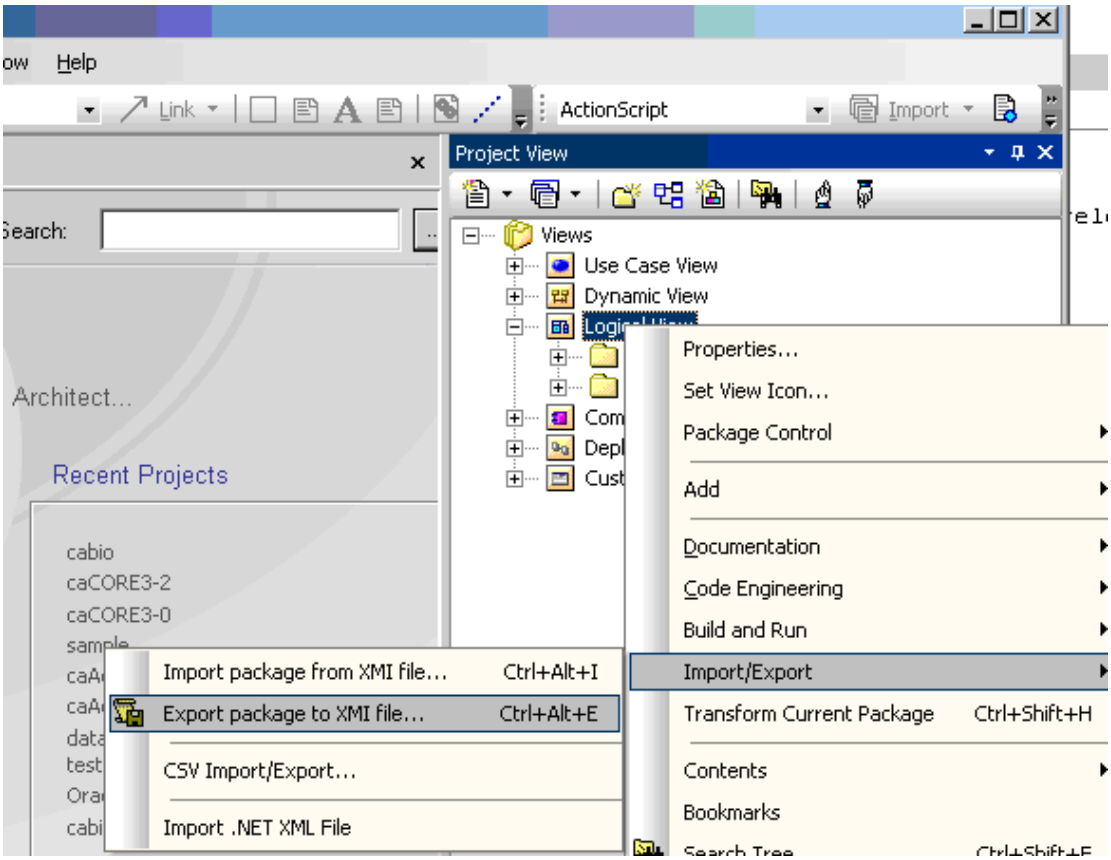


Figure 6.4 Export xmi file from EA

2. In the Export XMI File window, select **Enable Full EA Roundtrip** and then **Format XMI output**. Specify the output file name of the XMI file and then click on the **Export** button. The generated XMI file is parsed by caAdapter (*Figure 6.5*).

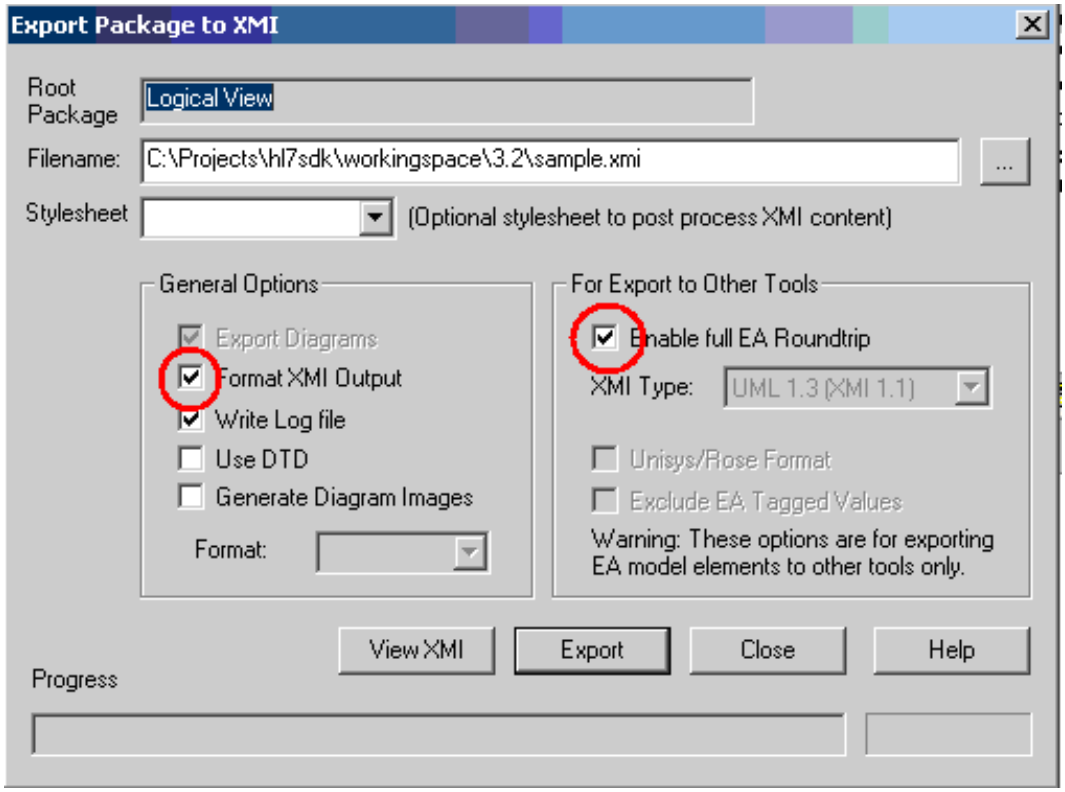


Figure 6.5 Options to export xmi file from EA

Creating an Object Model to Data Model Map Specification

Perform the following steps to create a new map specification.

1. Select **File > New > Model Mapping Service > Object Model to Data Model Map Specification** (*Figure 6.7*) to open a new mapping tab with empty source and destination panels.

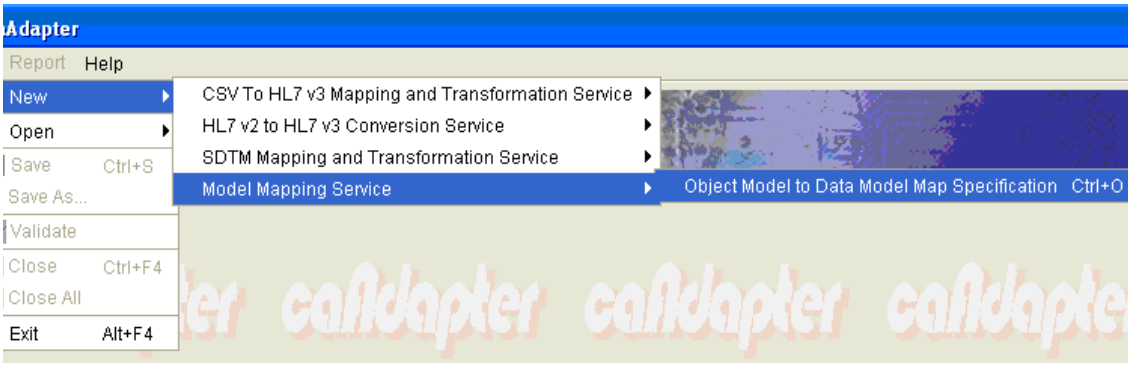


Figure 6.6 Creating an object model to data model map specification

2. Click **Open XMI File…** to display the Open XMI file … dialog box (*Figure 6.7*). Select the XMI file to start mapping an object model to a data model.

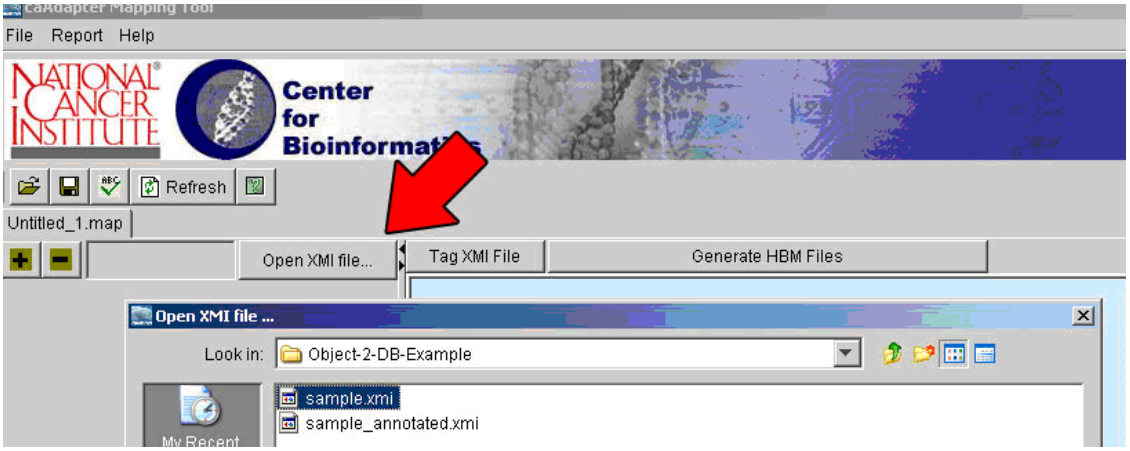


Figure 6.7 Open XMI file

Opening an Existing Object to Database Mapping Specification

Perform the following steps to open an existing map specification.

1. Select **File > Open > Object to DB Map Specification**. The **Open Map File** dialog box displays.
2. Select the map specification file and click **Open**.
3. If XMI is needed or cannot be found to open this mapping file, the **Select XMI file** dialog opens. Browse to the correct XMI file and click **Open**.

Basic Mapping

Perform the following steps to create an object to database mapping specification.

1. Select a source element from the Object Model and drag it to the appropriate target element. The cursor indicates if the source is or is not allowed to be mapped to the target element (*Figure 6.8*). Drop the source on the target element.

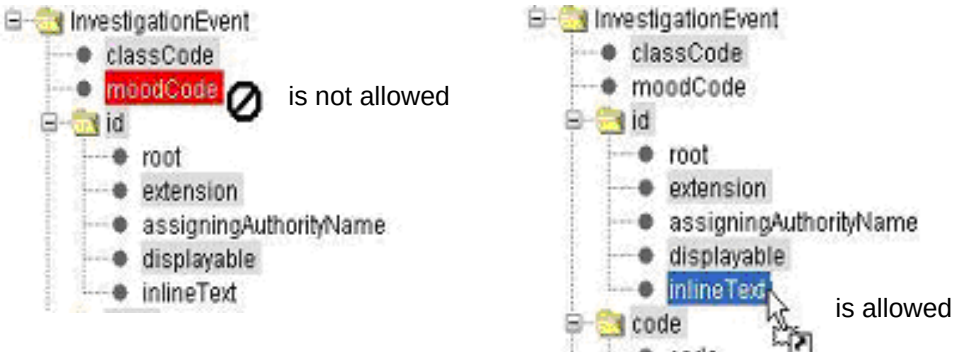


Figure 6.8 Cursor indicates whether mapping is allowed

2. Once a source field is mapped to a target element, a mapping line appears between them in the mapping panel. *Figure 6.9* shows a mapping line between

BANK in the Object Model, on the left, and BANK in the Data Model, on the right.

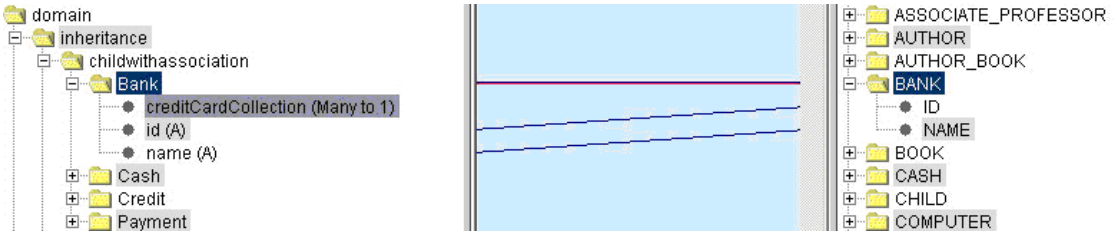


Figure 6.9 Mapping line between a source field and target element

Dependency Mapping (Object to Class)

Perform the following steps to create a dependency mapping.

1. Select a source field from the Object Model, for example BANK. Click and drag to BANK in the Data Model.
2. A mapping line between BANK in the Object Model and BANK in the Data Model should now be visible (*Figure 6.10*).

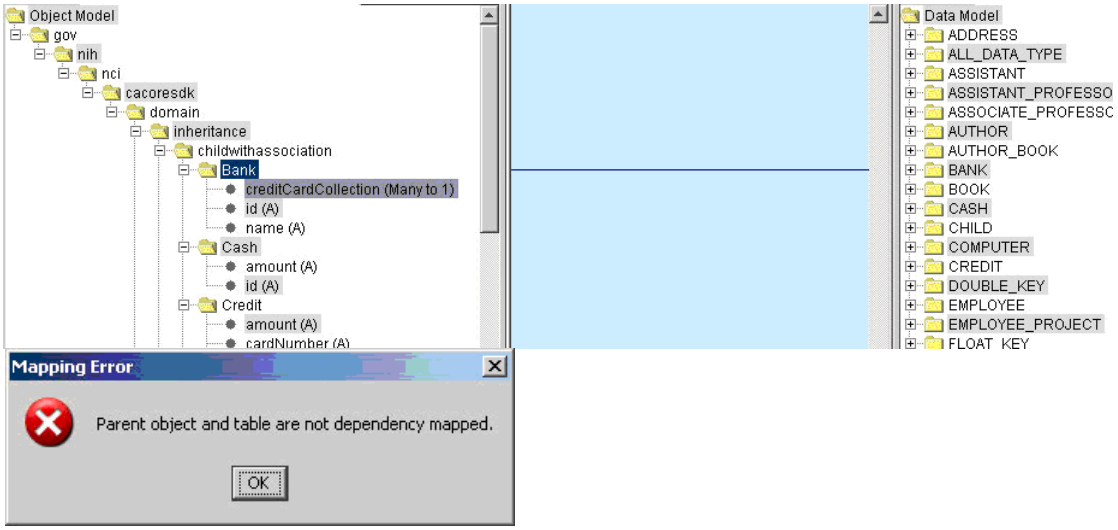


Figure 6.10 Dependency Mapping

Attribute Mapping

Perform the following steps to create an attribute mapping.

1. Select 'id (A)' from the Bank in the Object Model. Click and drag to the Data Model, BANK->ID.

2. A mapping line should be visible (*Figure 6.11*). Repeat this for 'name (A)' to BANK->NAME.

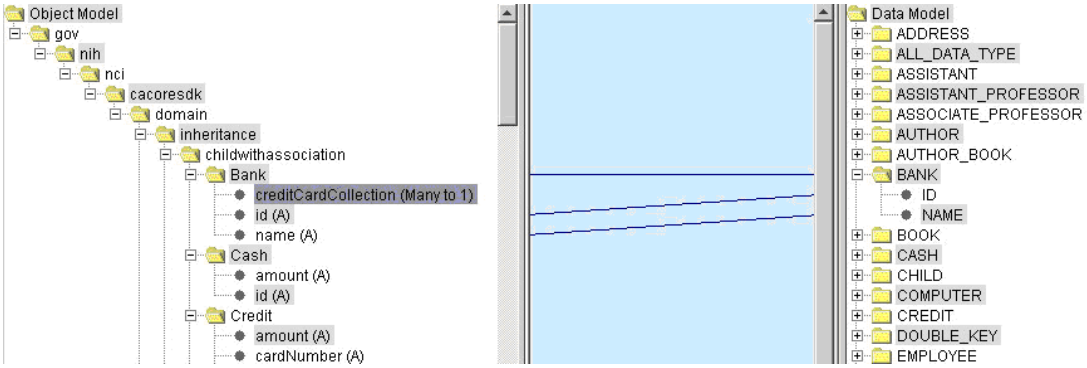


Figure 6.11 Attribute Mapping

Association Mapping

Perform the following steps to create an association mapping.

1. Dependency Map 'Target' to 'TARGET'.
2. Map 'id (A)' to TARGET -> TARGET_ID, 'name (A)' to TARGET -> TARGET_NAME, and 'type (A)' to TARGET -> TARGET_TYPE.
3. Click and drag 'geneCollection (1 to 1)' to GENE_TARGET -> GENE_ID. When complete, the final result should look like *Figure 6.12*.

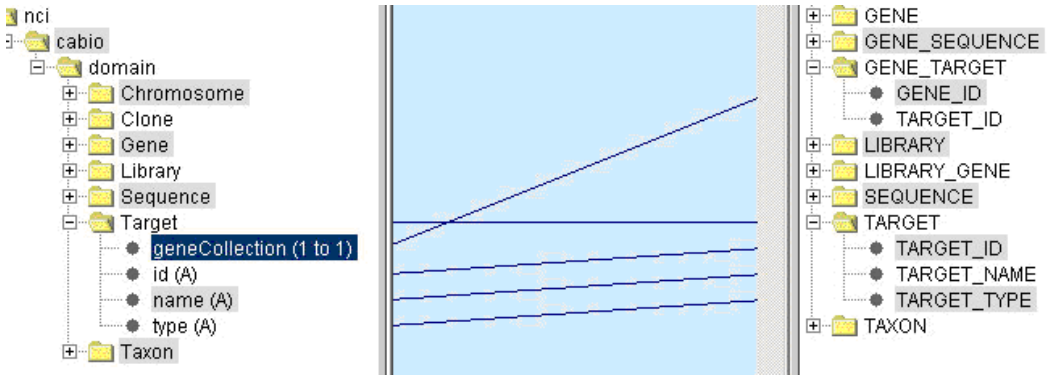


Figure 6.12 Association Mapping

Deleting Mapping Lines

Perform the following steps to delete a mapping line.

1. Select the mapping line by left clicking on it in the mapping panel. The line is highlighted red.

2. Right click on the highlighted mapping line. An option box displays with the first option being delete (*Figure 6.13*). Left click on **Delete** to delete the mapping line.

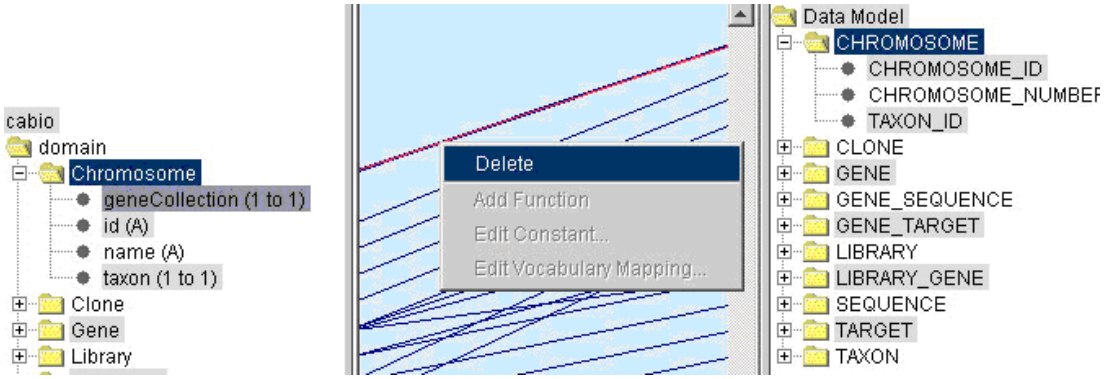


Figure 6.13 Deleting Mapping Lines

Validating Mapping Specifications

Perform the following steps to validate the object to database mapping specification.

1. Click the **Validate** button (top of *Figure 6.19*). The following message displays: "Validation process completed successfully with no message received". If there are errors in the validation process, the following message displays: "Validation process completed but received # ERRORs".

2. If there are errors the Message Dialog (bottom of *Figure 6.14*) window opens and allows examination of any messages, errors, or warnings.

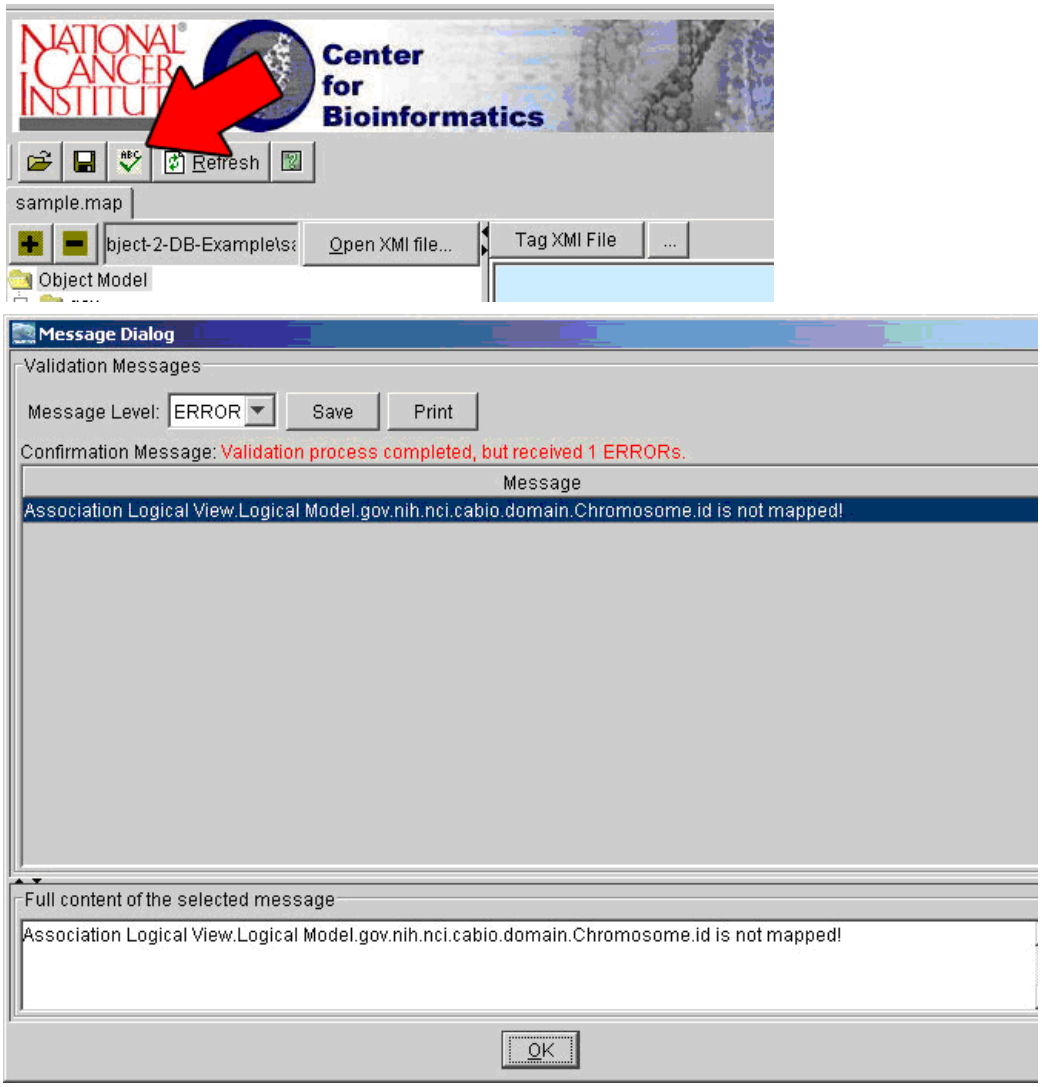


Figure 6.14 Message dialog displaying errors on validation

Saving Mapping Specifications

To save a mapping specification, select **File > Save**. The Save Complete dialog displays when completed.

Tagging XMI Files

To tag an XMI File, click the **Tag XMI File** button (*Figure 6.15*). When tagging is complete, a Tagging Complete dialog box displays.

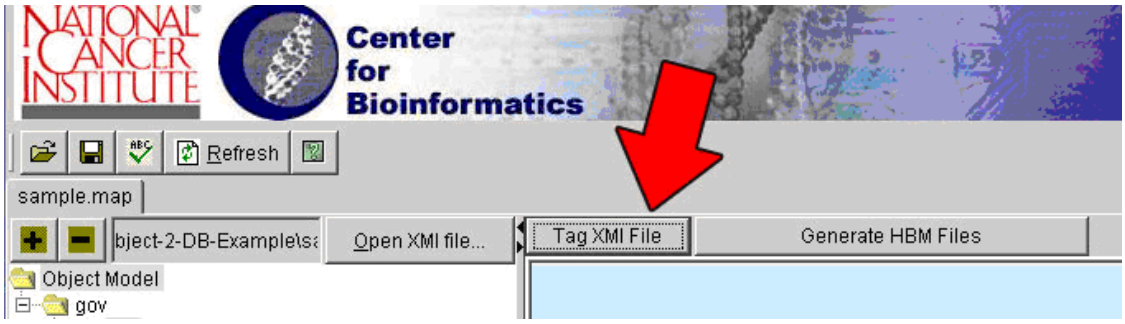


Figure 6.15 Tag XMI File

Generate XMI for caCORE SDK Integration

Use the following steps to generate an XMI file for caCORE SDK integration.

1. Reimport the XMI file to EA.
2. After the tagging is complete, the tagged XMI can be reimported back into EA and reexported for caCORE SDK integration. Create an empty project in EA without selecting any models (*Figure 6.16*).

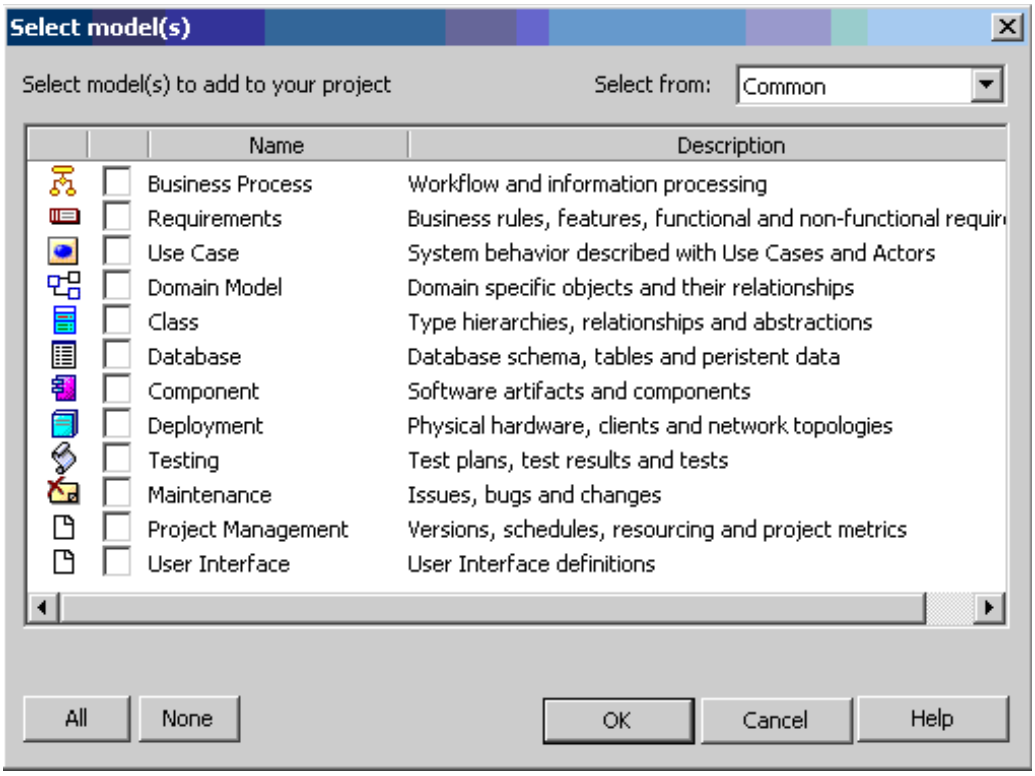


Figure 6.16 Options for newly created project in EA

3. Right-click on the Model element and select **Import model from XMI...** from the pop-up menu.

4. Select the tagged xmi file and click **Import**. The model is reimported back into EA (*Figure 6.17*).

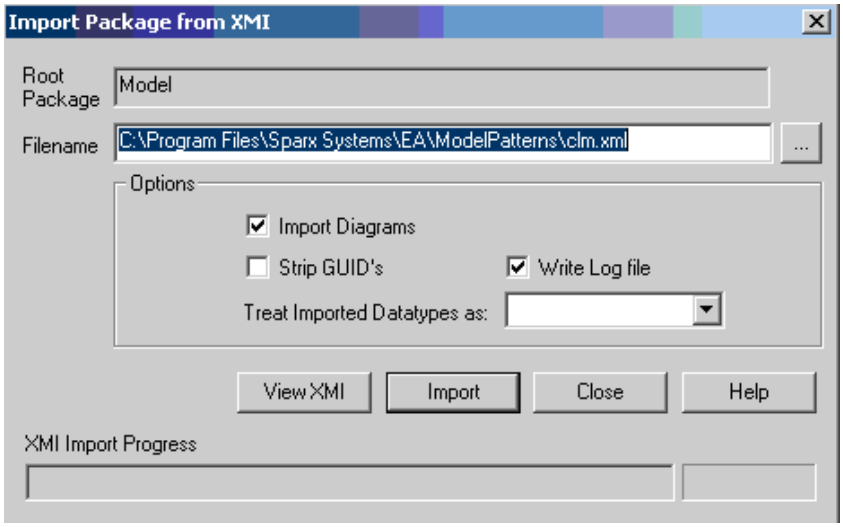


Figure 6.17 Import xmi file dialog

Export XMI File from EA for caCORE SDK Integration

Use the following steps to export the XMI file from EA for caCORE integration.

1. Once the import is complete, right click on the **Logical View** element, select **Import/Export** from the pop-up menu, and then select **Export package to XMI file** from the sub menu.
2. In the Export XMI File window, uncheck the option **Enable Full EA Roundtrip** and **Export Diagrams**. Check **Unisys/Rose Format** and **Exclude EA Tagged Values** (*Figure 6.18*).

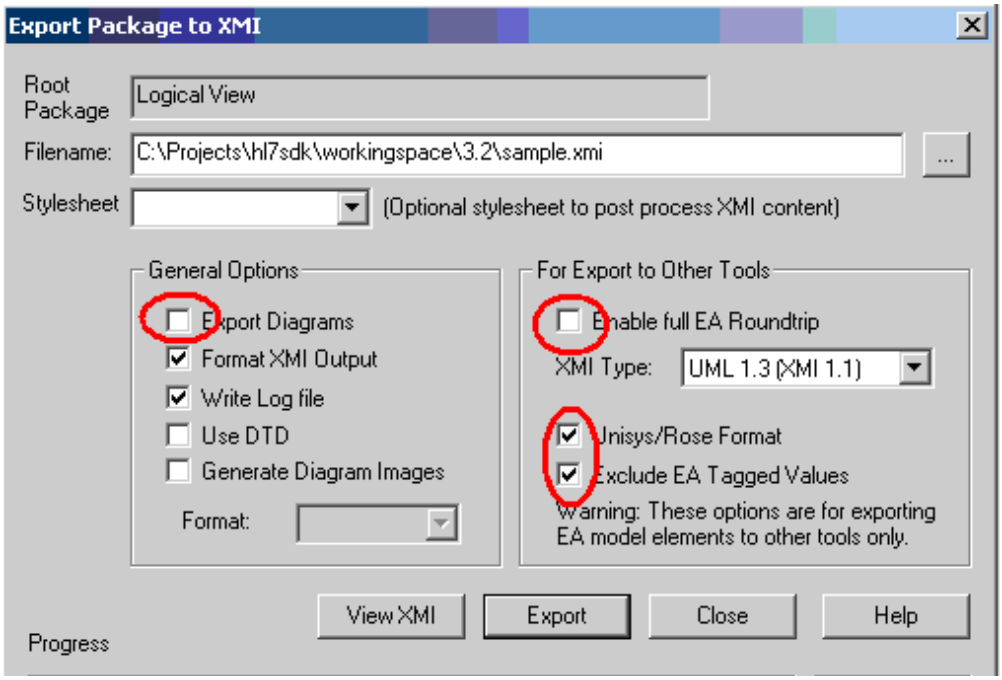


Figure 6.18 Exporting xmi file for caCORE SDK integration

3. Specify the output file name of the XMI document. Click on the **Export** button. The generated XMI file can be used for caCORE SDK integration.

Generating Hibernate Mappings

Perform the following steps to generate Hibernate files from the current object to database mapping.

1. Click the **Generate HBM Files** button to open the Open dialog box (*Figure 6.19*).
2. Select a directory to save the HBM file(s) and click **Open**.

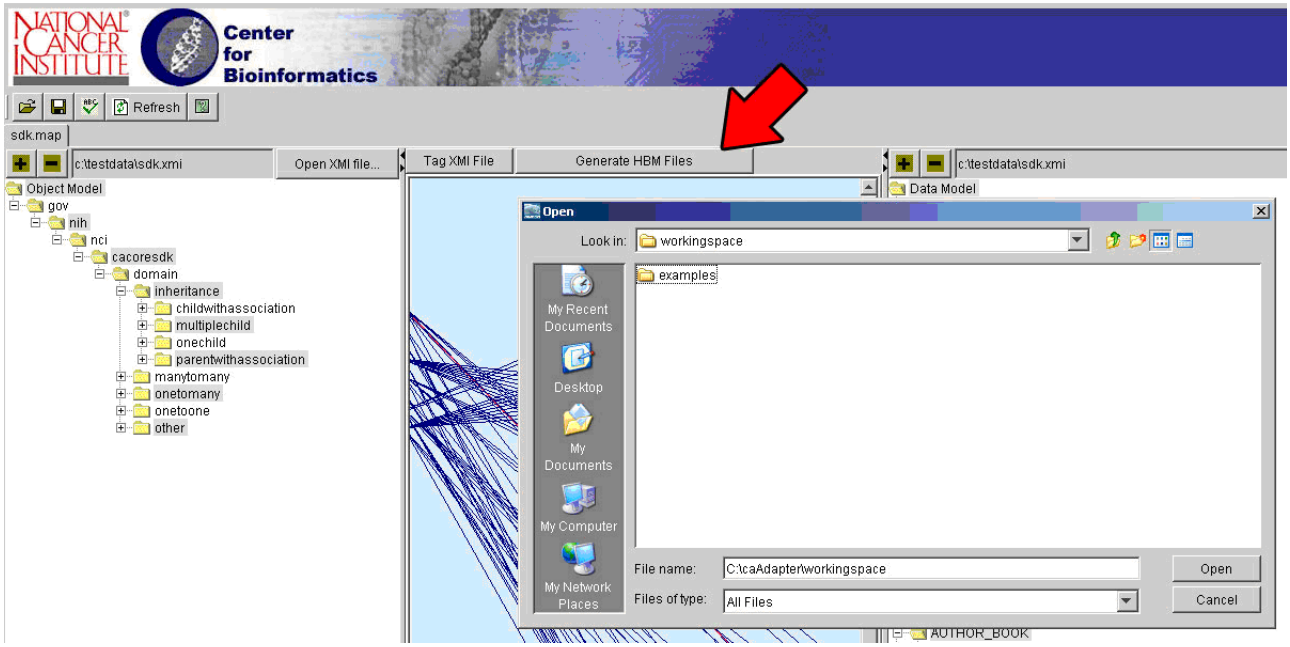


Figure 6.19 Generate HBM Files

caAdapter Mapping Scenarios

Before performing any of the following mapping scenarios, all dependency mapping between objects and tables must be complete.

One-to-One Bidirectional Mapping

To map one-to-one bidirectional relationships (Product and Orderline), drag the association (product, caAdapter uses "1 to 1" to indicate the corresponding element represents a one to one association) and drop it on the foreign key (ORDERLINE) of the corresponding table (PRODUCT). For one to one bi-directional mapping, only one end

of the relationship must be mapped; the other end (line) does not need to be mapped (*Figure 6.20*).

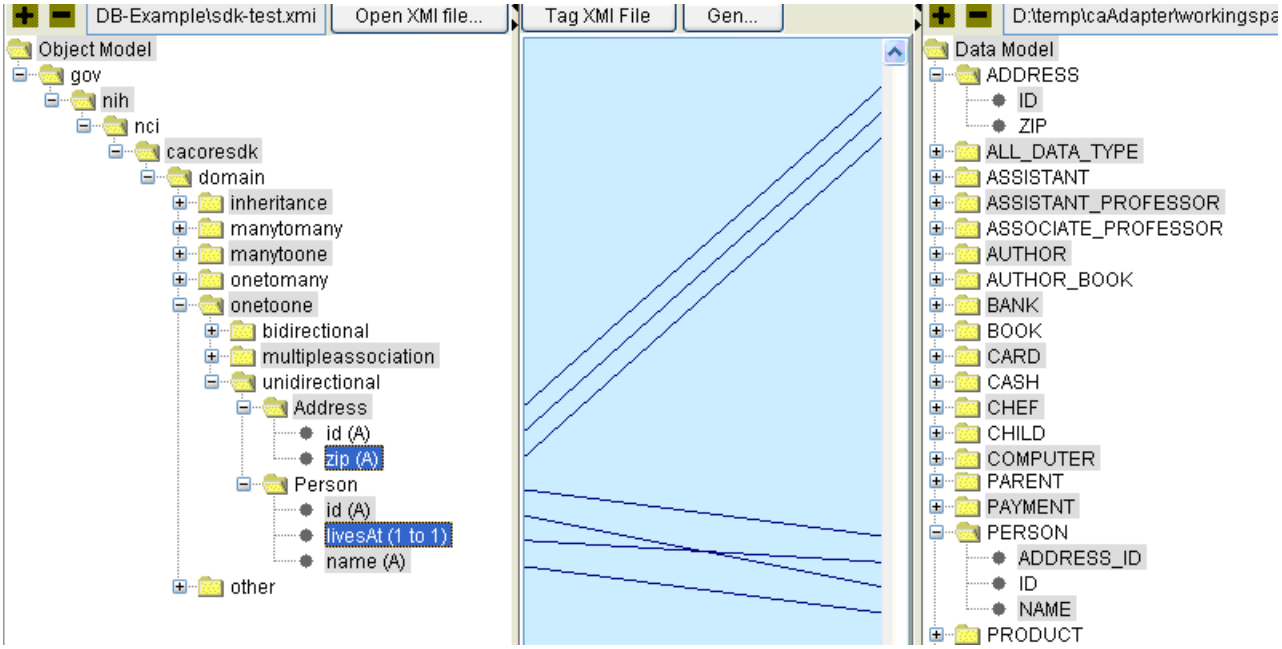


Figure 6.20 One-to-one bidirectional mapping

One-to-One Unidirectional Mapping

To map one-to-one unidirectional relationships (Address and Person), drag the association (liveAt, caAdapter uses "1 to 1" to indicate the corresponding element represents

an one to one association) and drop on the foreign key (ADDRESS_ID) of the corresponding table (PERSON) (*Figure 6.21*).

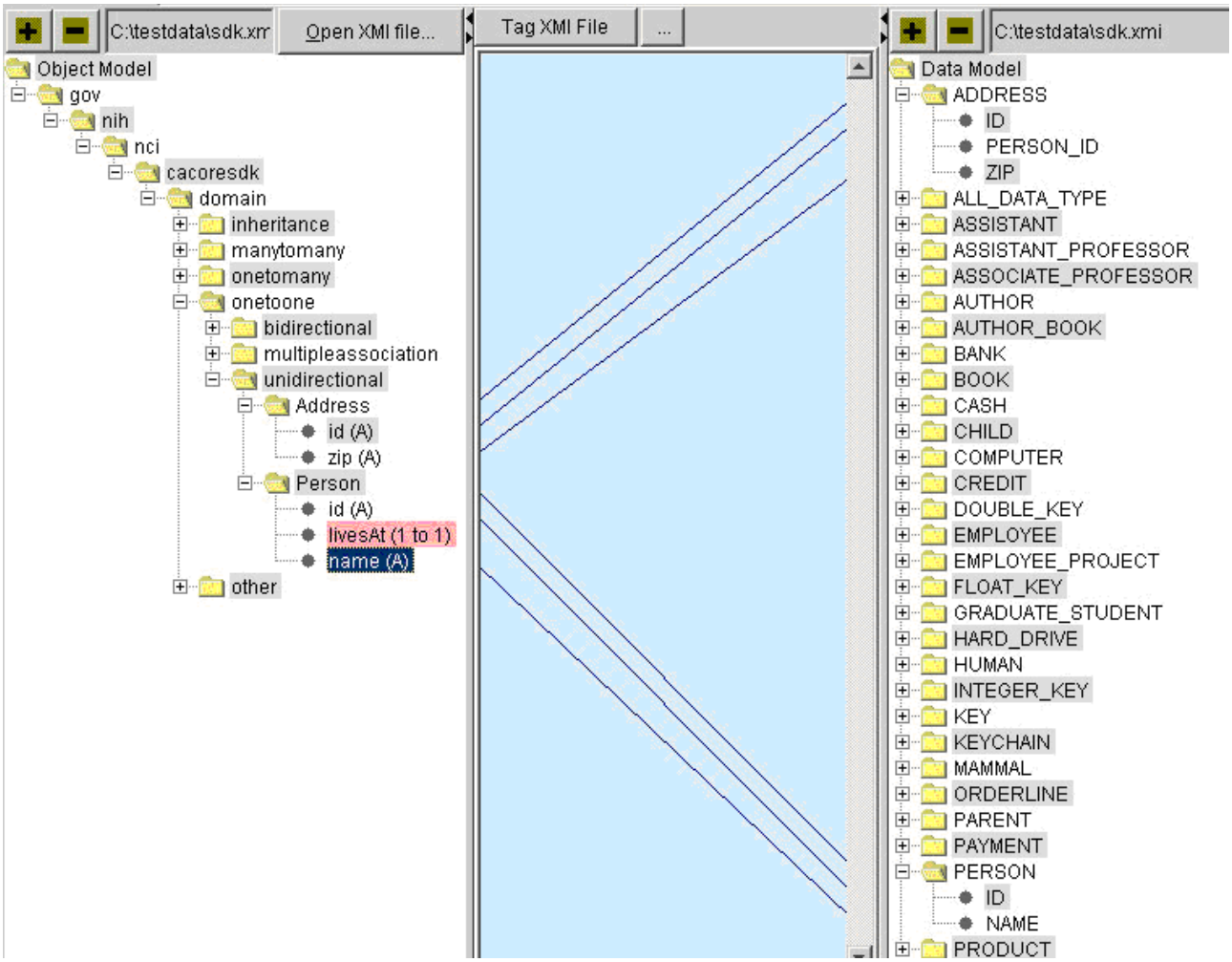


Figure 6.21 One-to-one unidirectional mapping

One-to-Many Bi-Directional Mapping

To map one-to-many bi-directional relationships (Computer and HardDrive), drag the association (computer, caAdapter uses "1 to Many" to indicate the corresponding element represents an one to many association) and drop on the foreign key (COMPUTER_ID) of the corresponding table(HARD_DRIVE). For one to many bi-direc-

tional mapping, the other end of the association is rendered by a dark blue element, and is not required to be mapped through caAdapter (*Figure 6.22*).

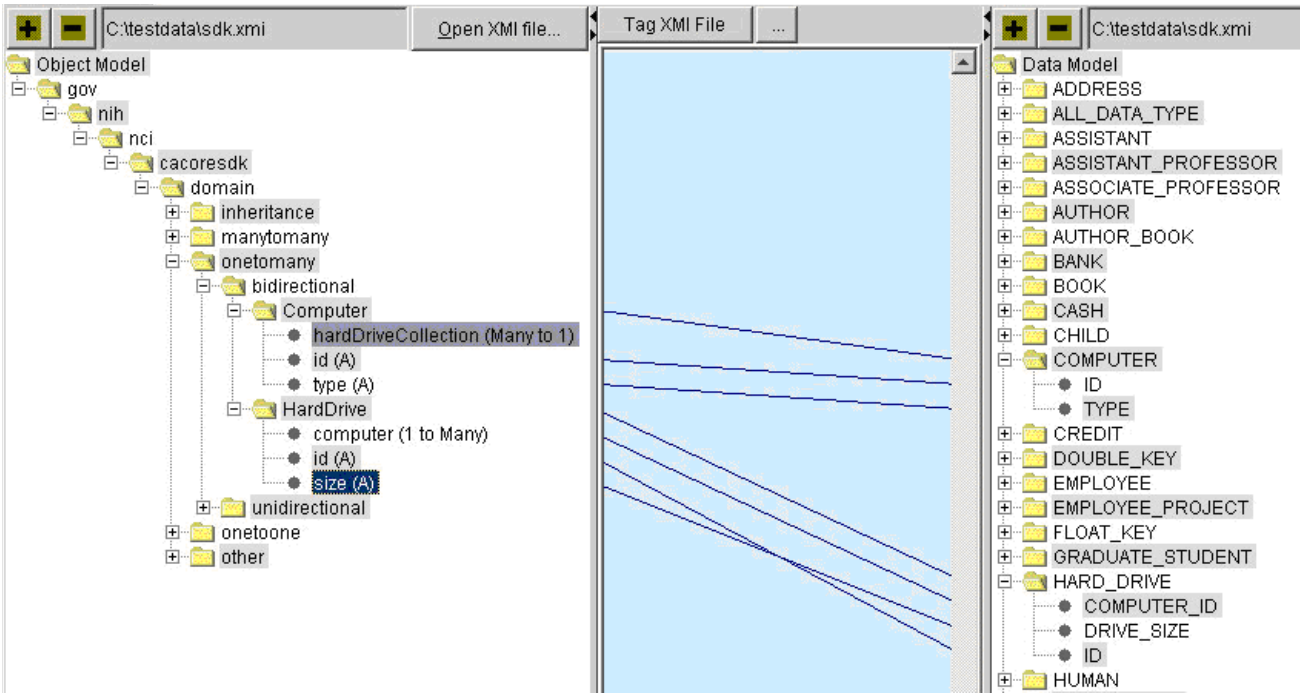


Figure 6.22 One-to-many bidirectional mapping

One-to-Many Unidirectional Mapping

To map one-to-many unidirectional relationships (Key and KeyChain), drag the association (keyRing, caAdapter uses "1 to Many" to indicate the corresponding element rep-

resents an one to many association) and drop on the foreign key (KEYCHAIN_ID) of the corresponding table (KEY) (*Figure 6.23*).

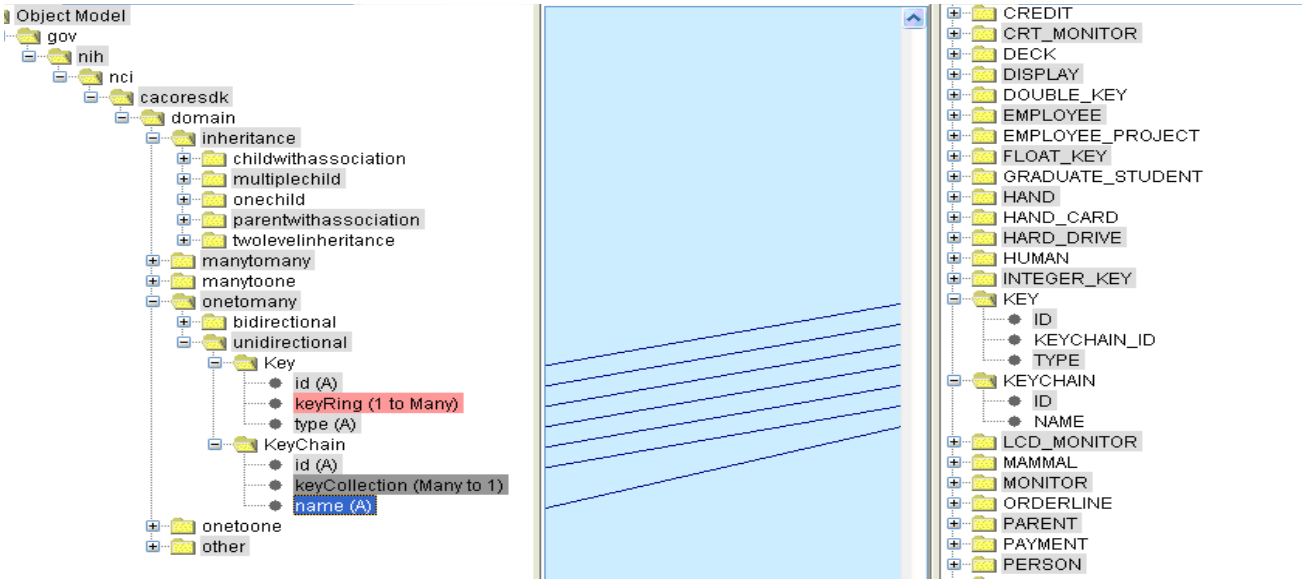


Figure 6.23 One-to-many unidirectional mapping

Many-to-One Unidirectional Mapping

The mapping of many-to-one unidirectional relationships (Chef and Restaurant) is conducted in a similar fashion as the one-to-many unidirectional relationship. Drag the association (restaurant, caAdapter use "1 to Many" to indicate the corresponding ele-

ment represents an one to many association) and drop on the foreign key (RESTAURANT_ID) of the corresponding table (CHEF) (*Figure 6.24*).

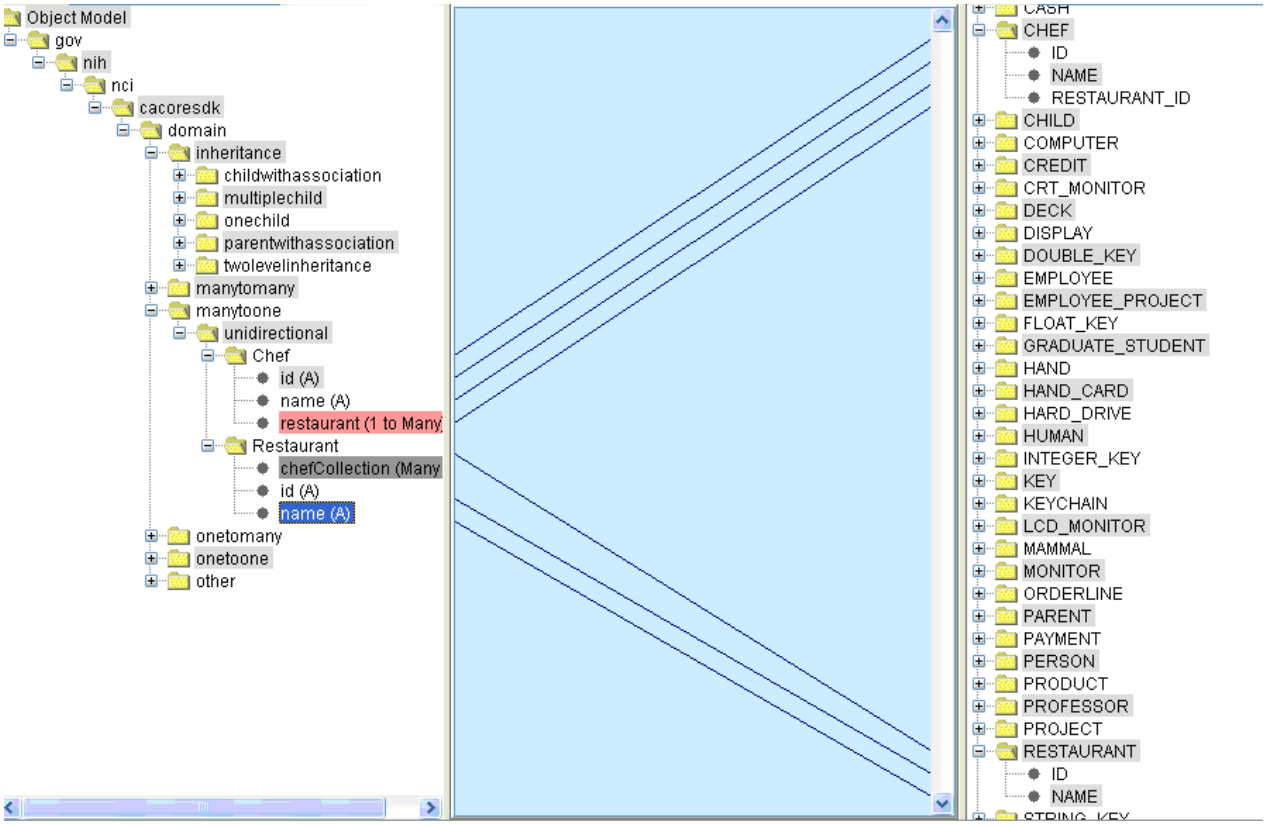


Figure 6.24 Many-to-one unidirectional mapping

Many-to-Many Bidirectional Mapping

To map a many-to-many bi-directional association, first identify a mapping table (EMPLOYEE_PROJECT, typically, the name of the mapping table is a concatenation of

the two tables corresponding to the two objects). Then, drag both ends of the associations and drop to the two columns in the mapping table (*Figure 6.25*).

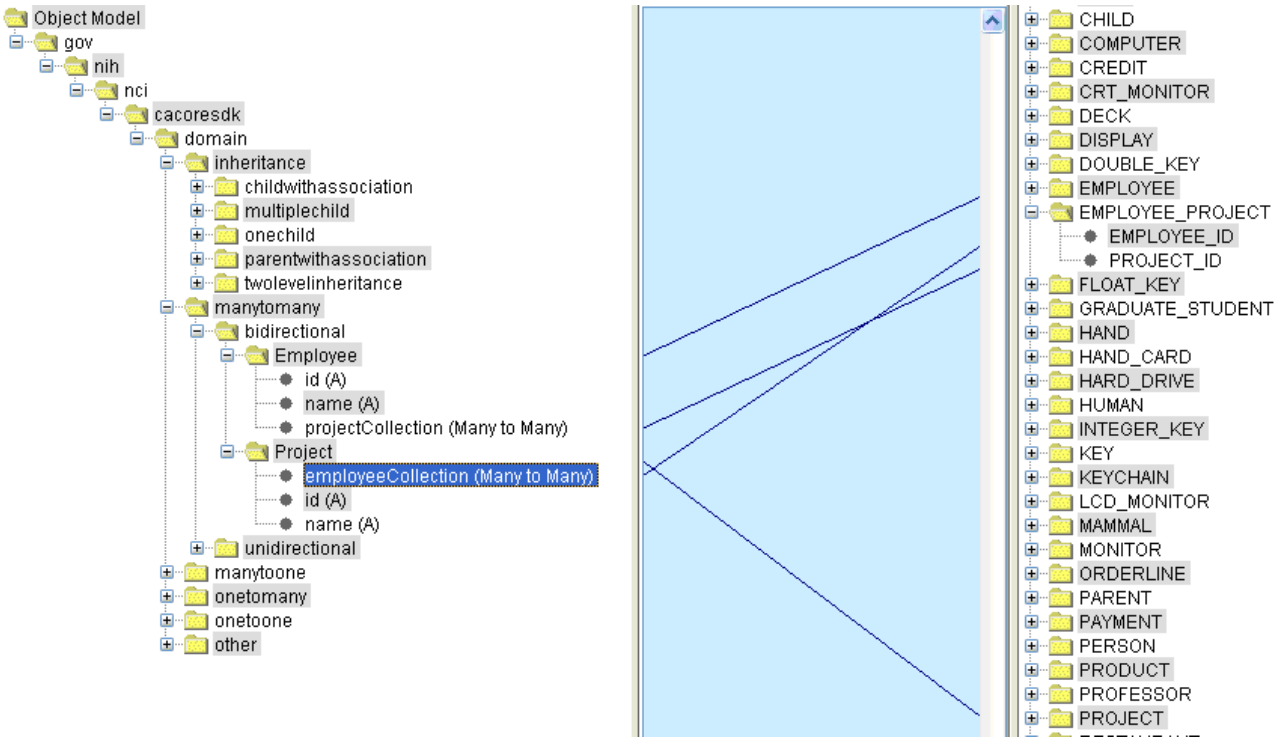


Figure 6.25 Many-to-many bidirectional mapping

Many-to-Many Unidirectional Mapping

To map a many-to-many unidirectional association, first identify a mapping table (AUTHOR_BOOK). Then, drag the appearing side of the associations to the corresponding column in the mapping table (*Figure 6.26*).

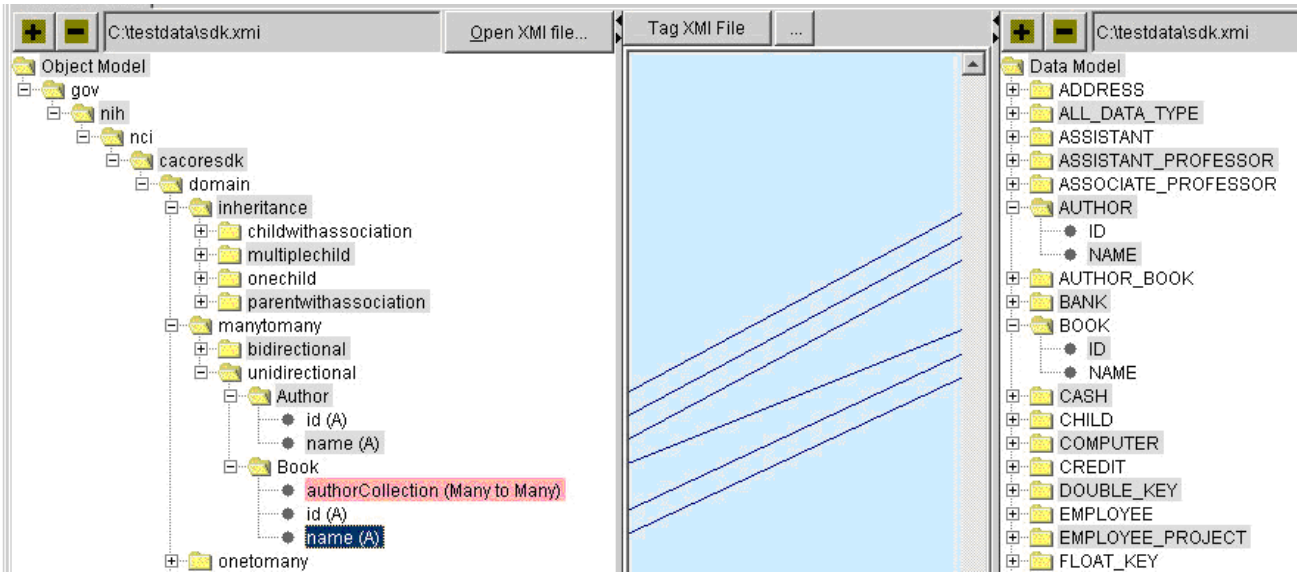


Figure 6.26 Many-to-many unidirectional mapping

Mapping Inheritance

To map inheritance through caAdapter, the mapping is done as in the previous examples. The tool will automatically marks inherited attributes as (A - Derived). Those attributes do not need to be mapped, and during the validation, an information level message is displayed (*Figure 6.27*).

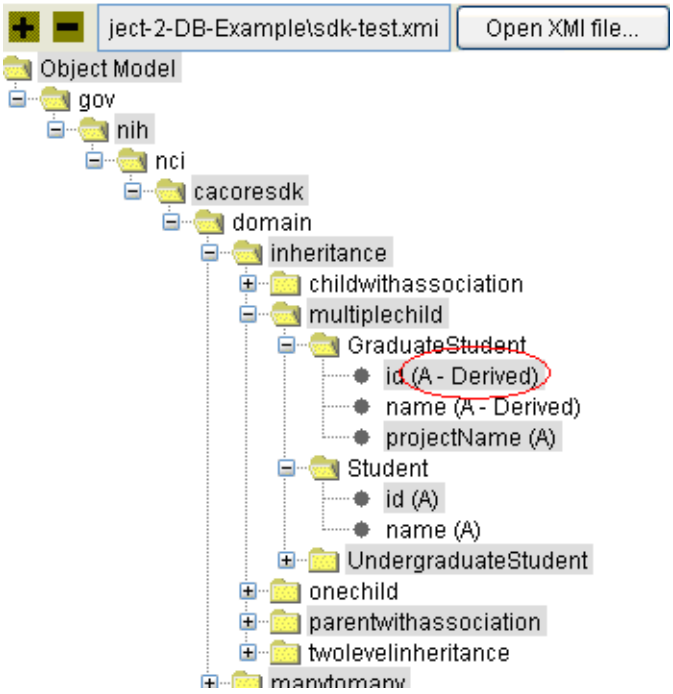


Figure 6.27 Inheritance mapping

User Interface Legend

Color Changes

- Light Gray - Cosmetic only, every other line
- Blue - Indicates the node represents a many-to-one association and is cannot be mapped.
- Pink - Indicates the association does not exist in the object model and will be used only for mapping purposes.
- Dark Blue - Currently selected item.

Node Details

(A) - Indicates the node is an attribute.

(A - Derived) -Indicates the node is an inherited attribute.

(1 to 1) - Indicates the node is a one-to-one association.

(1 to Many) - Indicates the node is a one-to-many association.

(Many to 1) -Indicates the node is a many-to-one association.

(Many to Many) - Indicates the node is a many-to-many association.

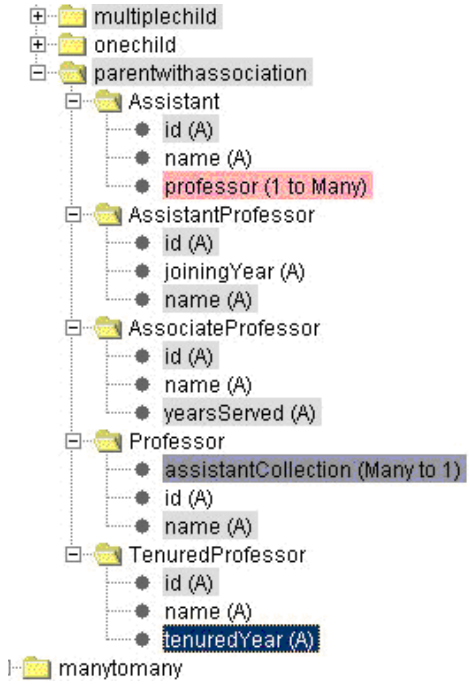


Figure 6.28 Color changes

CHAPTER

7

PERFORMING SEMANTIC INTEGRATION

This chapter describes the necessary procedures to prepare UML Models for loading into the caDSR using the Semantic Integration Workbench (SIW).

Topics in this chapter include:

Introduction

Semantic integration refers to the aspect of the caCORE architecture that addresses mapping of data element metadata to controlled vocabularies using immutable concept codes. For a UML model, proper semantic integration requires that each UML class and class attribute gets mapped to appropriate metadata that are based on a set of concepts in a controlled vocabulary. At NCICB, the preferred vocabulary is the NCI Thesaurus, maintained by the Enterprise Vocabulary Services (EVS) staff. It is the association with concept codes that permits unambiguous interpretation of UML model objects and mapping between objects in different domains. The resulting data elements are more sharable and interoperable.

There are two methods for accomplishing semantic integration:

1. Using the Semantic Integration Workbench (SIW)[1], which is the preferred method. The SIW facilitates and streamlines the process of semantic integration.
2. Manual annotation of the UML model using a modeling tool such as Enterprise Architect (EA) by inserting the required concept tags. For semantic integration tags and an example of the Semantic Connector report, contact NCICB application support at nciappsupport@mail.nih.gov. CAUTION: It is HIGHLY recommended to use the SIW to facilitate the semantic integration process. Manual annotation has the potential to introduce human errors that can cripple metadata registration using the UML Loader.

Once the EVS annotated version of the model has been approved by the model owner and EVS, a UML model is exported in XMI and sent to the NCICB caDSR team to be transformed into caDSR metadata via the UML Loader. The caDSR metadata registry, based upon the ISO/IEC 11179 standard, registers the descriptive information needed to render cancer research data reusable and interoperable. For more information about the ISO/IEC 11179 standard, see http://isotc.iso.ch/livelink/livelink/fetch/2000/2489/Ittf_Home/PubliclyAvailableStandards.htm??Redirect=1.

Semantic Integration Workbench

The Semantic Integration Workbench (SIW) is designed to facilitate the semantic annotation process and to remove, whenever possible, the requirement that users should understand ISO/IEC 11179, the relationship between UML Models and ISO 11179, the specific implementation of ISO 11179 objects in caDSR, or the more complicated details of the Semantic Connector implementation used in previous versions of the caCORE SDK. The SIW includes the following features:

- Validates against accepted datatypes.
- Automates searching and matching to potentially matching terms from the vocabulary system. The Semantic Connector acts to facilitate the matching of UML elements to controlled vocabulary.

1. The use of the SIW replaces the use of Excel spreadsheets previously described in *Performing Semantic Integration* in previous versions of the *caCORE Software Development Kit Programmer's Guide*.

- Ensures that files loaded into the caDSR have been checked for missing information and validated using Silver Level compatibility rules.
- Streamlines semantic annotation by offering direct queries to the NCI Thesaurus. Inserts the concept information from the search into the user's file in the SIW, creating the appropriate tag names automatically. Syntax errors in the final XMI file are no longer possible.
- Allows review of the final XMI file before it is loaded; allows individual review and acceptance of each entry.
- Allows viewing the XMI file after concept curation and allows better understanding of which concepts will be associated to the model when it is loaded. (Curation is performed by NCICB personnel.)
- Annotated XMI files contain the concepts, data elements, or value domains to which classes and attributes are mapped.
- Allows setting UML Loader defaults.
- Allows entering new EVS concepts (not yet in NCI Thesaurus) to associate with classes and attributes.

Launching the SIW

1. To launch the SIW, enter the following URL in your browser: http://cadsrsiw.nci.nih.gov/.
2. If Java Web Start is installed on your computer, follow the prompts to launch the SIW. The SIW always checks to ensure you are executing the most current version and, if not, it automatically updates itself. If you have the latest copy, it will proceed to the SIW menu.
   a. The Java Web Start dialog box displays (*Figure 7.1*). You will see the Semantic Integration Workbench start-up executing. This will take a few minutes.

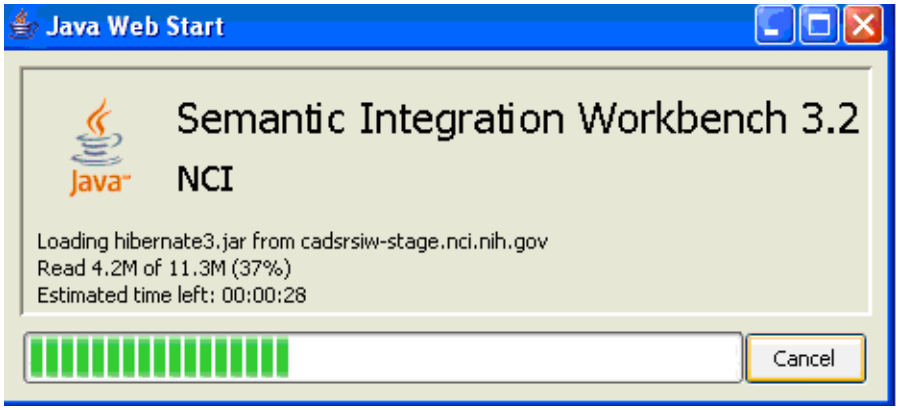


Figure 7.1 Java Web Start dialog box

   b. At the Security Warning prompt, click **Yes**.
   c. At the next prompt, click **Install**.
      The portions of the application are downloaded to cache, but the entire application itself is not downloaded.

3. If you get a File Download dialog box (*Figure 7.2*), you do not have Java Web Start installed. Click **Cancel**, and go to the Sun website to download it: http://java.sun.com.

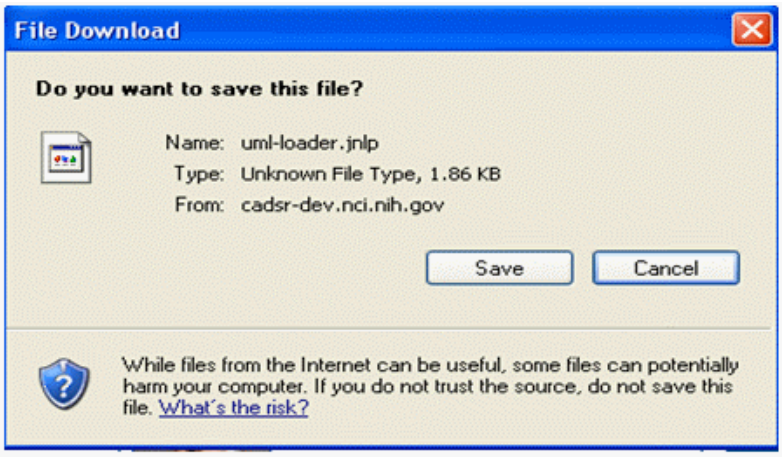


Figure 7.2 File Download dialog box that opens if you do not have Java Web Start installed on your computer.

4. Open a browser and enter http://cadsrsiw.nci.nih.gov/.

SIW User Modes

When the SIW is running, the viewer displays the Welcome panel that includes the five modes listed in the order of their use in the SIW workflow (*Figure 7.3*).

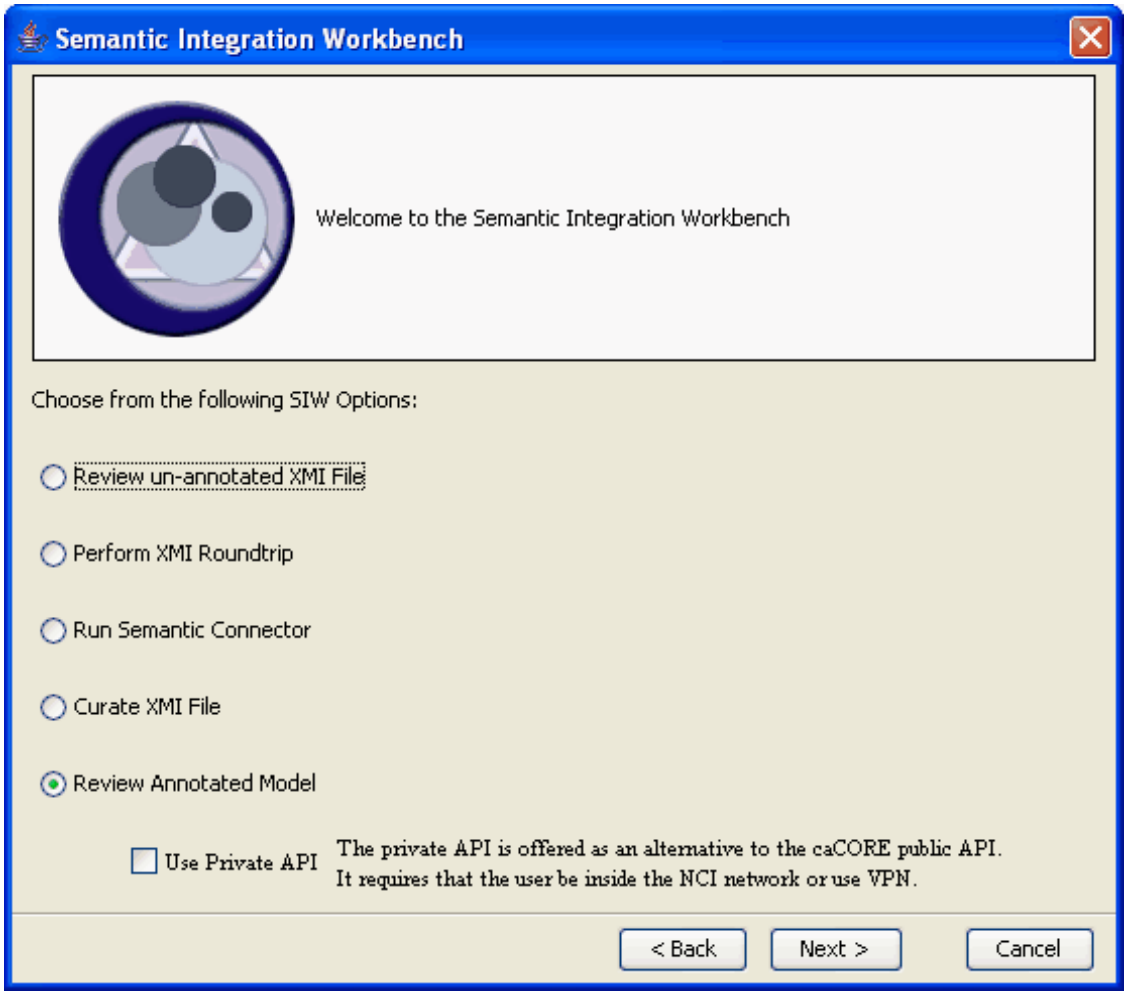


Figure 7.3 User modes of the Semantic Integration Workbench

Each mode of the SIW is described below. The use of each mode is then described in *Suggested Workflow for the SIW* on page 97.

Review Unannotated XMI File

This mode allows you to view the XMI representation of the UML Model exported by Enterprise Architect and is performed by the model owner. It should be used before running other SIW steps. This is an easy way to check for missing description or documentation tags for classes and attributes. These are flagged as missing in the Errors tab in the SIW. The caCORE 3.2 release allows a model to be exported **using the default settings** instead of having to select specific options.

Note: The "Fix XMI" task is no longer needed to prepare the file for use with SIW or caAdapter. Therefore the XMI file can be imported and exported from your modeling tool to make changes without losing SIW concept annotations.

Input: The original UML model in XMI format, exported from Enterprise Architect (EA). From EA, the XMI file is named with a user-defined name: `${filename}.xmi`.

Example: `myModel.xmi`

Output: `${filename}.xmi`: Any changes you make to the XMI file can be saved and you may name the file anything you wish. We recommend that you use the naming conventions established in prior releases. The output file can be saved with any name the model owner chooses by clicking **Save** from the SIW File option, and then **Exit**.

Perform XMI Roundtrip

This optional step in the semantic integration process is performed by the model owner in an attempt to automatically annotate his or her model with existing data from caDSR. Running the XMI roundtrip can save time in the following two situations:

1. A prior version of this model was previously loaded and parts of the model have not changed, specifically when no changes have been made to the names of the classes and attributes.
2. The model shares classes and attributes with another, previously loaded model, using the same names for the matching classes and attributes.

   Input: The XMI file: `${filename}.xmi`

   Output: `roundtrip_${filename}.xmi`. A partially annotated XMI file. The file is annotated with caDSR public IDs rather than EVS concepts. After performing this step, the model owner should review the mapping in the XMI Review mode (mode five)

Run Semantic Connector

This mode launches the Semantic Connector from the SIW and is performed by the model owner. This mode provides model owners with the option to selectively include only certain packages in this step by entering the package names into a user interface. If no package names are included in the filter, all packages are processed by the Semantic Connector. This mode performs an EVS search for each element in the UML Model and attaches one or more EVS concepts per element to produce the Semantic Connector Report (XMI). Although the Semantic Connector will add concepts to anything not marked as "reviewed", even those mapped to a CDE, when validating and loading the model later, the concepts are ignored in favor of the mapped CDE.

Input: The un-annotated or roundtrip xmi file: `roundtrip_${filename}.xmi` or `${filename}.xmi`.

Output: The output file from this step is appended with `FirstPass_` in XMI format: `FirstPass_$(filename}.xmi`, referred to as the "Semantic Connector Report". This file is sent to the EVS curation team for review and insertion of new concepts to match the UML class and attribute entities.

The Semantic Connector Report is the input to the Curate XMI File step.

Curate XMI File

This mode is performed by the EVS Concept Curation team. During this step, the EVS team adds and removes concepts and indicates recommended semantic mappings. In this mode, new or existing EVS concepts can be used to annotate the model.

Input: The First Pass XMI File: `FirstPass_${filename}.xmi` or `FirstPass_roundtrip_${filename}.xmi`.

Output: The resulting updated XMI file, still named the same as the input file, `FirstPass_${filename}.xmi` or `FirstPass_roundtrip_${filename}.xmi`. This is referred to as the Curated XMI File, which is the input to the mode, Review Annotated Model.

Review Annotated Model

This option is performed by the model owner or reviewer. The SIW performs a number of validation checks to ensure that the XMI file will be correctly transformed into caDSR metadata. In this mode, users can search EVS for concepts to change the mapping between a specific class or attribute and EVS concept. Users may also choose to map UML attributes to existing caDSR value domains or data elements.

Note: If an attribute is mapped to a CDE, its Object Class is used as the basis for the containing class, regardless of whether or not the Class is mapped to concepts. When this occurs, a warning message displays.

Input: The Curated XMI file, `FirstPass_${filename}.xmi` or `FirstPass_roundtrip_${filename}.xmi`.

Output: The reviewed and completed XMI file is saved; it is recommended that the name be saved as `Annotated_FirstPass_${filename}.xmi`. This file is used as input to the UML Loader. This file is referred to as the Reviewed Annotated XMI file.

To proceed with using the SIW, read the following suggested workflow. Each mode in the SIW is then described in more detail in the sections following the suggested workflow.

Suggested Workflow for the SIW

The semantic integration process using the SIW is illustrated in *Figure 7.4* and is divided into four phases:

1. In Phase One of the process, the model owner creates an XMI file from the UML model constructed in Enterprise Architect (EA) and prepares it for submission to NCICB. This optionally includes running SIW Roundtrip to match the model to a previously loaded version.
2. In Phase Two, the names of entities (class, attribute) in the UML Model are semantically annotated by EVS in an iterative process with the model owner. The model owner reviews the Annotated XMI file and, if accepted, completes the submission template and sends a request to NCICB.
3. In Phase Three, the UML model is loaded to the caDSR Sandbox. If the model loads successfully, it is submitted to load to caDSR Production for final curation and a compatibility review.

4. Phase Four is the last part of semantic integration. Once the model has passed the compatibility review, the model owner can use the SIW to run RoundTrip, which will insert the caDSR metadata identifiers for the model into the XMI as tagged values and ready the model for reuse or the next version. Model owners can also produce the final public APIs using the SDK code generator.

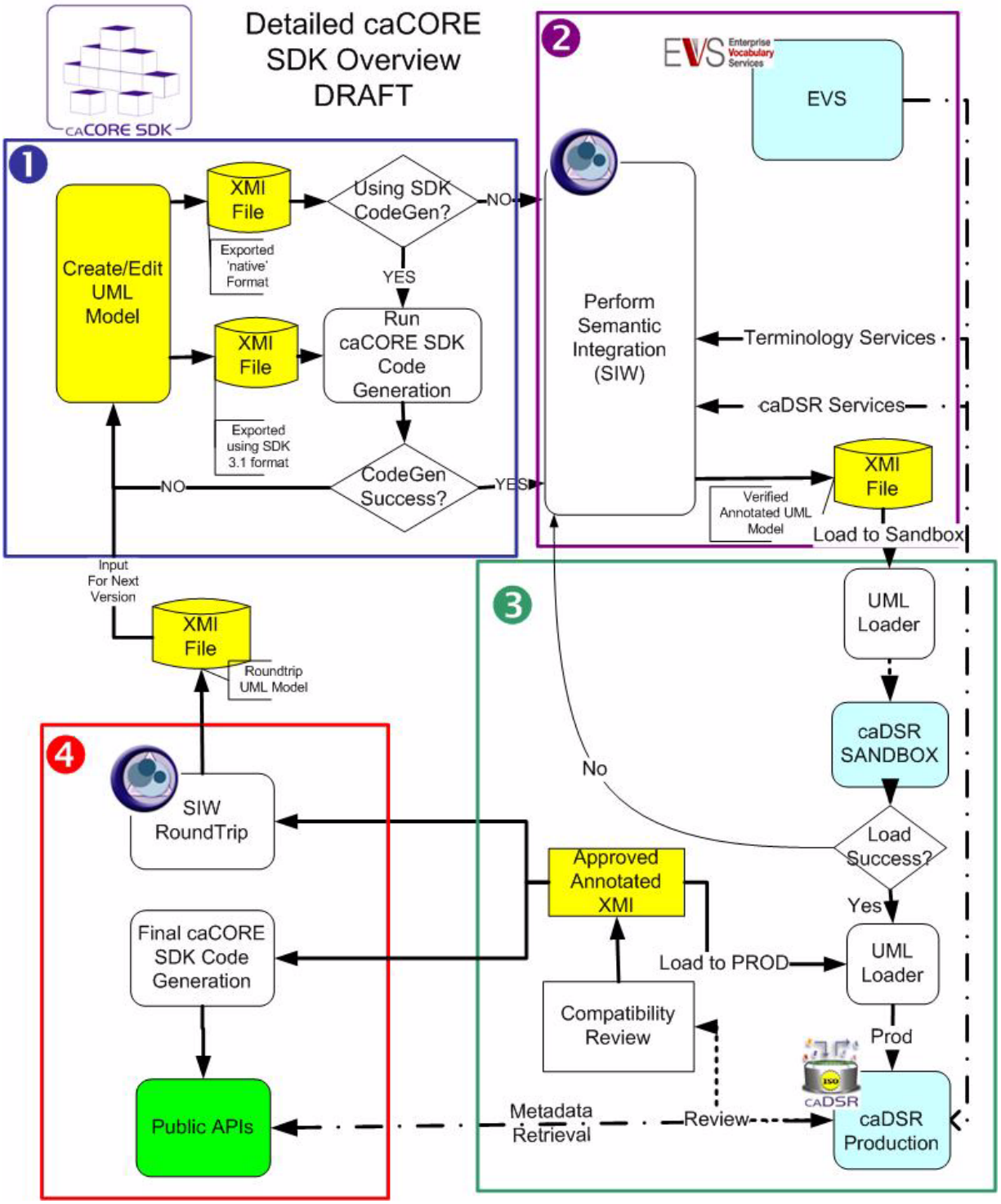


Figure 7.4 Overview of the semantic integration process; caCORE infrastructure components = light blue, caCORE SDK components = white; artifacts (documents) = yellow; generated software system = green

See the GForge UML Model Project site and click the **Docs** tab for a detailed Standard Operating Procedure (SOP) and for a cross functional view of the above process depicting who performs each phase of the process.

Using the XMI Roundtrip Mode

Running the XMI Roundtrip mode can save model owners a considerable amount of time by automatically annotating models based on other previously loaded models. The roundtrip task maps UML classes and attributes to existing caDSR Common Data Elements (CDE). Use the following steps with this mode. The automated matching is based upon the new model having used exactly the same class and attribute names within their unannotated model.

1. Enter the Classification Scheme Long Name of a project that was previously loaded. The Classification Scheme Long Names of projects are displayed in the CDE Browser tree structure under “Classifications”. Enter the Classification Scheme Version of the project you are referring to as it was previously loaded. For example, a PIR project name would be entered as “Grid-enablement of Protein Information Resource (PIR)”. For version 1, enter the version as “1.0”. The **Next** button is not available until a valid project is typed and verified by clicking the **Verify** button. If the SIW cannot find the project, a window opens with the message "Project Not Found".

   In *Figure 7.5*, the model owner wants to automatically annotate version 2 of his model, he chooses a project name with version 1. In order to enable the **Next** button, the SIW must verify that the supplied project name / version combination exists in caDSR.

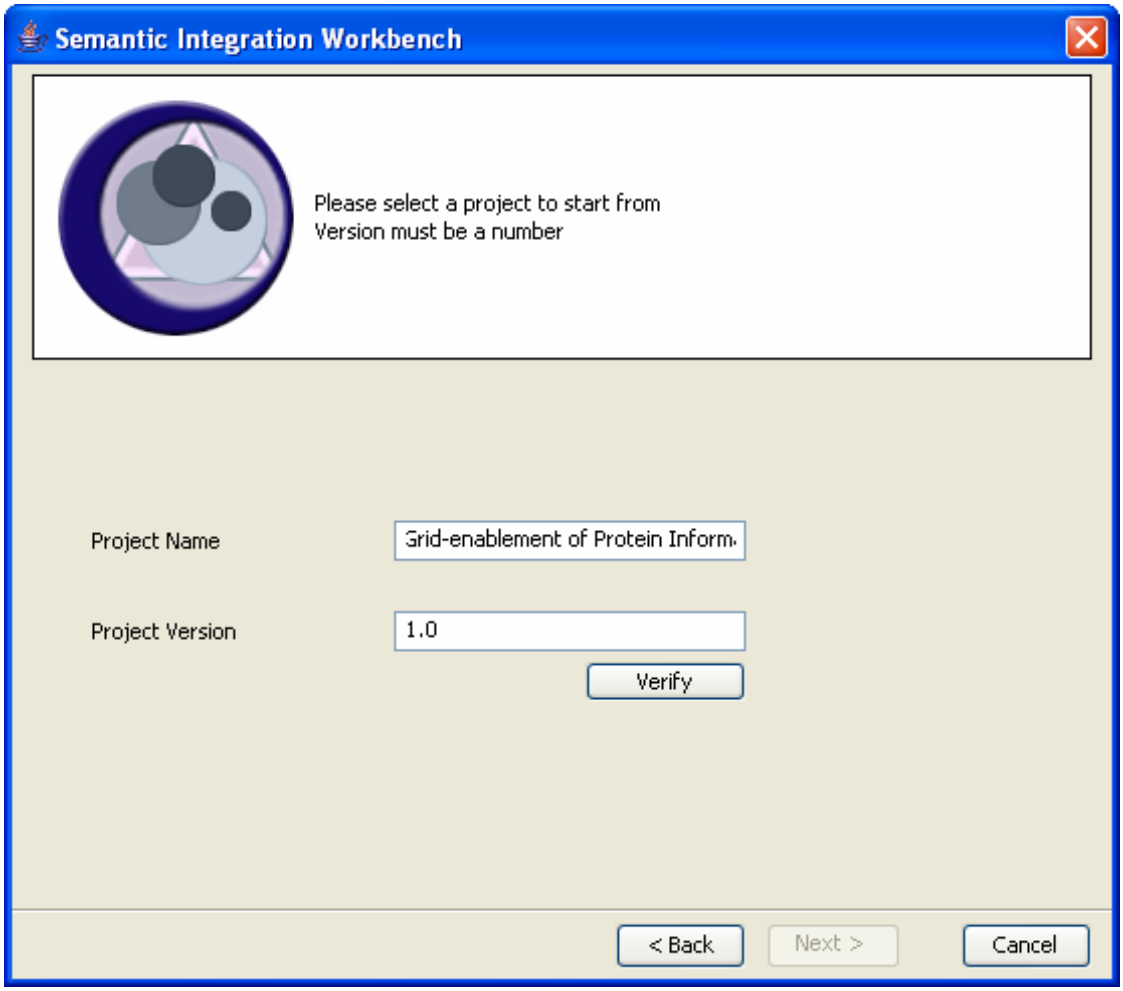


Figure 7.5 Selecting a Project Name and Version

2. Click **Next** to trigger the UML roundtrip task.
3. Use one of the following options to select your XMI file (*Figure 7.6*):
   - Click the **Browse** button. In the file that opens, navigate to the appropriate directory. Select the file and click **Open**.
   - Enter the full path of a file into the text box. Example: `C:/XMI/test.xmi`. Select the XMI file exported from EA.
   - Select a file from the Recent Files list that is located below the text box. (The five most recently selected files display. Files other than those listed must be selected using one of the other options.)

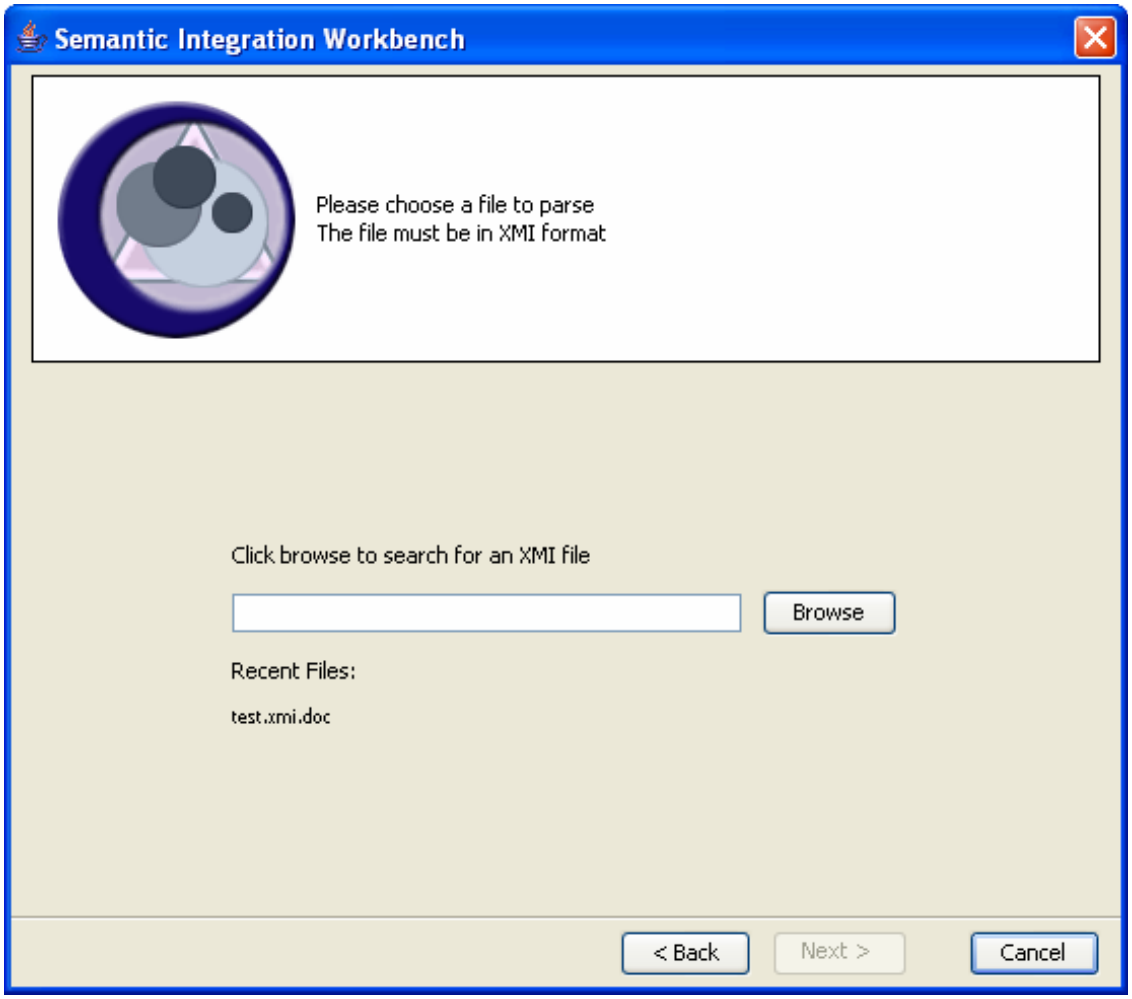


Figure 7.6 SIW retrieves CDE public ID and version for each attribute

4. For each UML Attribute in the XMI file, the SIW attempts to find a CDE with an alternate name matching the fully qualified attribute name. If that alternate name is classified with the supplied project name / version, the CDE selected for the UML attribute is annotated with the CDE public ID / Version. The model owner should then open the SIW in XMI Review mode and review the mappings.

Running the Semantic Connector

1. To initiate the Semantic Connector, select **Run Semantic Connector** in the SIW Welcome screen and click **Next**.
2. Set the package filter to select only specific UML entities and exclude the rest. Enter text in the Package field, then click **Add** to add the name of packages to include in the Semantic Connector step. Click **X** to remove a package's name. If no names are specified, all packages are included in the semantic connector run (*Figure 7.7*).

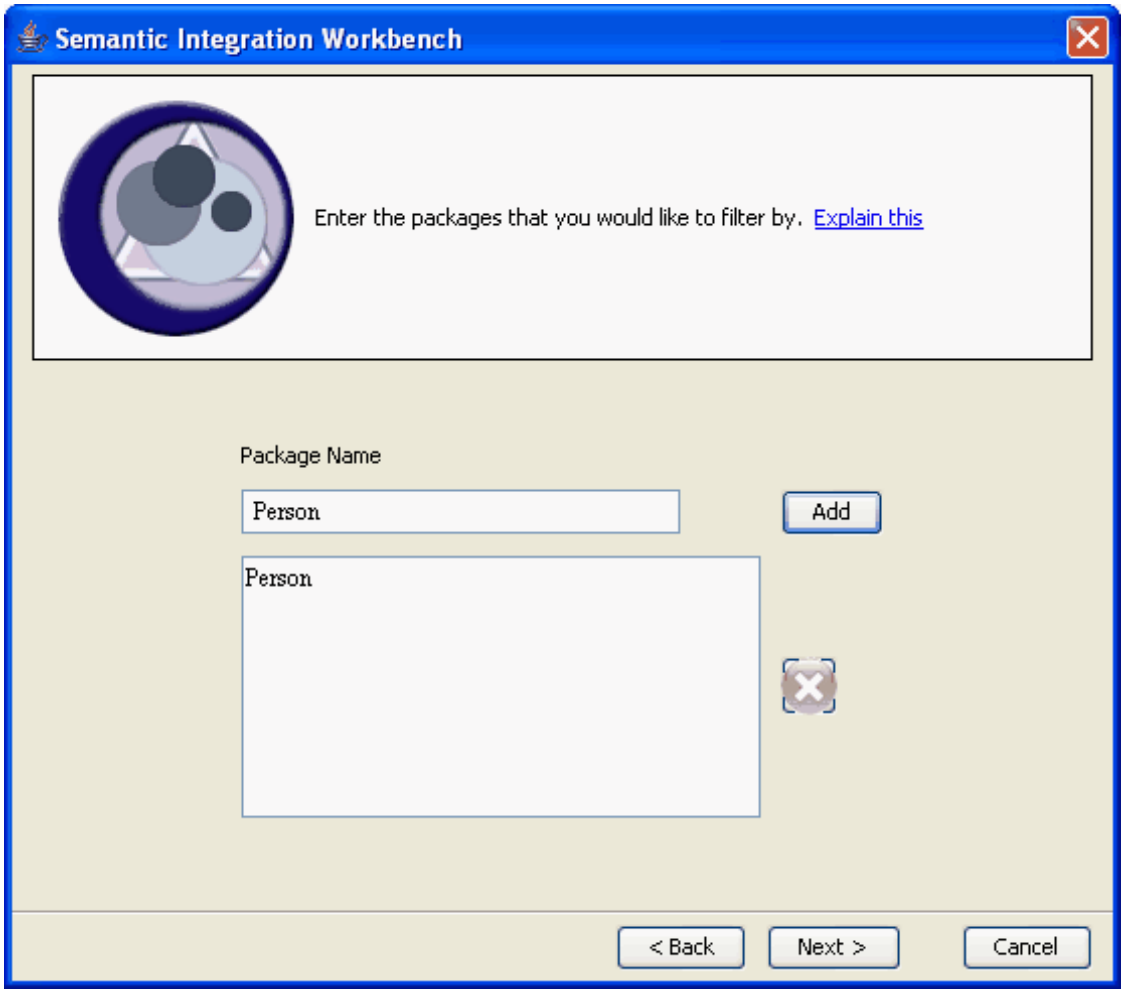


Figure 7.7 Filter for selecting a package

3. Use one of the following options to select your XMI file (*Figure 7.8*):
   - Click the **Browse** button. In the file that opens, navigate to the appropriate directory. Select the file and click **Open**.
   - Enter the full path of a file into the text box. Example: `C:/XMI/test.xmi`. Select the XMI file exported from EA.
   - Select a file from the Recent Files list that is located below the text box. (The five most recently selected files display. Files other than those listed must be selected using one of the other options.)

   Click **Next** to launch the Semantic Connector process.

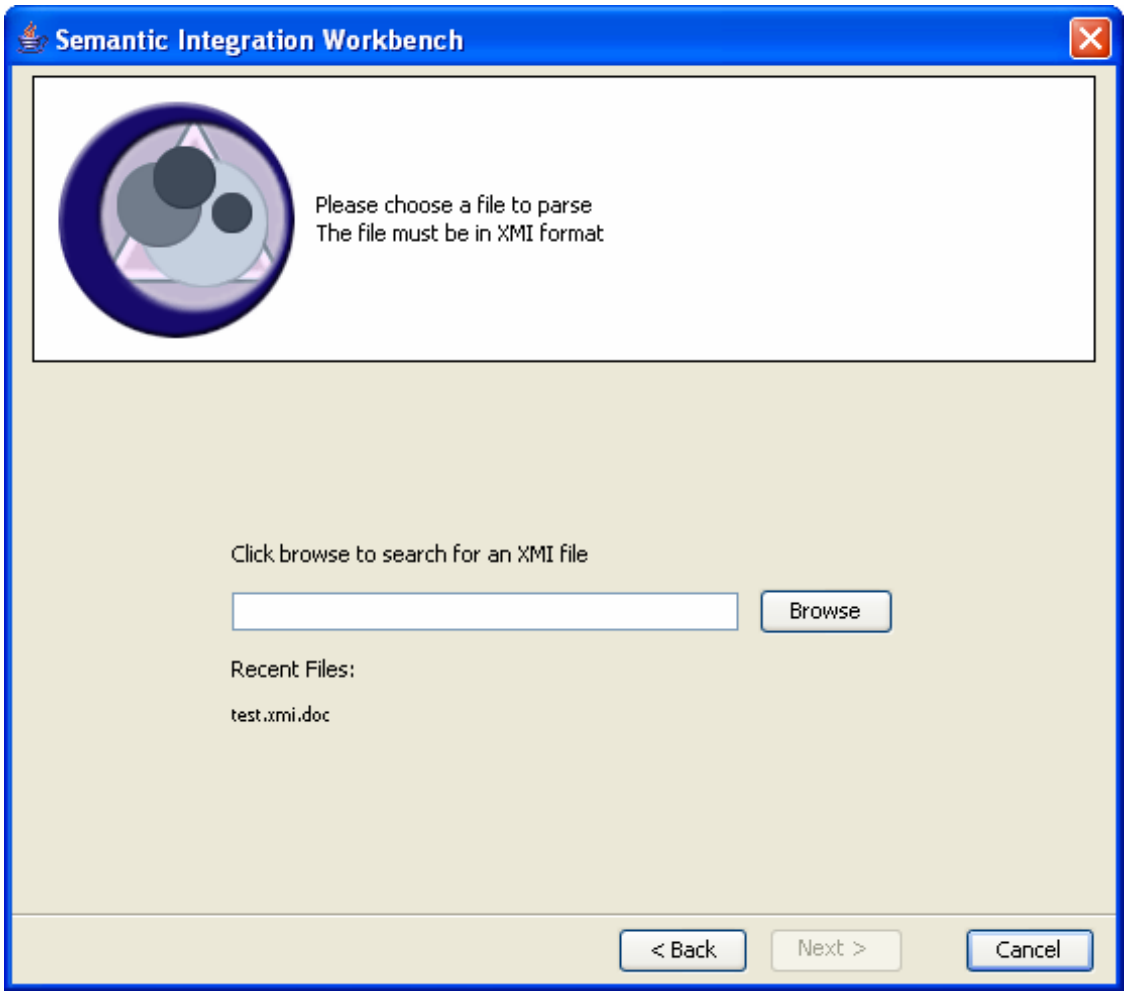


Figure 7.8 Select an XMI file to parse

4. The SIW displays a task monitor and the message, **Parsing XMI File**, during the process.
5. If successful, the Semantic Connector generates tagged values and inserts them into the XMI file. A new panel displays a confirmation of the process and the location of the newly generated file.

   The generated EVS report in XMI is placed in the same root directory as the input file. For example, if the input file was in `C:/TEST/mydirectory/protein.xmi`, then the output file is `C:/Test/mydirectory/FirstPass_protein.xmi`.

   If the Semantic Connector process was not successful, an error message displays, or it simply may not complete. The most likely cause of your error is that your XMI file is not in the expected format and should be corrected before re-running the Semantic Connector. Check to ensure that the file was exported from EA with the correct settings as shown in *Figure 7.9*.

Note: For performing semantic annotation, the file should be exported using the default EA setting for all options. Note that the required export settings have changed from 3.1.

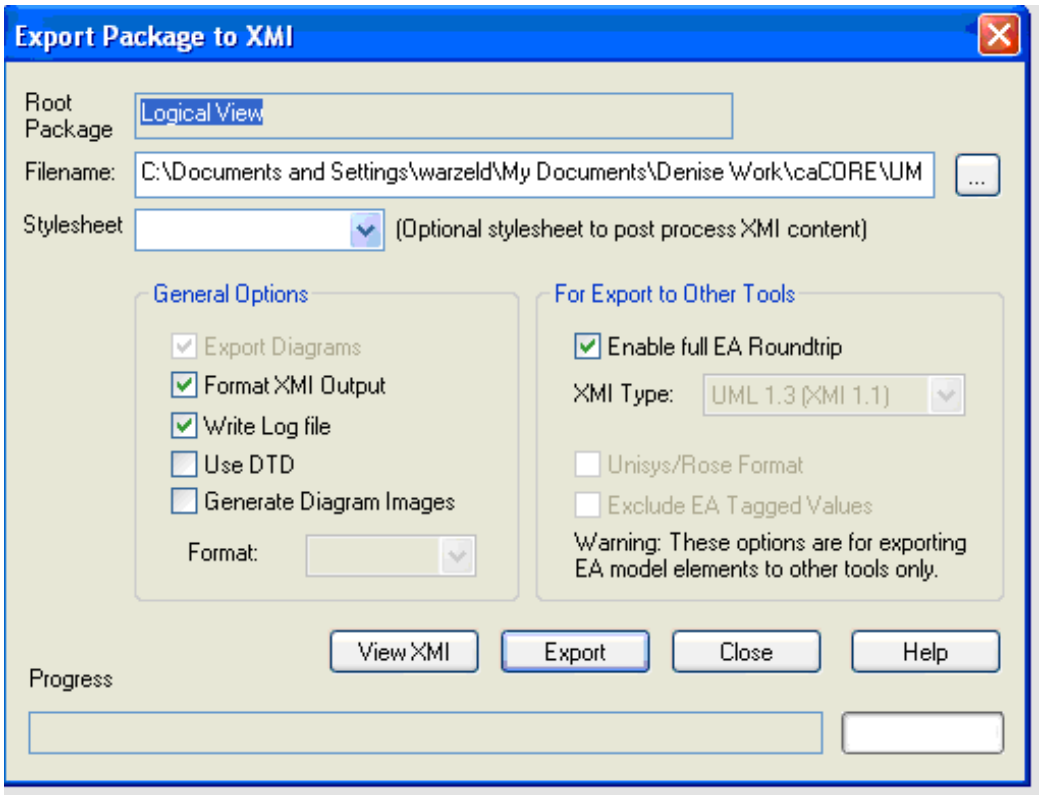


Figure 7.9 The Export Package dialog box from Enterprise Architect for defining XMI file export settings

Exiting the SIW

To exit the SIW from all modes, select **File > Exit**.

Curating XMI Files

The **Curate XMI File** option on the SIW Welcome screen is used by EVS concept curators before it is submitted to the model owner for review.

1. Once you select the **Curate XMI File** option, navigate to and open the report in XMI format you want to review and modify.

   The SIW viewer window opens, displaying the Navigation Tree in the left-hand pane, and the Errors and Log tabs at the bottom of the viewer (*Figure 7.10*).

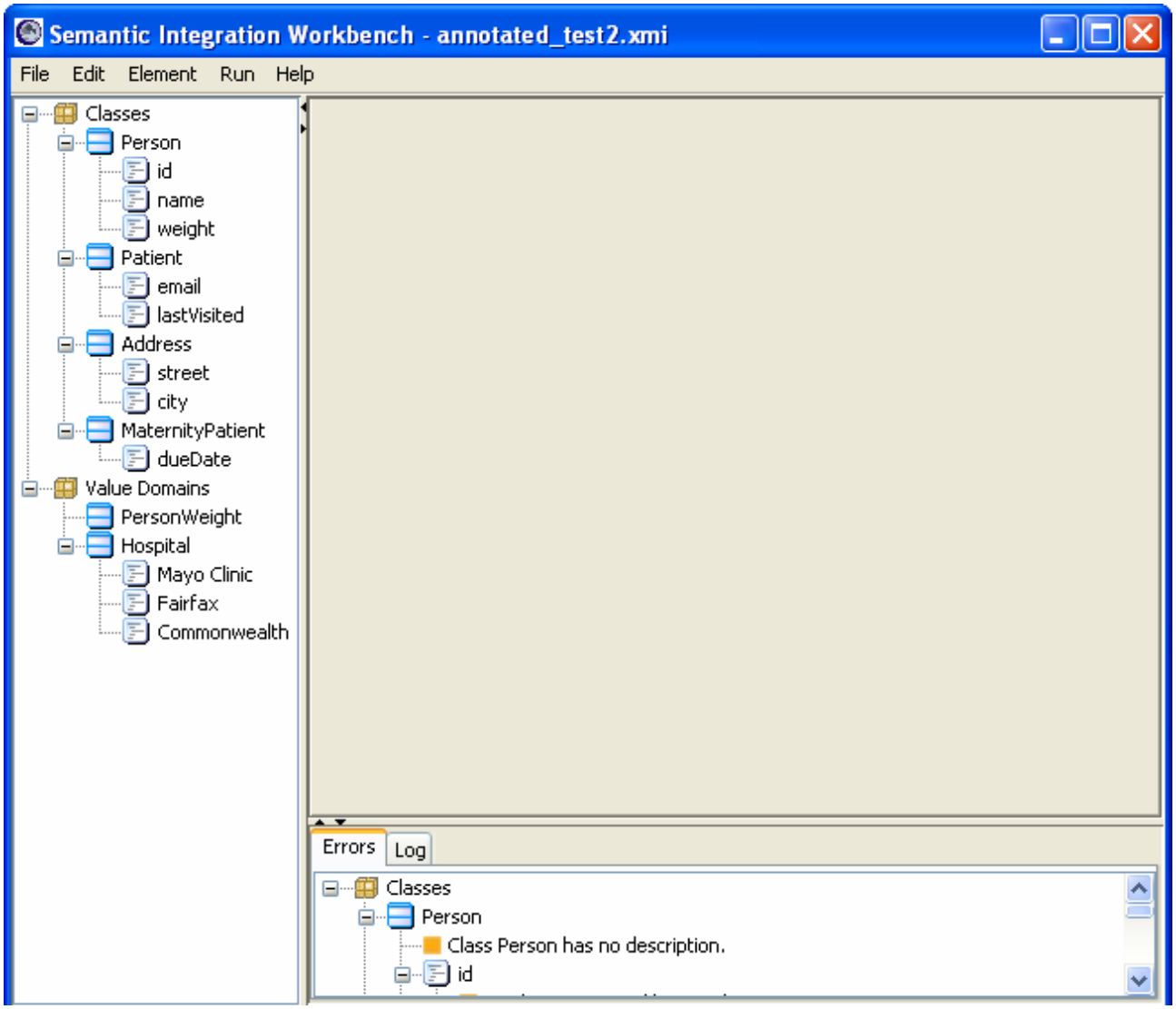


Figure 7.10 SIW viewer window

Browsing the Navigation Tree

The Navigation Tree displays the structure of your model as indicated in your XMI file. All elements of the open file are listed in the tree. You can browse the tree and quickly see by the checkmarks which elements in a file have been reviewed and completed (*Figure 7.11*).

The Navigation Tree is initially completely expanded to display all internal nodes, but it can be expanded and collapsed by clicking on the individual nodes.

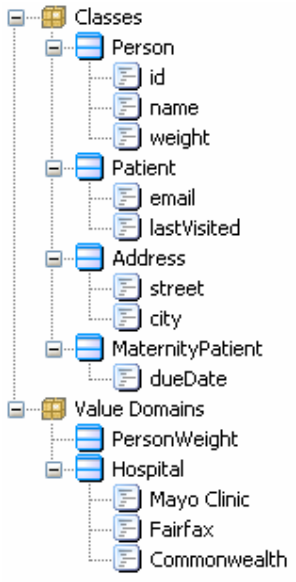


Figure 7.11 Navigation Tree

When you click on a node, the concept information of that node displays in the UML Concept Code Summary pane to the right hand side of the tree, along with the UML model definitions. If you select a node and the UML Concept Code Summary pane remains empty, or no UML documentation or description field displays, this indicates that the element is not yet annotated and also that it did not have a UML definition (*Figure 7.12*). UML definitions are recommended, because they help EVS curators understand the purpose and intent of the model owner, but are not mandatory unless you are striving to meet caBIG Silver-level compatibility guidelines. If you do not supply a definition, one will be supplied.

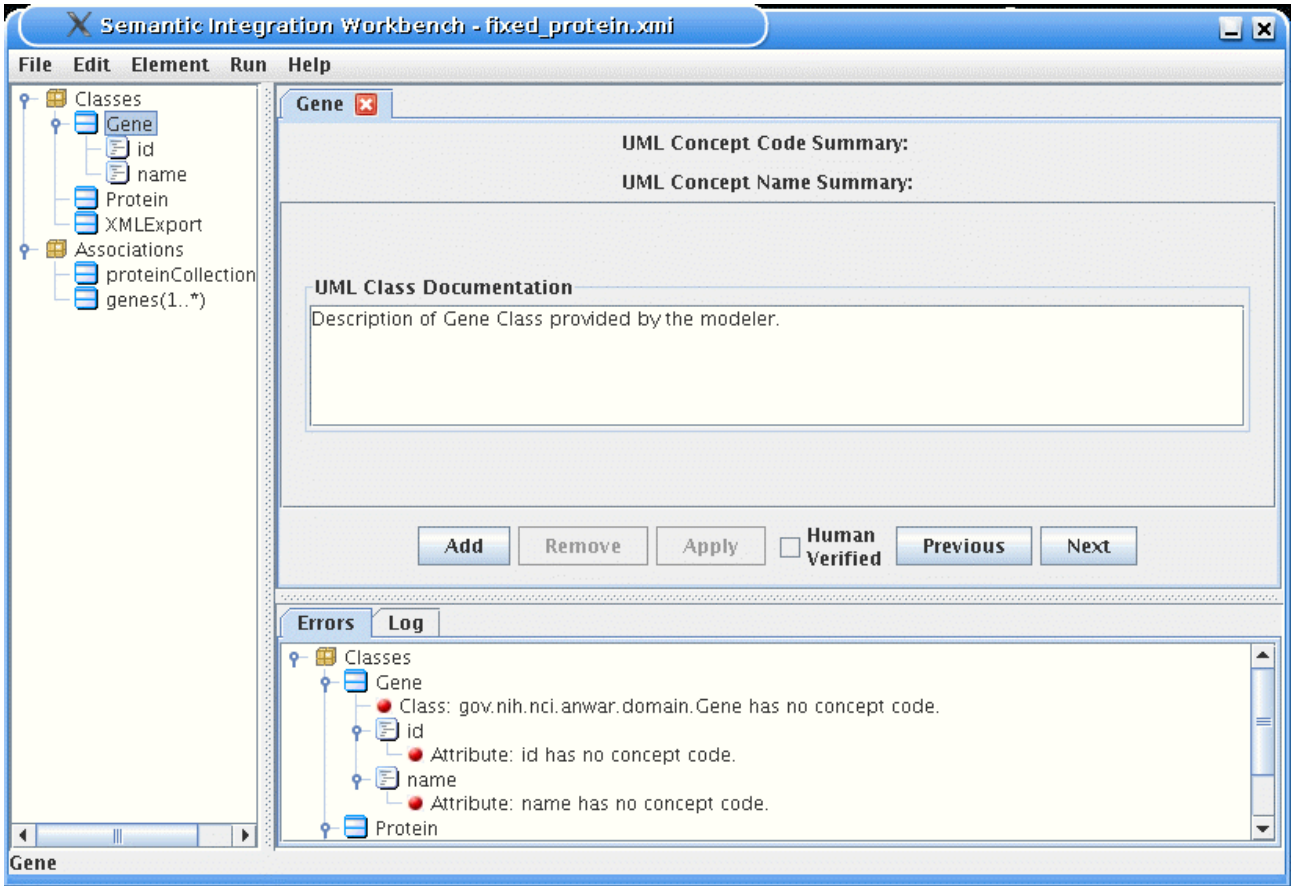


Figure 7.12 A UML Concept Code Summary pane displaying a selected class with no concept information.

If the class or attribute is annotated, the UML Concept Code Summary pane displays the information, including the concepts when they are available (*Figure 7.13*). In this case, the list is positioned in semantic order, with one 'Primary Concept' and one or more 'Qualifier' concepts, with the highest numbered Qualifier listed first in the Review panel, the Primary Concept last: QualifierN Qualifier1 PrimaryConcept. The concept names are also ordered so you can compare the name that will be created in caDSR with the name of the UML Element. This order is consistent with the naming conventions explained in *Chapter 5* and reflect how the concept names will be positioned when forming the name of the item in caDSR.

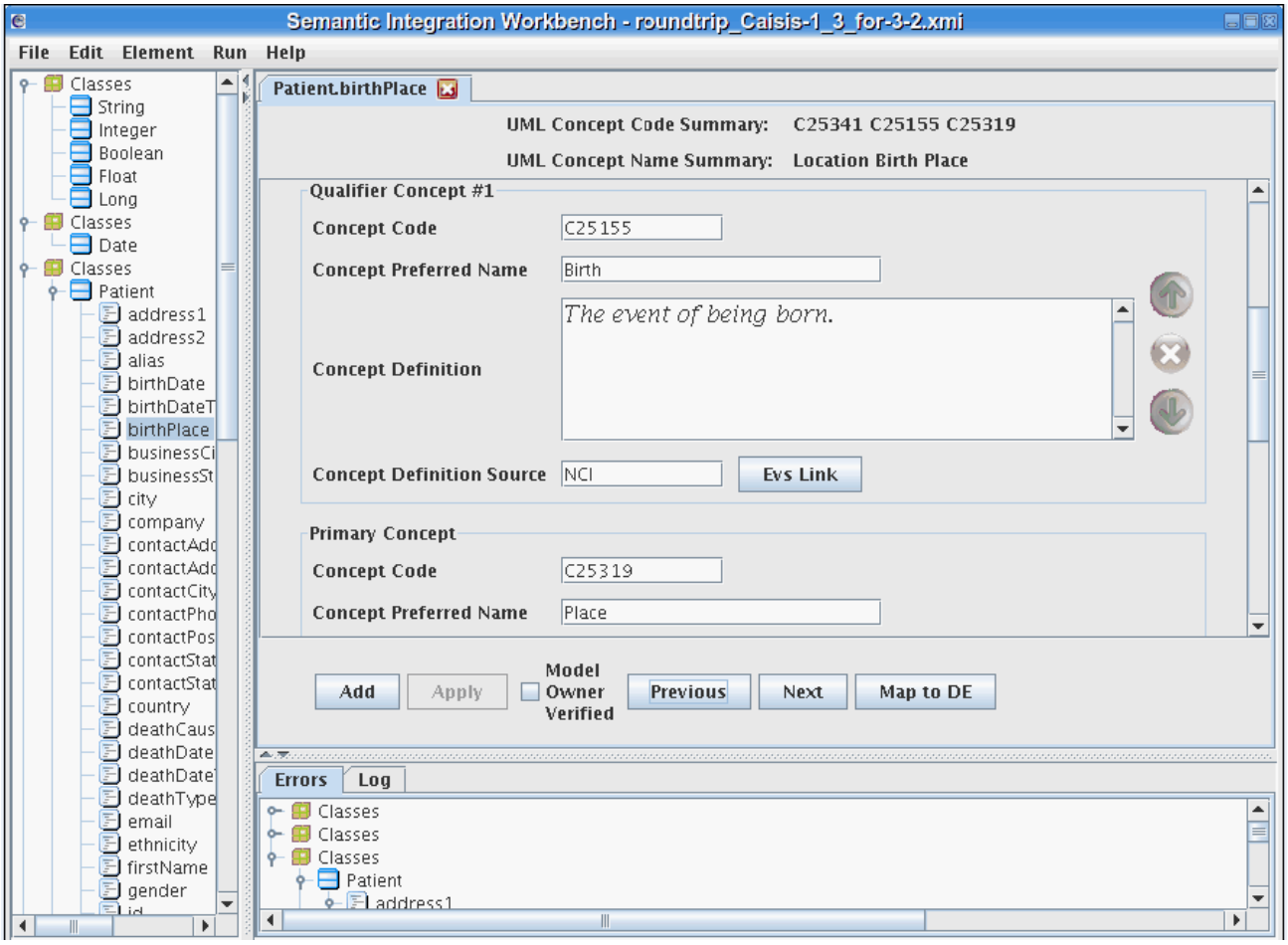


Figure 7.13 A UML Concept Code Summary pane displays concepts when they are available

Once a node has been selected, a set of buttons below the concept information displays (*Figure 7.13*). These include buttons that govern browsing, as well as other features.

- **Previous** selects the node in the tree previous to the current node.
- **Next** selects the next node in the tree after the current node.

The other buttons apply to editing annotations and are discussed in *Editing Annotation Details* on page 111.

Note: The **Previous** and **Next** buttons are not available if an Association Node is selected.

Annotation Basics

Each UML element, classes as well as attributes, must be annotated with at least one concept. The target and source ends of associations may also be annotated, although this is not a requirement. In all cases, a concept is made up of the fields described in Table *7.1*.

Concept Fields	Example
Code	C16612
Name	Gene
Definition	The physical and functional unit of ...
Definition Source	NCI-GLOSS

Table 7.1 UML element concepts

Additionally, elements can have UML Descriptions. Those are provided by the model owners as tagged values in the UML; ‘documentation’ for classes and ‘description’ for attributes. They can be used by the EVS concept curators to understand the model owner’s intended meaning of an attribute or class. If the caCORE SDK is used to create the public APIs for your software system, these tags become part of the JavaDocs. The description tag for attributes should be within the specific context of the containing class, as opposed to a generic description of the attribute. For example, the description tag for the attribute 'name' in a class 'Gene' might be 'The name of the Gene' as opposed to just "the words by which something is known".

Identifying Errors in the Source File

One valuable feature of the SIW is the ability to quickly isolate and identify errors in the source files, primarily missing or inaccurate information. Errors in a file display in the **Errors** tab at the bottom of the browser window (*Figure 7.14*). A description of the error identifies the source of the error.

- Errors relevant to a Class are shown as a child node to that Class's node.
- Errors relevant to an Attribute display with the full path of the attribute, for example, Package/Class/Attribute/Error.

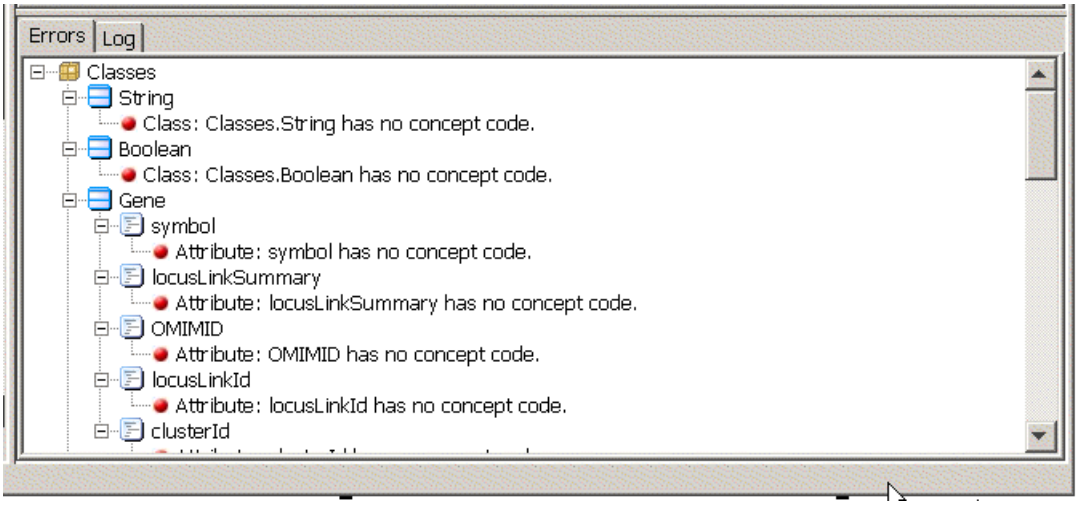


Figure 7.14 An Errors tab displays errors in a source file

All errors must be corrected before a model can be loaded to the caDSR.

Note: The errors are generated when you open a file; it does not update while you are editing. To regenerate the Errors tab, save your work and then exit the SIW. Re-open the SIW and review the newly generated errors.

Table *7.2* displays a list of possible errors that the SIW can report, along with a more detailed explanation and possible reasons which can correct the error.

Error	*Interpretation*	*Reasons*
"XYZ" has no concept code.	This error means the Semantic Connector was not able to attach concept information to this particular node.	• The EVS curation process was not completed.
Datatype "ABC" is invalid for attribute "XYZ".	The model owner has used a datatype for his model that does not conform to the caCORE SDK Toolkit convention.	• The modeler has chosen to model associations as attributes. For example, if the model contains two classes, 'Person' and 'Car' and a person 'has' cars, this should be modeled as an association rather than an attribute. There should not be an 'ownedCars' attribute to class 'Person'. The SIW in this case would report that the Datatype 'Car' for attribute 'ownedCars' is invalid. • The modeler is using a synonym for a primitive type that is not recognized by the UML Loader. If this datatype is indeed valid, this should be reported back to the UML Loader maintainer so he can ensure this synonym will be recognized. • The modeler used a datatype that is not currently in caDSR. This should be carefully reviewed to determine whether this datatype should be added to caDSR.

Table 7.2 Possible SIW errors and their interpretations

Verifying the Curated XMI File

Once you have selected a node in the Navigation Tree, you can click the **Human Verified** checkbox that is located next to the **Previous** and **Next** buttons. (After the EVS curators have reviewed the XMI file, the XMI version should have all **Human Verified** checkboxes marked with a check mark.)

Notes:

- The checkbox is unavailable if any of the concept information is missing. This prevents anyone from verifying the review of an incomplete element.
- If the item you verified is an attribute, the node associated with the reviewed attribute in the Navigation Tree displays the **Reviewed** icon (), and the next node in the tree becomes selected for your review.

*This feature allows you to quickly review items without having to click the **Next** button repeatedly.

- If the verified item is a class, the **Reviewed** icon () displays only after all attributes under that class have been Human Verified.

Warning! Any item without a checked **Human Verified** will **not** be inserted into the XMI file.

Editing Annotation Details

EVS Curators have the authority to independently change any concept field, and they can use concepts that may not be in the NCI Thesaurus. On the other hand, model owners should only change concept information by picking an EVS Concept from the Thesaurus or entering one supplied by EVS. New EVS concepts cannot be verified using the EVS Link feature, so preferably those are only used by EVS curators during the Curate XMI File step, however, they can be added during the Review Annotated Model step.

Editing While Leaving the Concept Code Unchanged

Within a model, two concepts with the same concept code must have the same concept name, definition and definition source, as well. This means that when you change any of the properties of a concept, while leaving the concept code unchanged, the SIW will apply the property changes to all other UML Elements within the model that use the same concept code. For example, two classes 'Person' and 'Animal' have an attribute in common, for example 'Name', and that attribute is annotated with the following Concept:

- `conceptCode: C25192`
- `conceptPreferredName: Name`
- `conceptDefinition: A word or group of words indicating the identity of a person usually consisting of ...`
- `conceptDefinitionSource: NCI`

Changing the definition but not the code for the Attribute '`Person.name`' applies the changed definition to '`Animal.name`'. The concept code remains unchanged.

Editing While Changing the Concept Code

If the concept code is changed, only the edited element is modified. Other UML Elements that point to the initial concept code are not modified. Using the previous example, if a user selects a different concept for `Animal.name`, it does not affect the mapping for `Person.name`. To force the change to apply to all concepts with the same name, see *Applying Changes to All Similar Nodes* on page 113.

Adding and Removing Concepts Mapped to an Element

Concepts can be added to an element and removed from an element. When a node is selected, the **Add** and **Apply** buttons located below the Viewer panel do the following:

Add—adds an empty concept to the element.

Apply—applies all changes that have been made so far.

Remove—removes a concept regardless of where it is in the Summary panel. Select the **X** from the Concept Summary panel.

To add a concept, use the following steps.

1. Select an element, which enables the **Add** button.
2. Click the **Add** button.

   Once the **Add** button is clicked, an empty concept is added below the first concept. Additionally, the **Add** button and **Apply** button become unavailable. The **Apply** button remains unavailable until the new concept has all four concept fields filled out.
3. Enter the appropriate information for all four text fields, either through text entry or by finding an EVS concept to populate the field. See *Filling in Annotations from EVS* on page 113 for more information. Once the fields are filled, the **Apply** button is enabled.
4. Click the **Apply** button. This saves the element with the new concept. (The new concept code will not be added to the view until **Apply** is clicked.)
5. After clicking the **Apply** button, the **Add** button once again is enabled. Click the **Add** button to add another empty concept.

Notes:

- Once an element has more than one concept, up and down arrows display next to each concept. Click the arrow buttons to move a concept up or down in the list (*Figure 7.15*).
- No changes are saved unless **Apply** is clicked.

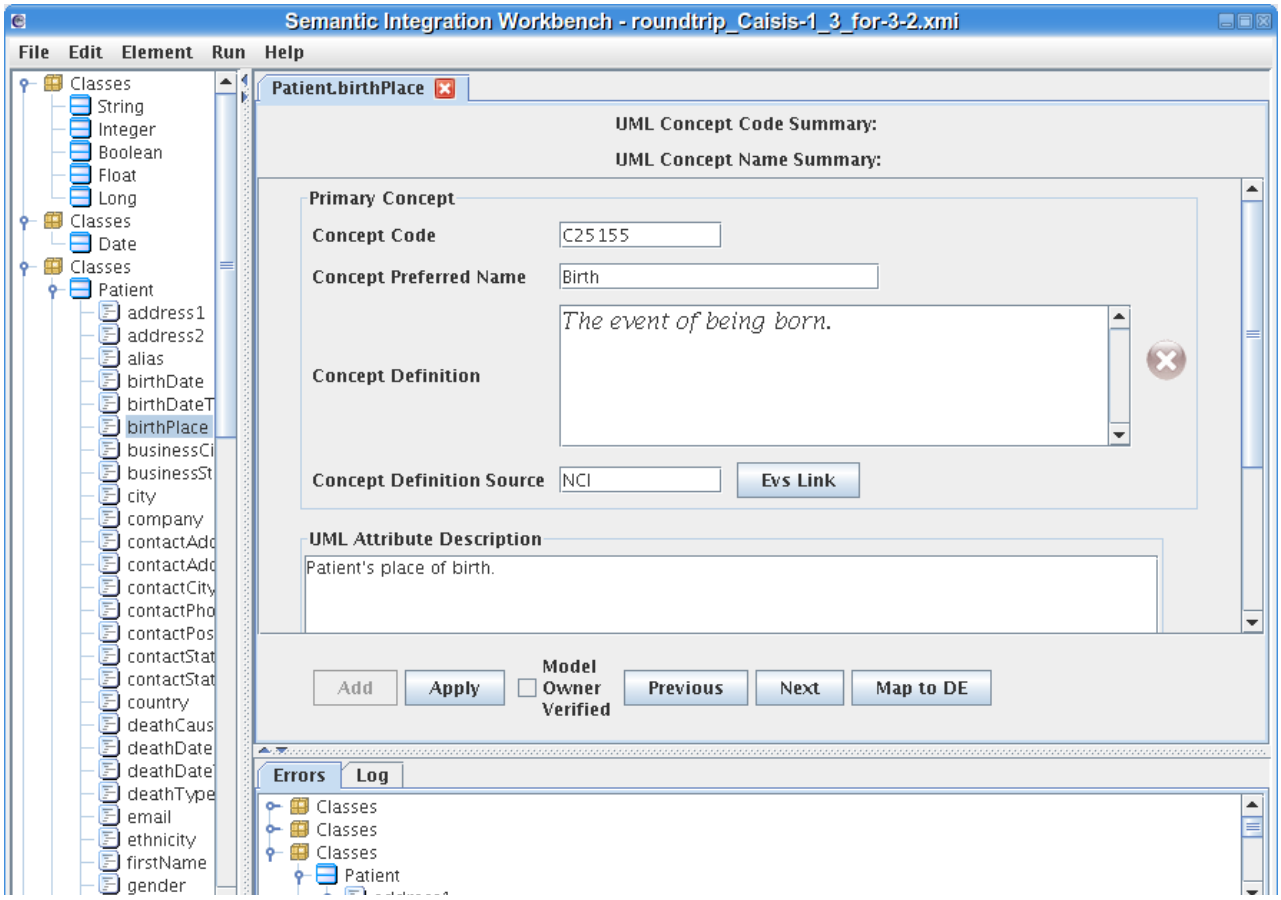


Figure 7.15 An attribute element with multiple concepts. The up and down arrows move the concept up and down in the concept list; the "X" removes it.

To remove a concept from the element, click the **X** located in the Concept Summary panel. This removes the concept listed.

Notes:

- The **X** button is always available.
- If you accidentally remove a concept you did not intend to remove, you can click **Previous** or **Next** without clicking **Apply** and you will be prompted regarding whether or not to save the change. Click **No** and the removed item will be restored.

Editing Text Fields

If any of the details in the concept fields are incorrect, you (curators only, please) can edit any text field. You can also modify concepts by performing a search in EVS. For more information, see *Filling in Annotations from EVS* on page 113.

Applying Changes to the Selected Node

Click the **Apply** button or select **Element > Apply** to apply the changes to a single selected element. Applied changes are not saved to the file. Use the Save functionality to save all applied changes to the file. See *Saving Changes to a File* on page 115.

Applying Changes to All Similar Nodes

In some cases, an attribute is reused several times across the model. A common example is the attribute `id`. Typically, you may want to modify the mapping for all attributes called `id` because it is likely that the modeler used this name with consistency and with a single meaning within the model. Once you have made changes, select **Element > Apply to All**. Changes are applied to all attributes with the same name.

Example:

If the attribute `id` is used 25 times across the model, and always means 'Identifier', except once, where it means 'Identified', select **Identifier**, then **Apply to All**, then modify the one term that is the exception.

Use the Save functionality to save all applied changes to the file. See *Saving Changes to a File* on page 115.

Filling in Annotations from EVS

From the Main Annotation panel, EVS Curators and model owners can modify a mapping by querying the NCI Thesaurus. They can either enter newly created concepts or select existing concepts from the NCI Thesaurus concepts using the EVS Link.

To begin the process, launch an EVS search by using the following steps.

1. Click the **EVS Link** button in the browser (*Figure 7.16*).

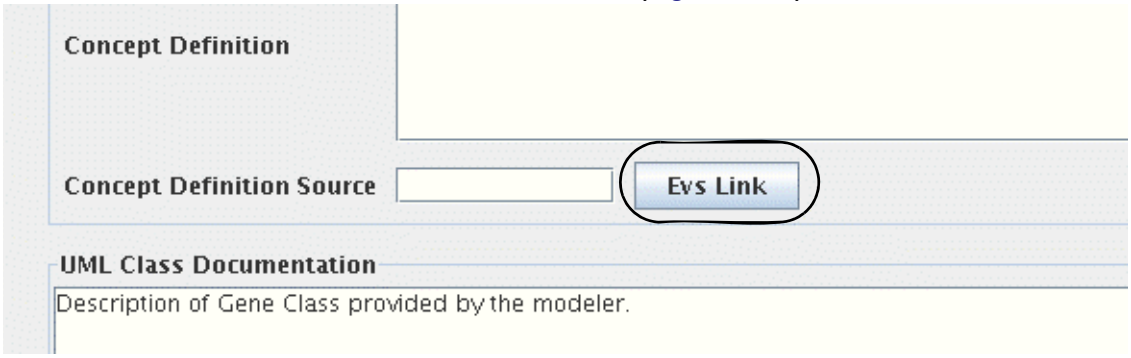


Figure 7.16 EVS Link button

2. This opens the Search Thesaurus dialog box (*Figure 7.17*). If the EVS link is selected with an existing concept, a search is performed automatically by default. This can be changed from the preferences, which are described in *Setting Preferences* on page 118. For a new concept search, enter the search term in the **Search** text box.

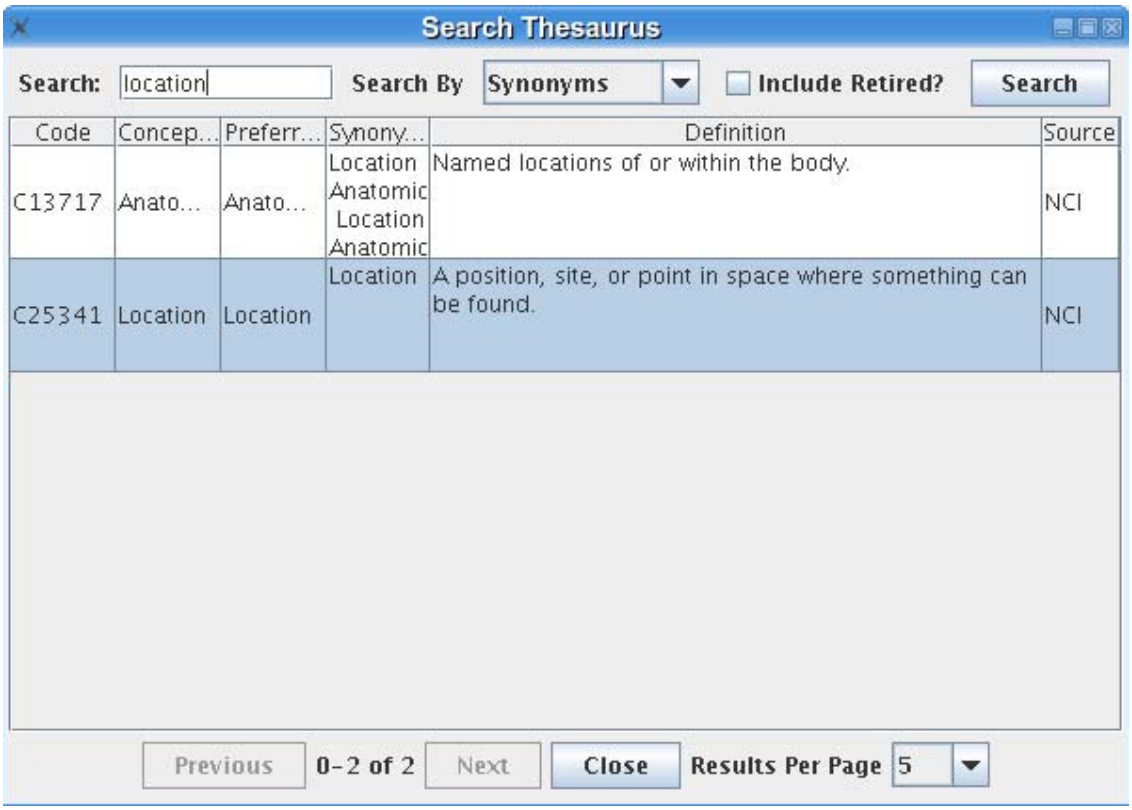


Figure 7.17 The EVS Search dialog box

3. In the **Search By** drop-down menu, select the method to be used for the search, **Synonym** or **Concept Code** As an example, a search for 'Location' will return the 'Location' concept because 'Location' is one of its synonyms.
4. Check the **Include Retired** box if you would like to include Retired items in your search. (Retired concepts are excluded by default.)
5. Click the **Search** button to launch the search that you have specified.

The results display in table format below the search options (*Figure 7.17*). If the concept name is not fully visible, hover your pointer over the item.

6. To select a search result, double click the appropriate line. This closes the dialog box and, in the SIW UML Concept Codes Summary, replaces the information for that class or attribute with the information from the EVS search.

Notes:

- If no matching terms are discovered in the search, then no Search Results displays and you will have to change the search to find matching items.
- Searches perform an exact match. The asterisk (*) can be used as a wildcard. Because the results are generally more inclusive, using a wildcard usually causes a search to take longer.

Viewing Search Results

The search result list is limited to 100 terms, with only five results shown per page by default. Change the default by selecting a value from the drop down next to the **Results per page** option at the bottom of the Search Results table. If more results are returned, the **Next** button is enabled, allowing you to easily walk through the results. Use the **Next** and **Previous** buttons to navigate through the search results.

Saving Changes to a File

Select **File > Save As** to specify a different name (not recommended) or location for the file.

Save the file with the same name as the input file, in the same directory (that is, `Reviewed_${filename}.csv`). This file is referred to as the Curated XMI file.

The Status Bar informs you that the file was successfully saved.

Note: The second run of the Semantic Connector is no longer necessary starting with caCORE 3.2.

The Curated XMI file is used as input to the last step, Reviewing an Annotated Model.

Reviewing an Annotated Model

During this review step in the semantic integration process, corrections can be made and/or verified in the Annotated Model XMI file. To view the completely annotated XMI file after this step, you must exit the SIW and relaunch the application from the URL.

After this review process is completed, the file is passed on to NCICB for logging into CVS and loading to caDSR.

To proceed through the review process:

1. Select **File > Exit**.
2. Launch the SIW again. Follow the steps described in *Running the Semantic Connector* on page 101.
3. Select the **Review Annotated Model** mode, and then click the **Next** button.
4. Select the appropriate annotated model XMI file. This must be the XMI file that is output after the Curate XMI file has been completed.

5. Click the **Next** button to continue. The next dialog box will inform you that the XMI file is being parsed.

   For each UML class and attribute, the SIW checks for the presence of at least one:

   - Concept code
   - Concept name
   - Concept definition
   - Concept definition source
   - Valid datatype

The SIW permits value meanings to be loaded without associated concepts. After this is completed, the SIW's Main panel displays. This panel looks and works in a very similar manner to the Main panel described in the section, *Curating XMI Files* that begins on page 104.

- The Navigation Tree displays model information on the left-hand side of the application. When you click a node in the tree, the concept information displays on the Concept Code Summary panel.
- The main difference between this viewer and that of the Curate mode is that the editor is working here using the final annotated XMI file, not the file created by EVS. This XMI file represents all the tag names and items from the UML Model that are needed to load into caDSR. Additional validation can be performed using this XMI, such as validating datatypes and viewing Associations, which was not possible using the Curate XMI mode. The annotated XMI file is the file that will be used for loading into caDSR.
- Adding concepts using the **Add** button works in the way described in *Adding and Removing Concepts Mapped to an Element* on page 111. The **Remove** and **Apply** button work in the same way as described in the same cross reference.
- Each item in the Annotated XMI Model will again be set to **Not Human Verified.** You (the Model Reviewer) can mark Human Verified by clicking the **Human Verified** checkbox as each item is reviewed.

Errors Tab

The Errors tab in the Review mode displays a tree structure similar to the main tree. It identifies where in the tree errors occur. If there are no errors, the tree does not display. As mentioned previously, the errors do not update until the file is saved and reopened. The errors file can be printed by right clicking on the Errors tab and selecting **Export Errors**.

Viewing Associations

In the Review mode, Associations display at the bottom of the tree. Association information displays in the Main panel as well when an Association is selected in the Navigation Tree.

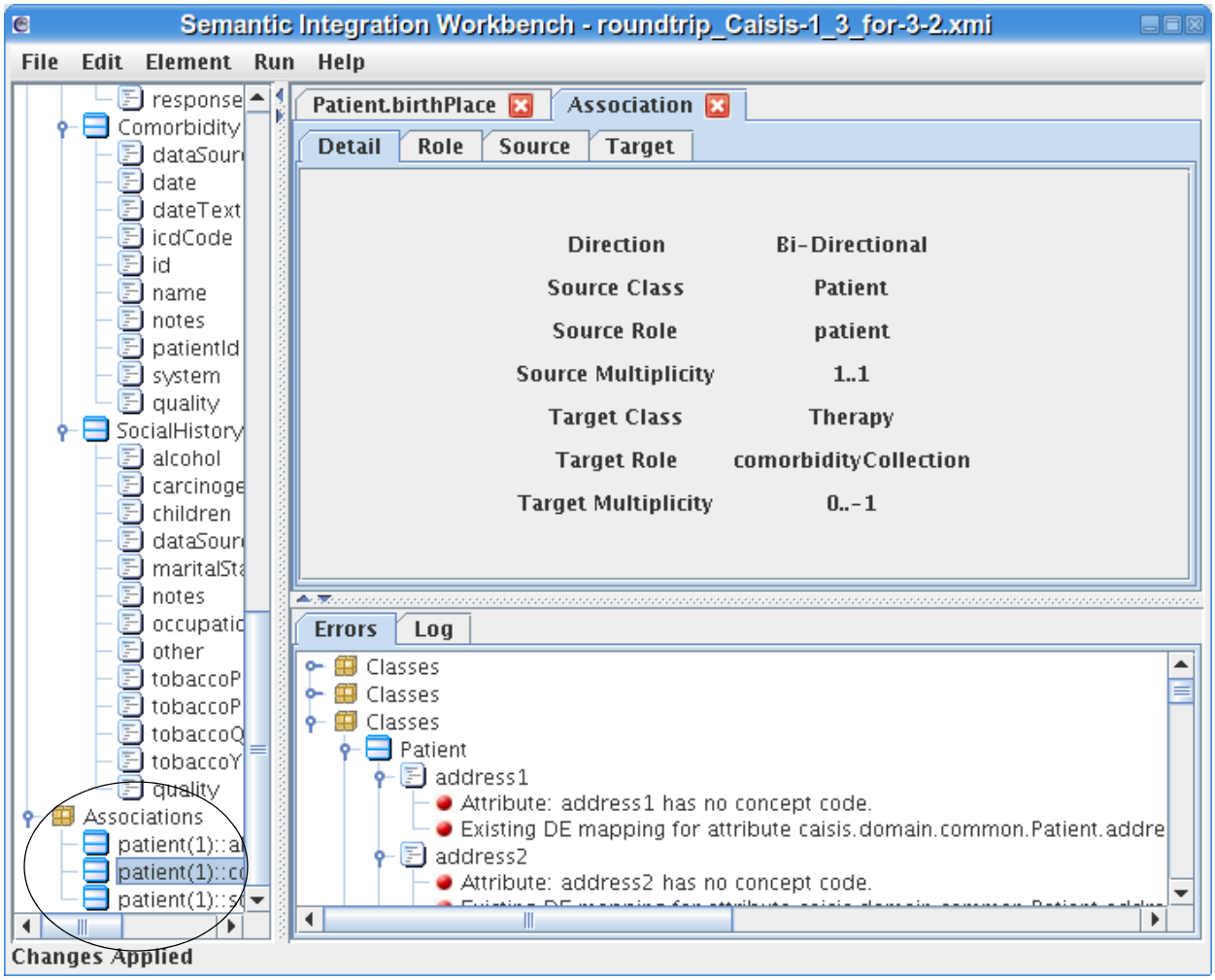


Figure 7.18 In Review mode, associations display at the bottom of the Navigation Tree.

- Click on the node in the Associations part of the tree and an **Associations** tab displays in the Main Viewer panel describing all of the details of the selected association.
- The information in the Associations tab can be modified to add optional concepts at the role, source, or target level.
- To close the Associations tab, click on the red **X** at the top of the tab. This returns the display to the previous tab, if applicable.
- Associations can also be viewed as part of the model tree by selecting **Edit > Preferences**. In the Preferences dialog box, the first option allows you to view associations in class tree. Click this option and then click **OK**. This closes the Preferences dialog box and displays associations as part of the tree.

Setting Preferences

Select **Edit > Preferences** to open a Preferences dialog box, where you can review or change the Preferences settings. Preferences are persistent across sessions, although the Preferences options vary slightly depending on the SIW mode that is activated.

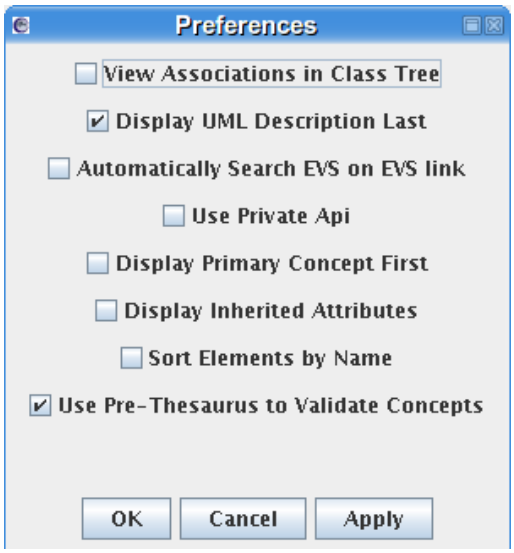


Figure 7.19 Preferences dialog box in SIW Review mode

Several preferences can be set in the SIW. Depending on which mode you are in, not all options are available in all modes, but when present they function the same way. Preferences are:

1. View Associations in the Class Tree
2. Display UML Description Last
3. Automatically Search EVS on EVS Link (in Review Annotated Model mode)
4. Use Private API
5. Display Primary Concept First
6. Display Inherited Attributes
7. Sort Element by Name
8. Use Pre-Thesaurus to Validate Concepts

Viewing Association

When viewing associations, four tabs display in the Association View panel: Role, Source, Target, and Detail. Associations can also be annotated with concepts. Their annotation can be independently done at three different levels.

1. Role Annotation

   This step may be completed to annotate the nature of the association itself. Select the **Role** tab to annotate the association.

2. Source Annotation

This step may be completed to annotate the nature of the source end of the association. Select the **Source** tab to annotate the source end.

3. Target Annotation

   This step may be completed to annotate the nature of the target end of the association. Select the **Target** tab to annotate the target end.

4. Select the **Detail** tab to get back to the detail view of the association.

Like all other concept annotations, there can be a primary concept plus qualifiers. For reference, the following tagged values are used:

1. To annotate the role:
   - AssociationRoleConceptCode
   - AssociationRoleConceptPreferredName
   - AssociationRoleConceptPreferredDefinition
   - AssociationRoleConceptDefinitionSource
   - AssociationRoleQualifierConceptCode(n)
   - AssociationRoleQualifierConceptPreferredName(n)
   - AssociationRoleQualifierConceptPreferredDefinition(n)
   - AssociationRoleQualifier(n) ConceptDefinitionSource
2. To annotate the source:
   - AssociationSourceConceptCode
   - AssociationSourceConceptPreferredName
   - AssociationSourceConceptPreferredDefinition
   - AssociationSourceConceptDefinitionSource
   - AssociationSourceQualifierConceptCode(n)
   - AssociationSourceQualifierConceptPreferredName(n)
   - AssociationSourceQualifierConceptPreferredDefinition(n)
   - AssociationSourceQualifierConceptDefinitionSource(n)
3. To annotate the role:
   - AssociationTargetConceptCode
   - AssociationTargetConceptPreferredName
   - AssociationTargetConceptPreferredDefinition
   - AssociationTargetConceptDefinitionSource
   - AssociationTargetQualifierConceptCode(n)
   - AssociationTargetQualifierConceptPreferredName(n)
   - AssociationTargetQualifierConceptPreferredDefinition(n)
   - AssociationTargetQualifierConceptDefinitionSource(n)

The Association View Preferences option is available only in the **Review Annotated Model** mode (*Figure 7.20*). See *Viewing Associations* on page 116 for more information about this view.

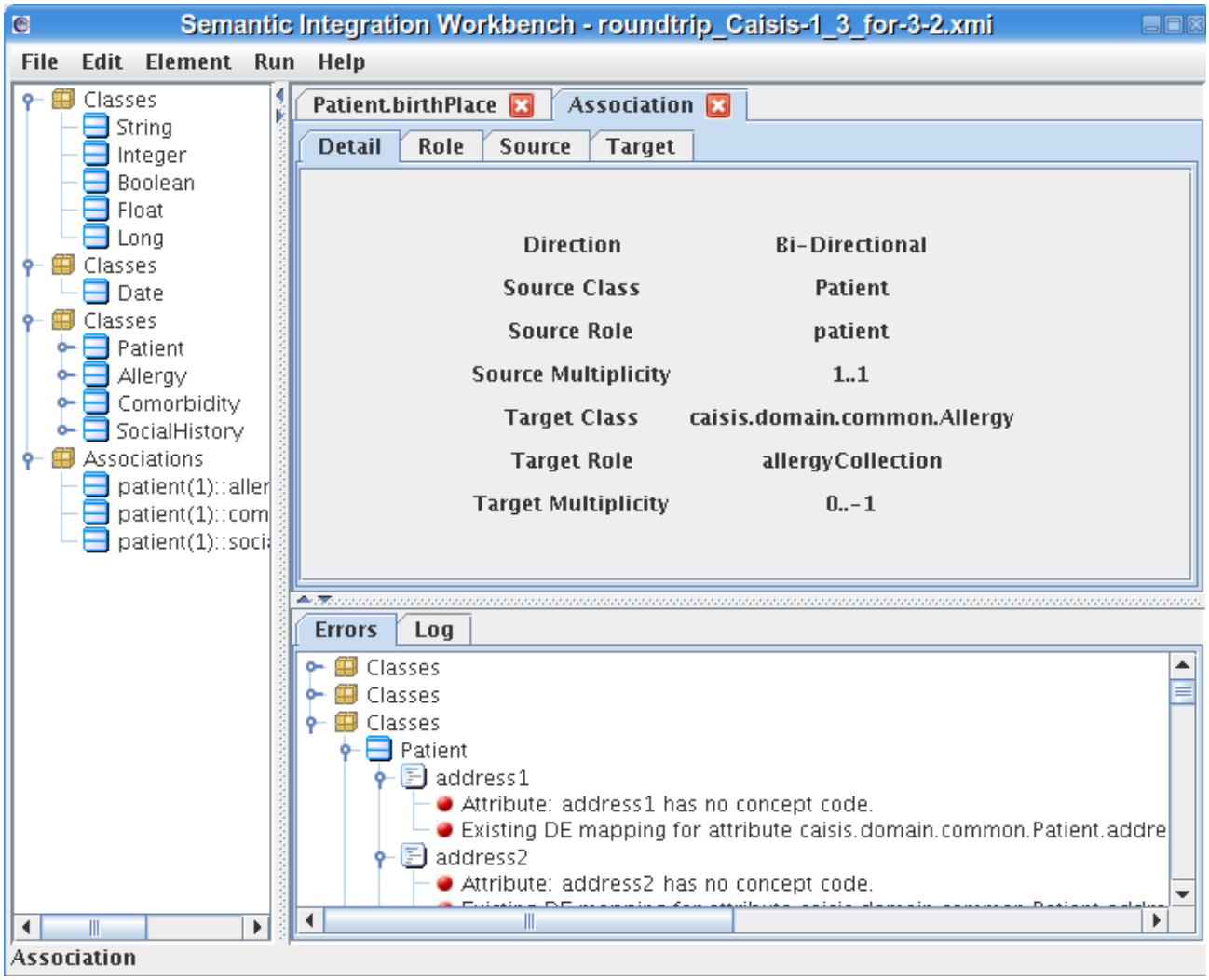


Figure 7.20 In Review mode, associations display at the bottom of the Navigation Tree. Association information displays in the Main panel as well, when an association is selected in the Navigation Tree.

Domain object associations display on the Navigation Tree in two ways:

- With the default setting, each Association displays as a separate node at the bottom of the Navigation Tree. This allows you to review annotations more efficiently.
- To view associations as children to classes, check the **View Association in the Class Tree** option of the Preferences dialog box. This is useful when trying to identify associations for a particular class.

UML Description

The SIW displays UML model descriptions whenever they exist. These are the Documentation and Description tagged values described in this guide and are mandatory.

The **Display UML Description Last** option in the Preferences dialog box allows you to display a description either preceding or following the concept mapping information.

- The default setting is to display UML Descriptions preceding the concept mapping.
- To view UML Descriptions following the concept mapping information, clear the the **Display UML Description Last** option in the Preferences dialog box.

Search EVS

By default, when you click on the **EVS Link** for an existing mapped concept, the SIW automatically executes an EVS search using the Concept Preferred Name. If the concept is new, using the **Add** feature, a search term must be entered in the EVS Search panel that displays after clicking the EVS Link.

When this preference is not available, the search dialog box opens, but no search will be run without user input.

- To change this setting, clear the **Automatically Search EVS on EVS Link** option in the Preferences dialog box.

Use Private API

This API performs slightly faster than the public API, but you must be connected via VPN or behind the firewall on the local NCI LAN to use it.

Display Primary Concept First

The default behavior is to display the primary concept last so that the string of terms mimics the way the name will be created in caDSR. This option will change the order of the concatenated concept codes so that the Primary concept is displayed first.

Display Inherited Attributes

This option displays the inherited attributes in the SIW Navigation panel for any class that inherits from a super class. The attributes are displayed in a node entitled "Inherited Attributes". These attributes should be annotated in the parent/super class, not in the "Inherited Attributes" node.

Sort Element by Name

This option sorts the classes in the Navigation panel by name. Deselecting this feature resets the display of classes to their order in the XMI file.

Use Pre-Production Thesaurus to Validate Concepts

By default, SIW searches NCI Production Thesaurus. This option causes the SIW to use the Pre-Production NCI Thesaurus, which is an early preview of the next release of the NCI Thesaurus, publicly accessible from the DTS server. The Pre-Production Thesaurus is more recent than the Production version and thus newer concepts will be found, less 'not found' errors produced when the "Validate EVS Concept" step is performed.

Setting UML Loader Run-Time Parameters

This option is only available in **Review Annotated Model** mode.

- Several run-time parameters can be set. Those parameters are later used to load the model to caDSR.
- You can enter data relevant to the model you are loading. This information is also provided as part of the submission document.
- The package filter field is used if you want the SIW to only parse certain packages. If this field is left blank, all packages will be parsed. This may be an issue if your XMI file contains objects that are not part of your domain model, such as data model elements.

The format for the package filter field is as follows:

```
<My Alias One>my.package.one, <My Alias Two>my.package.two
```

In the above example, only `my.package.one` and `my.package.two` will be parsed. Other packages will be ignored.

To set the run-time parameters, follow these steps:

1. Select **Run** > **Defaults.**
2. Fill in all fields from the model submission template (*Figure 7.21*).
   a. **Project Name** is the Classification Scheme Preferred Name specified in the model submission template.
   b. **Project Long Name** is the Classification Scheme Long Name specified in the model submission template.
   c. Entries for **Package Filter**, as described above, should be separated by brackets and commas. If no names are entered, all packages will be assigned one Classification Scheme Item equal to the Project name. The format for the package filter field is as follows:

      ```
      <MyAliasOne>my.package.one, <MyAliasTwo>my.package.two
      ```

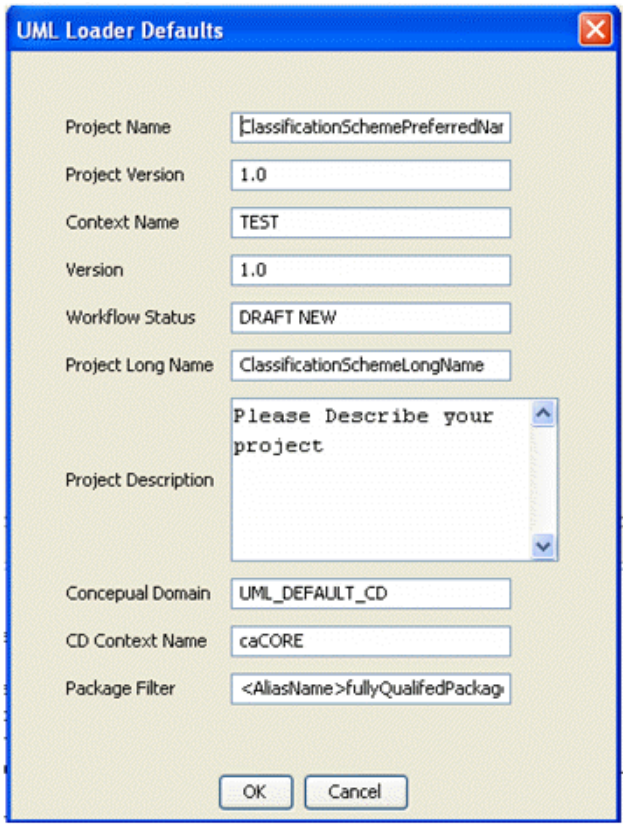


Figure 7.21 UML Loader default settings

Updating UML Model Definitions

One problem frequently encountered in the semantic integration process is the discovery that the documentation and description tags from the UML Model are not present, or that they need to be updated. Prior to the SIW, there was no way to conveniently get these changes into the XMI file for loading to caDSR. Since these tags are mandatory, this process must be used to create or update these tags.

With the SIW and careful planning, you can change or update the UML documentation or description tags in the UML model in Enterprise Architect, export them to XMI, and go back through the process, while preserving the EVS concept curation before loading the model into caDSR.

Updating UML Model Definitions Workflow

In summary, the workflow is as follows:

1. The new UML documentation and description tagged values have to be added using EA. Simply import the XMI file you have been working with in SIW, add the needed information, and then the export the model as XMI again.
2. The SIW makes changes directly to the Annotated XMI file that is used to load to caDSR.

3. Launch the SIW and select **Review Annotated Model**. You will now see the updated UML documentation and description tags associated with each class and attribute.

Errors and Log Tabs

The Errors Tab

Validation Errors

There are several validation rules that are applied to each model. Any violations of the rules are displayed in the tree that appears in the Errors tab and can be exported by right clicking on the Errors tab and selecting **Export Errors**.

Duplicate Errors

If two Object Classes have the same Public Id or Preferred Name then a duplicate mapping error will occur. If two Data Elements belonging to the same Object Class have the same Preferred Name then a duplicate mapping error will occur.

Association Errors

For associations, a missing Source Role or Target Role causes an error. A missing Source Role name or Target Role name also causes an error.

Value Domain Errors

If definition, Vdtype, datatype, Public Id, or Version are either missing or invalid then an error will be displayed. If a VD is in the model but is not used by a DE in the model then this causes a warning message to be displayed. Finally, if two VDs are mapped to the same concepts this causes an error as well.

The Log Tab

The Log tab is a new feature of the SIW and displays detailed parsing and runtime information. Each log event is represented as a line in the Log tab. Log events are organized into four categories:

- Errors: Errors either in the SIW or in the model being parsed.
- Warnings: Warnings of validations that may be incorrect and should be reviewed.
- Info: Events that may be helpful to users
- Debug: Information that is most useful for developers.

Select or deselect any checkbox to view or ignore any event category (*Figure 7.22*).

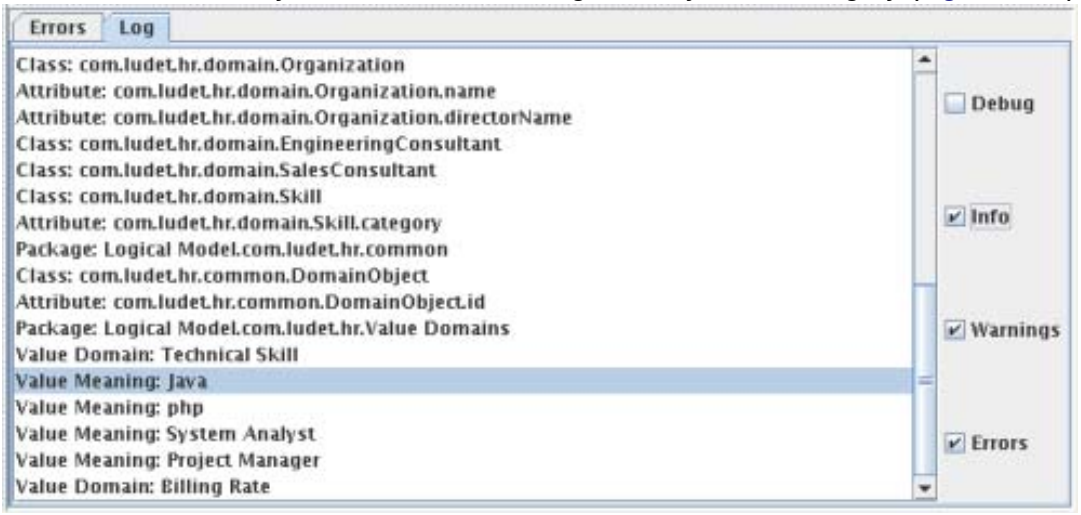


Figure 7.22 The Log tab displays detailed parsing and runtime information

Mapping UML Attributes

Mapping a UML Attribute to an Existing Common Data Element

After opening a model in the mode Review Annotated Model, you can map any attribute to an existing Common Data Element (CDE) with the following steps.

1. Begin by clicking on any attribute in the tree (*Figure 7.23*). This displays the concept information in the View panel to the right of the tree.

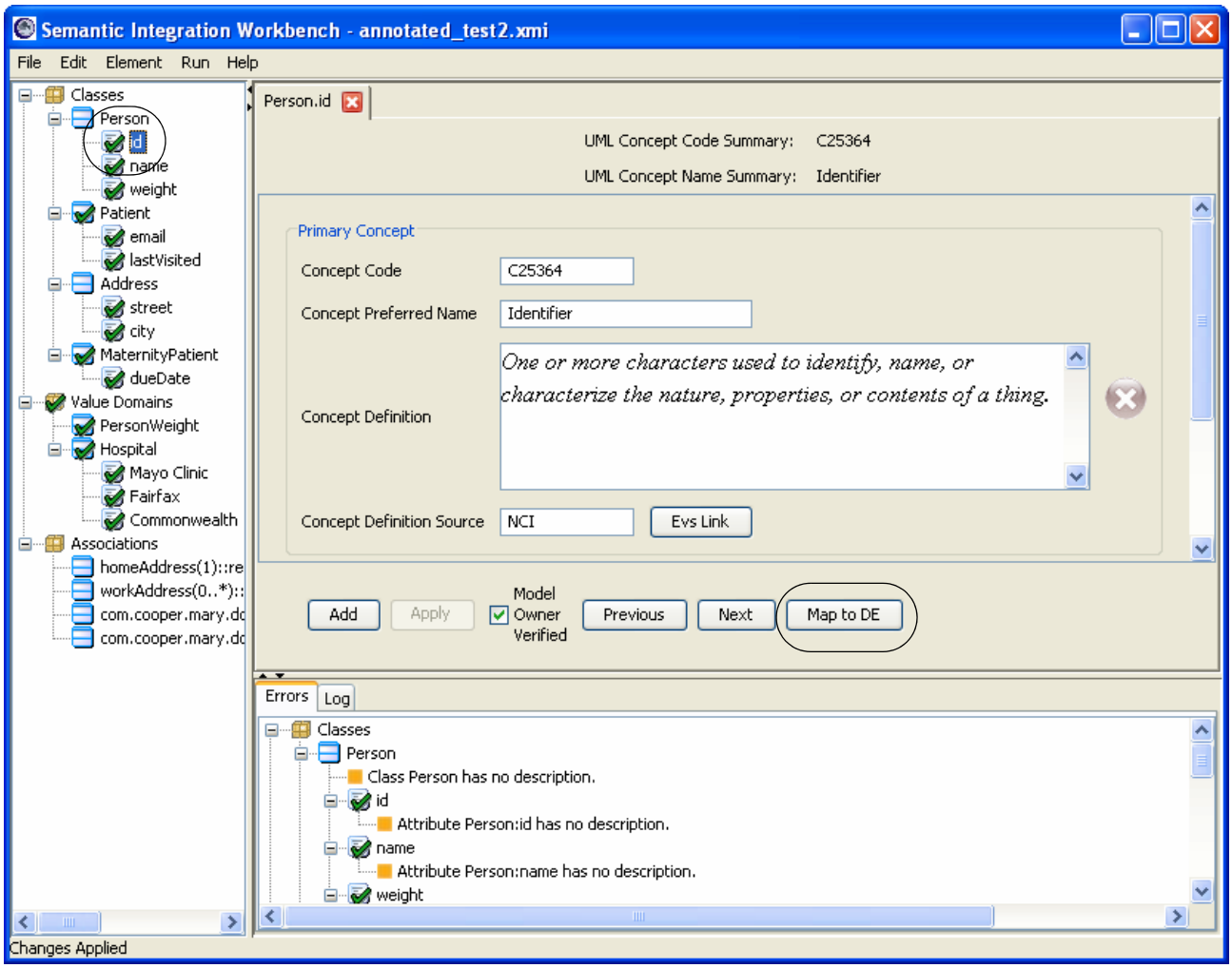


Figure 7.23 Displaying an attribute for mapping to a CDE

2. Directly below the View panel is a row of buttons that includes the **Map to DE** button. Click this button to change the View panel to display Data Element information. Instead of showing which concepts the attribute is mapped to, it displays which DE the attribute is mapped to (*Figure 7.24*).

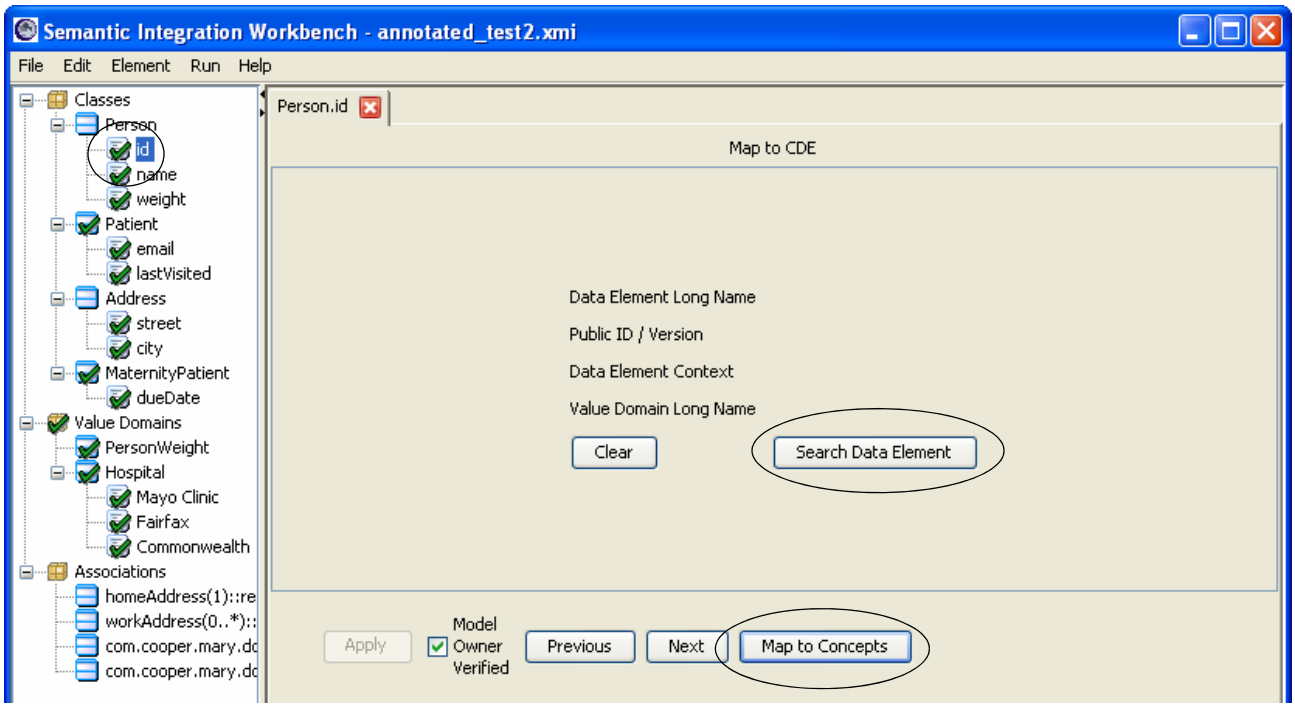


Figure 7.24 Mapping an attribute to a CDE

3. Since the attribute is not mapped initially to a Data Element, all of the fields are empty. The buttons also change to reflect the options that are available when mapping to a DE. Mapping to a DE is not available if the attribute uses a local Value Domain. See *Pointing a UML Attribute to a Value Domain* on page 136.

4. Select a CDE by clicking the **Search for Data Element** button, which opens the Search for Data Element dialog box (*Figure 7.25*). This search can take advantage of the Freestyle search engine which performs a search on the caDSR (http://freestyle.nci.nih.gov/freestyle/do/search). Enter a term in the Search field and click **Freestyle Search**. You can also click **Suggest**.

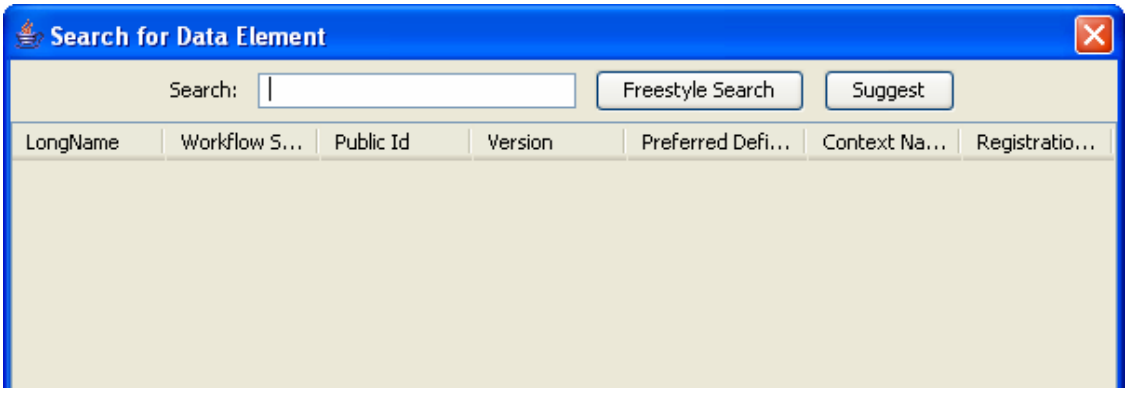


Figure 7.25 Search for Data Element dialog box

5. Results are returned in the table below the search term (*Figure 7.26*). Below the table, use the **Previous** and **Next** buttons to flip to the next or previous pages if

results are displayed over multiple pages. The numbers between the Previous and Next buttons indicate which records are currently being viewed.

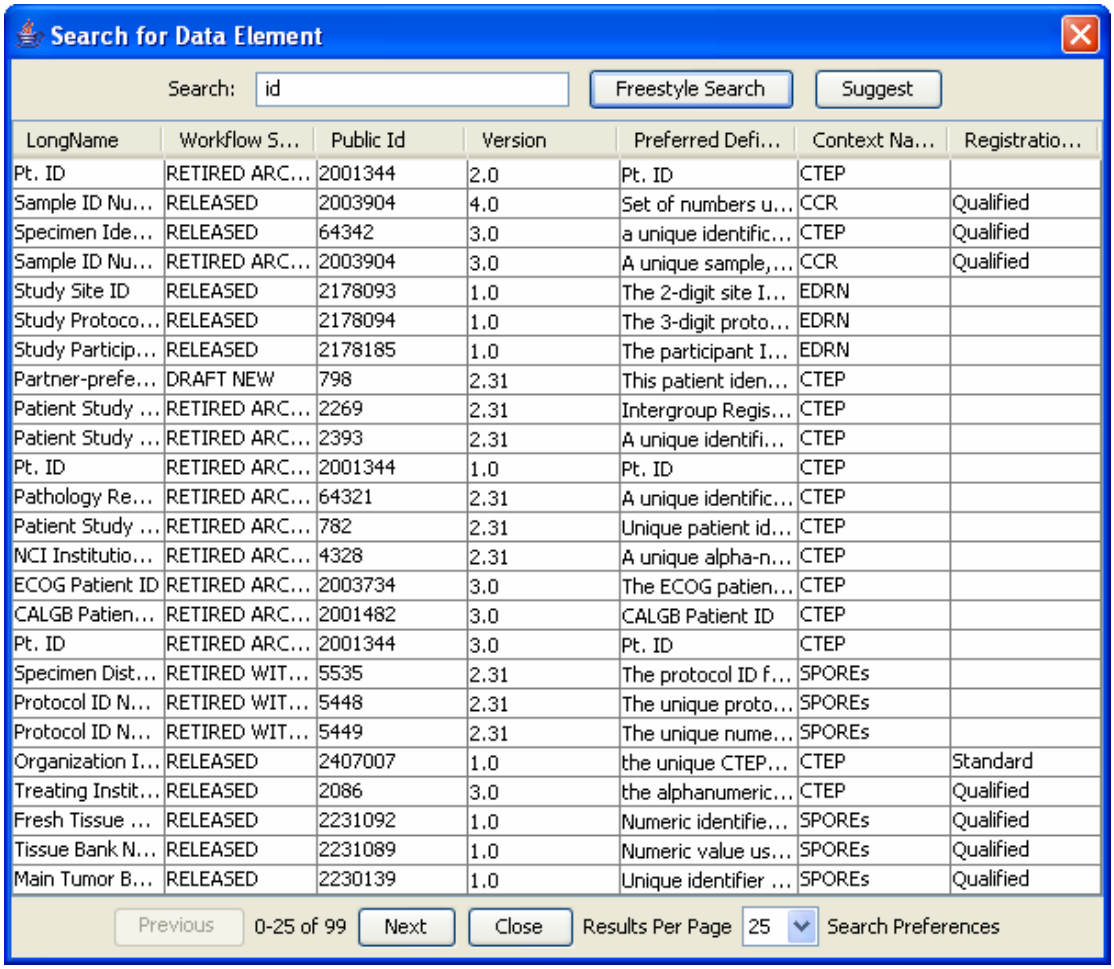


Figure 7.26 Searching caDSR for existing CDEs

6. The Close button closes the dialog box without selecting any DE. By double-clicking on one of the DEs the dialog box is closed and the information from that DE is populated into the View panel (*Figure 7.27*).

Note: You may receive an invalid selection error that informs you that the Data Element that you have selected is not mapped to an Object Class or Property. This data is not correct and cannot be used for this mapping. Another possible error is that mapping this attribute to this DE might cause a duplicate mapping. A

duplicate mapping is where a different attribute has already been mapped to this DE.

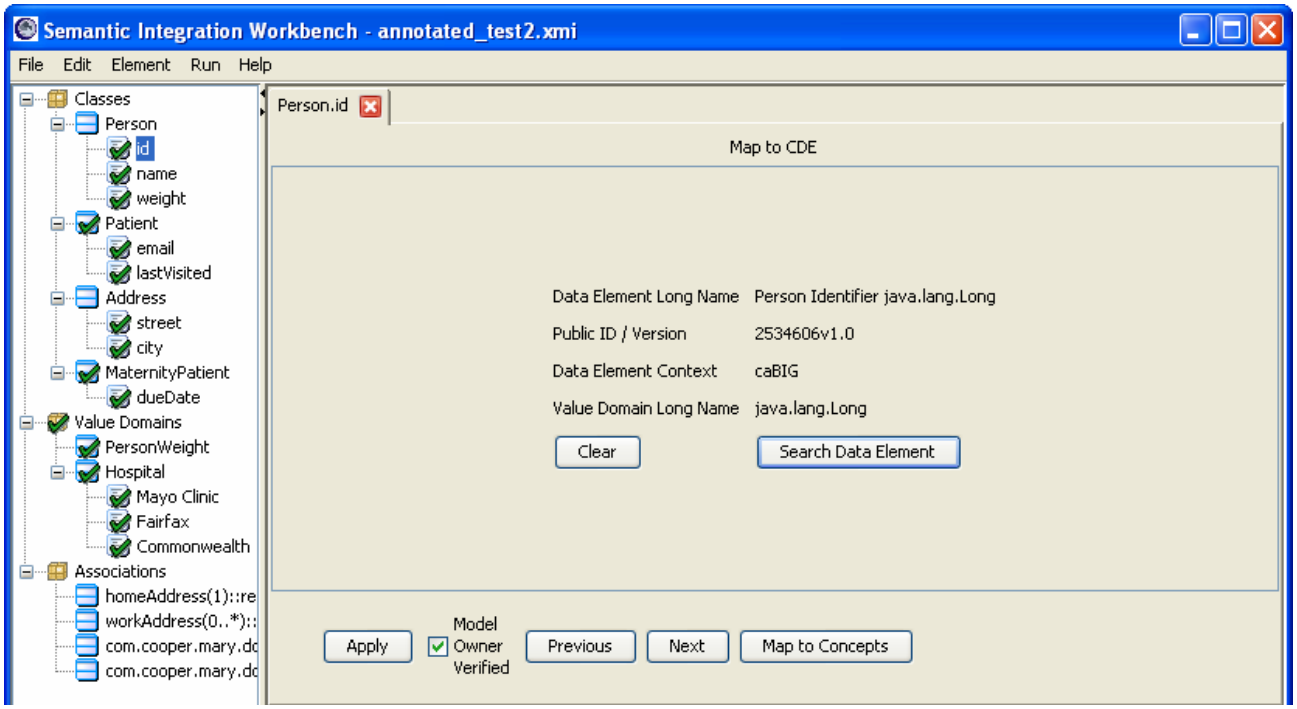


Figure 7.27 An attribute is now mapped to an existing CDE

If you select a Data Element that is valid, then all of the Data Element's information is entered in the View panel. Options include:

- Click **Clear** to undo the choice of the DE. This removes the DE information from the View panel.
- Click **Apply** after clicking the Clear button to undo the process of mapping to a DE.
- Click **Apply** to complete the process of mapping the attribute to a DE. The Map to Concepts button is not available after clicking Apply when a DE has been selected. At all other times it is available. You can switch back to concept information in all cases except when the attribute has been mapped to a DE. The DE mapping overrides any concept information.

When an attribute is mapped to a DE, the class that the attribute belongs to is mapped to the Object Class that the DE is mapped to. This is displayed in the View panel if the user selects a class that has an attribute that is mapped to a DE. Instead of the concept information the Object Class information is displayed in the View panel for that class (*Figure 7.28*). A warning message displays when this occurs.

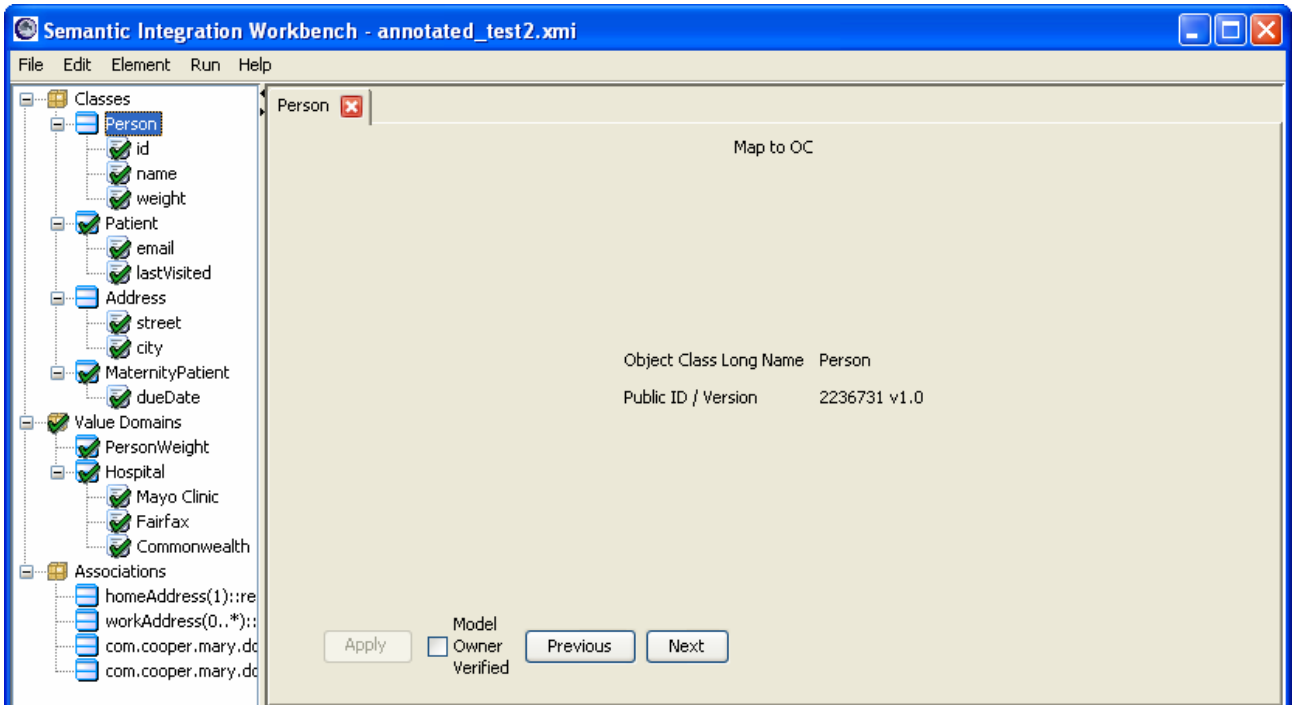


Figure 7.28 Mapping an attribute automatically maps its class

Mapping a UML Attribute to an Existing Value Domain

If an attribute is not mapped to a DE then the concept information is displayed in the View panel. Below the concept information is a section for selecting a Value Domain (*Figure 7.29*).

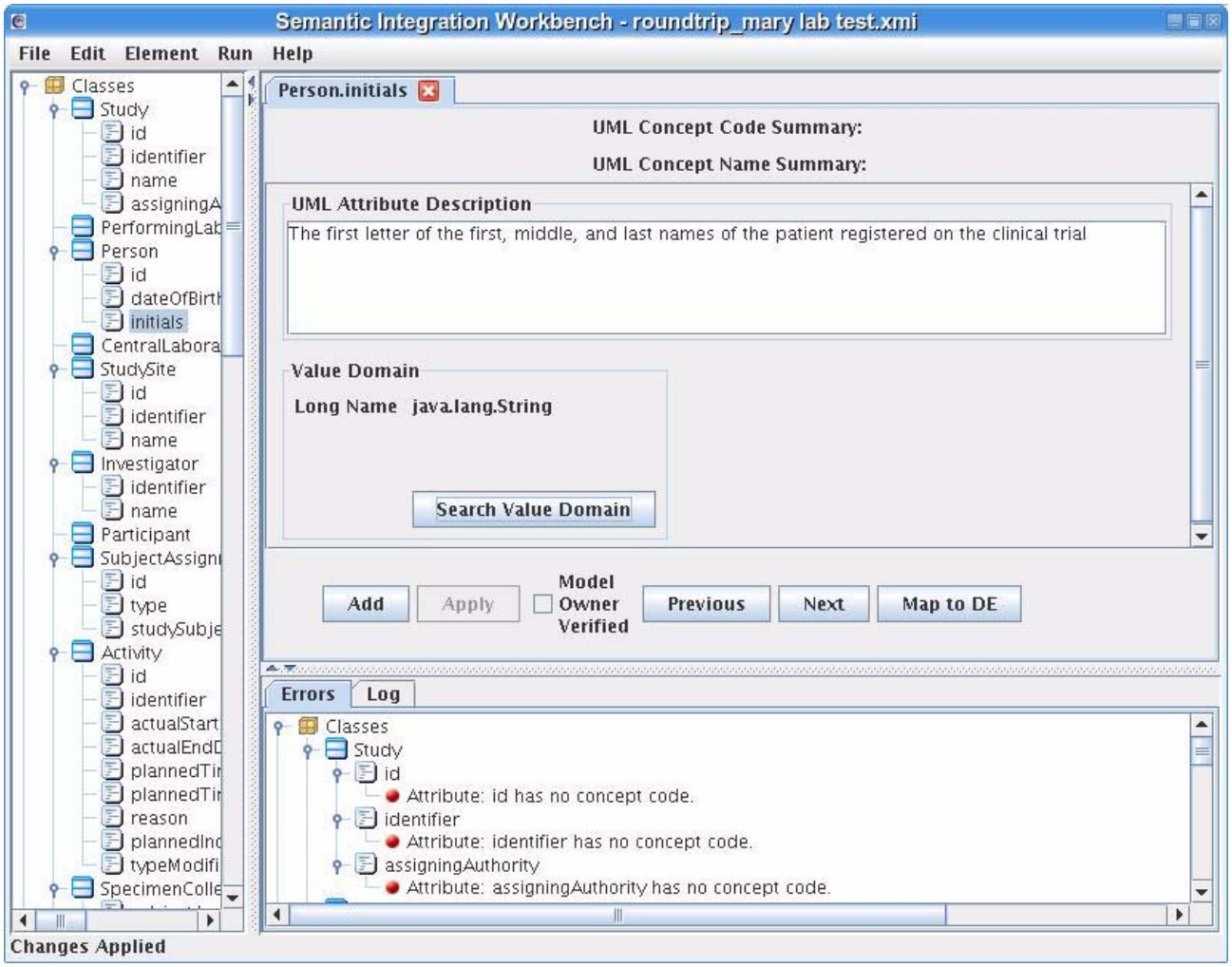


Figure 7.29 The Map to existing Value Domain box is located below the concept annotation area

By clicking the **Search Value Domain** button, the **Search for Value Domain** dialog box opens (*Figure 7.30*). This dialog box is similar to the Data Element dialog box except that it searches for Value Domains instead.

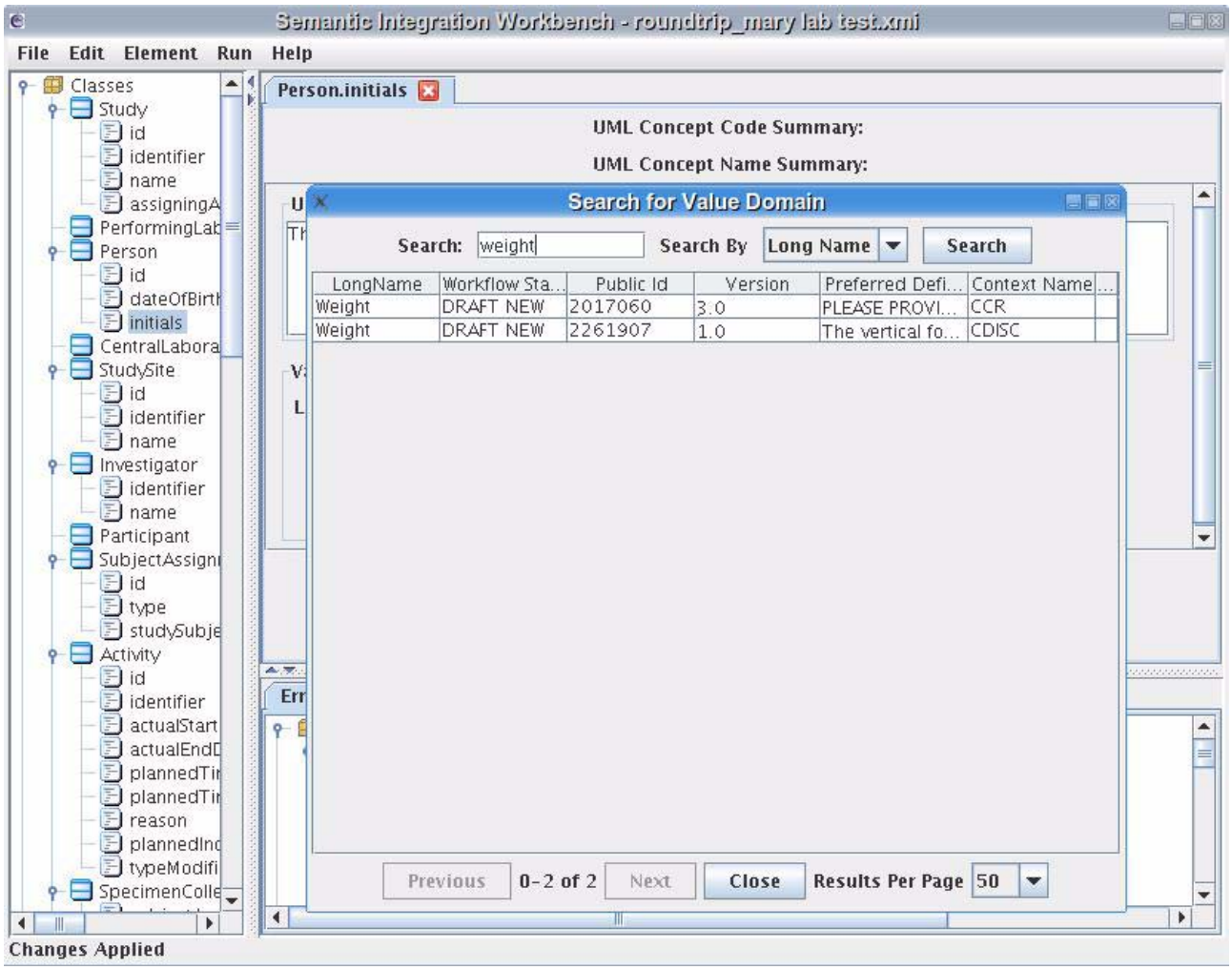


Figure 7.30 Searching caDSR for existing Value Domains

After performing a search, select a Value Domain by double clicking it. This closes the dialog box and places the Value Domain information in the View panel. The process is not complete until you click the **Apply** button. Otherwise the new VD is not applied to this attribute.

Validating Concept Mappings Against EVS

SIW provides the capability of validating the concepts against EVS by using the Validate Concepts option, run by selecting **Run > Validate Concepts**. A dialog box opens informing you that this process could take some time. It also asks if you want to continue. If you select No, the box closes and no validation occurs. If you select Yes, the Validate Concepts dialog box opens (*Figure 7.31*). A progress bar opens on the bottom of the dialog box indicating the progress of the validation. It also displays which concept is being validated directly above the progress bar. When the process is complete, **Done** displays below the progress bar. The left-hand side of the dialog box displays a list of

classes and attributes that are in the model but whose concepts are not exactly the same as they are in EVS. Each item of the list can be selected by clicking on it.

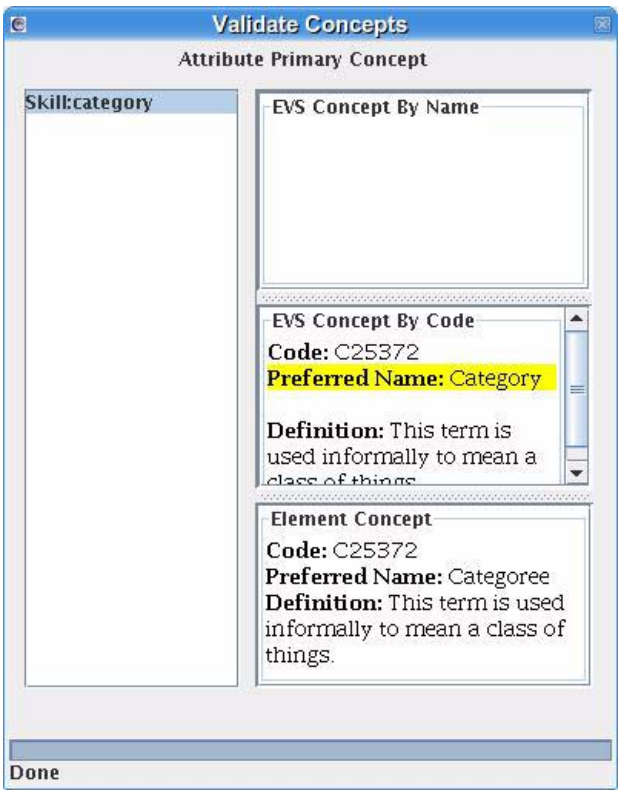


Figure 7.31 The EVS validation window displays discrepancies between the concepts in EVS and those in the XMI file

When an item is selected, the three boxes on the right-hand side of the dialog box are populated.

- The first box, EVS Concept By Name, searches EVS for a concept with the same Preferred Name and displays the result.
- The second box, EVS Concept By Code, searches EVS for a concept with the same code and displays the result.
- The third box, Element Concept, is the concept as it appears in the model.

The first two boxes can be empty, which means that no match was found. The information displayed in these three boxes is the Code, Preferred Name, and Definition. These three fields can be highlighted in the EVS Concept By Name and EVS Concept By Code box. A highlighted field means that this part of the concept is different in EVS than what you have in the Element Concept box. One final feature is that when a class or an attribute is selected in the list, the tree in the main window also selects the node that corresponds to it. This allows easy review of the concepts for that node.

Creating Value Domains

Besides being able to point a UML attribute to a specific caDSR Value Domain, model owners have the ability to create a Value Domain. Creating a Value Domain is done from the UML modeling tool. For the UML Loader to create a Value Domain, the model owner must first create a UML Class and stereotype it as either a <<CADSR Value

Domain>> or an <<enumeration>>. When a UML Class is stereotyped in this way, the SIW considers it as a Value Domain rather than an Object Class. The 'Name' of the stereotyped class will become the name of the Value Domain in caDSR. caDSR naming conventions specify that the representation type should be the last term in the Value Domain name, such as "State Code" where the representation term is "Code". The following six tagged values must be present in the <<enumeration>> or the <<CADSR Value Domain>> class in order to be recognized as a valid Value Domain.

- **CADSR_ValueDomainDefinition** - Used as the Value Domain Preferred Definition.
- **CADSR_ValueDomainDatatype** - Used as the underlying datatype for the value domain and must be one of the valid caDSR datatypes.
- **CADSR_ValueDomainType** - used as the underlying Value Domain type and must be one of 'E' (for Enumerated Value Domains) or 'N' (for Non Enumerated Value Domains)
- **CADSR_ConceptualDomainPublicID** - The combination of conceptual domain public id and version must point to an existing conceptual domain in the caDSR.
- **CADSR_ConceptualDomainVersion** - The Conceptual Domain tagged value for this class should be entered as either a whole number or a decimal e.g. 1 or 1.0 for "Version 1.0".

Value Domains may optionally contain the following annotation tagged values:

- ValueDomainConceptCode
- ValueDomainConceptDefinition
- ValueDomainConceptDefinitionSource
- ValueDomainConceptPreferredName
- ValueDomainQualifierConceptCode(n)
- ValueDomainQualifierConceptDefinition(n)
- ValueDomainQualifierConceptDefinitionSource(n)
- ValueDomainQualifierConceptPreferredName(n)

If these tagged values are present when editing the Value Domain Permissible Value list using the caDSR Curation Tool, the list of permissible values is constrained such that only children of the named NCIt concepts may be selected as Permissible values. Although the SIW does not allow modifying Value Domain tagged value, it can be used to review them (*Figure 7.32*). Create local Value Domains in a separate package from the regular domain objects.

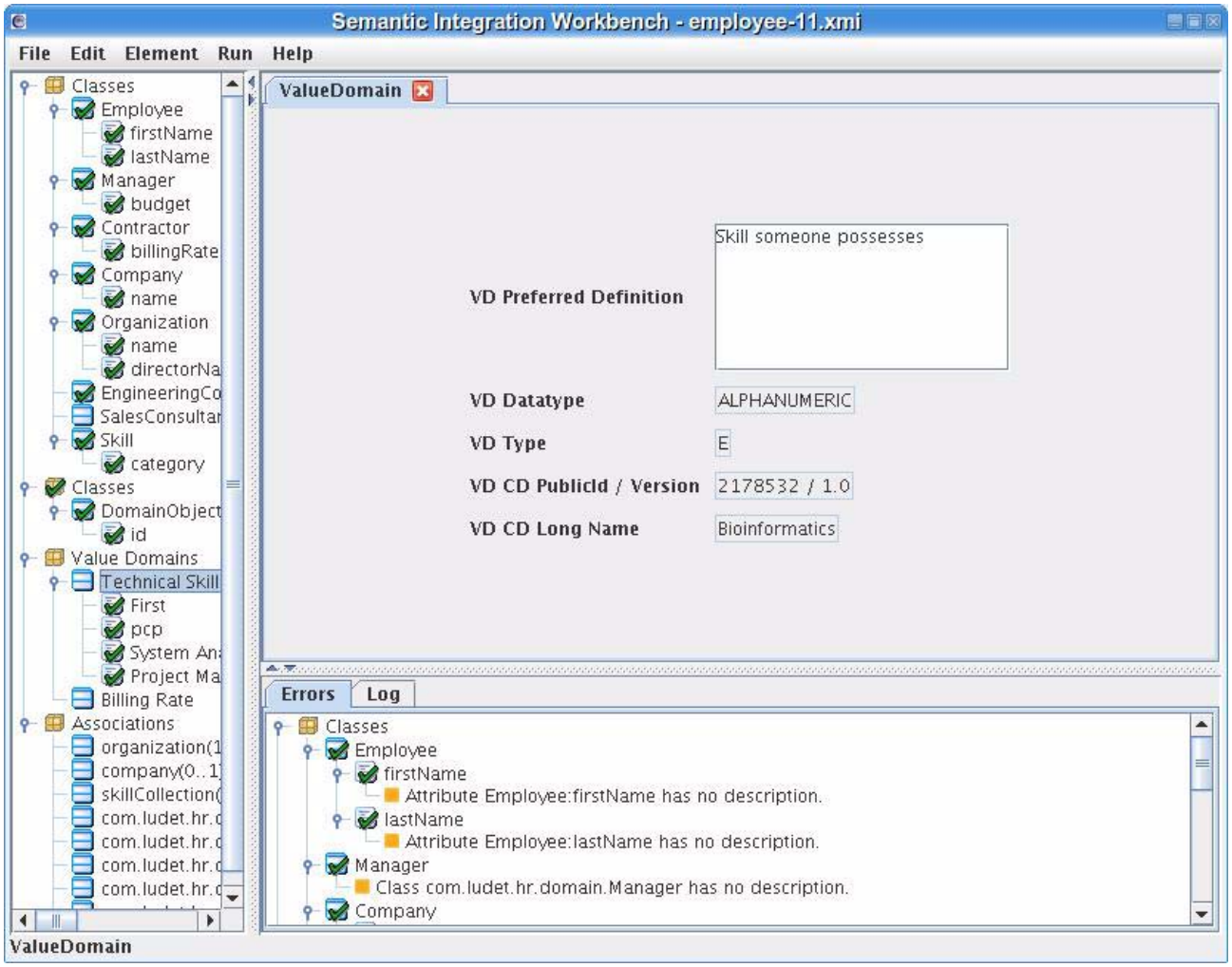


Figure 7.32 Reviewing Value Domain properties

Value Meanings

Each attribute within a <<CADSR Value Domain>> or <<enumeration>> class is considered a Value Meaning. Adding UML Attributes to a Value Domain class is the preferred way to create value meanings. Value Meanings must be annotated with EVS concepts. Annotation for Value Meanings is done the same way as it is for attributes with the exception that the tag names are different:

- ValueMeaningConceptCode
- ValueMeaningConceptDefinition
- ValueMeaningConceptDefinitionSource
- ValueMeaningConceptPreferredName
- ValueMeaningQualifierConceptCode(n)
- ValueMeaningQualifierConceptDefinition(n)
- ValueMeaningQualifierConceptDefinitionSource(n)
- ValueMeaningQualifierConceptPreferredName(n)

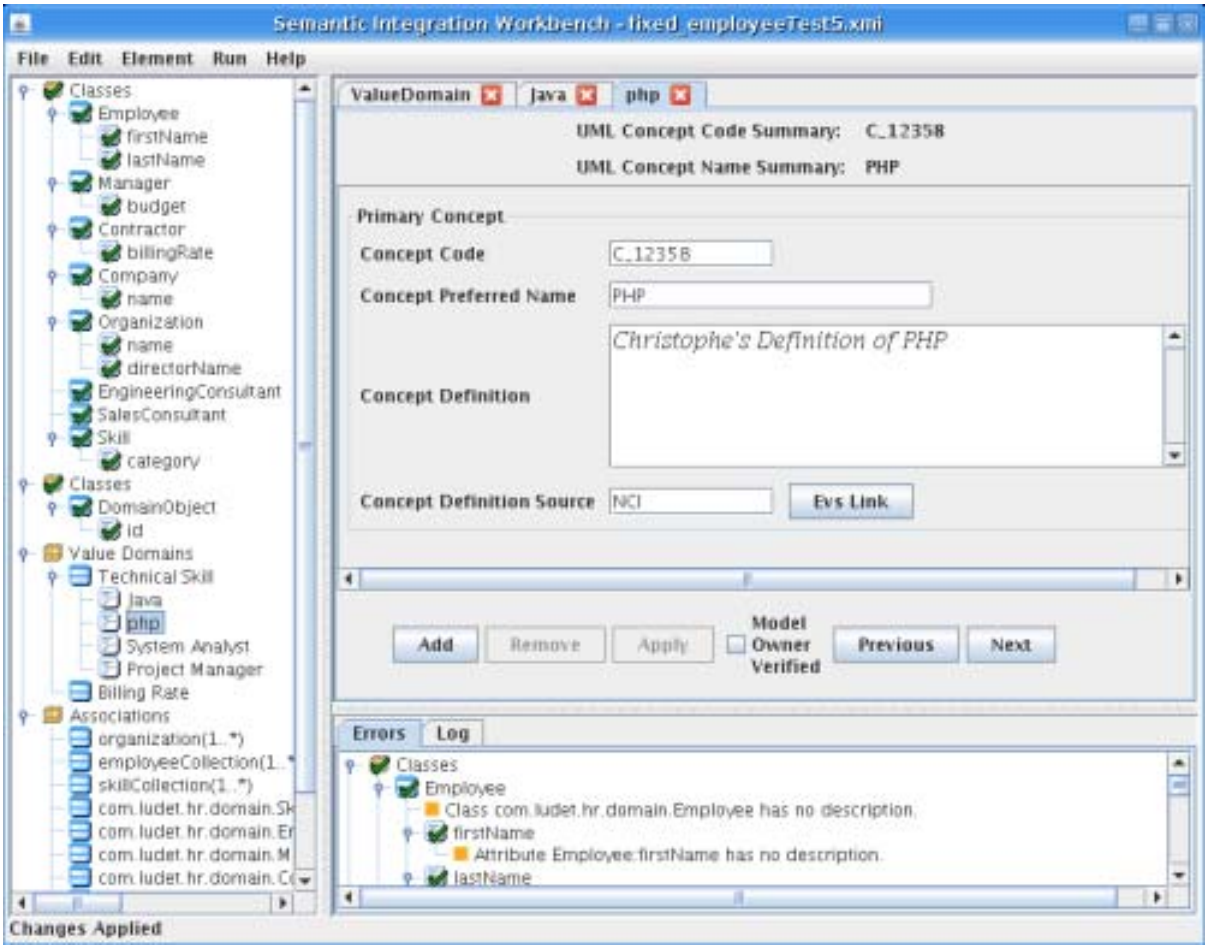


Figure 7.33 Value meaning annotations are similar to attribute annotations

Pointing a UML Attribute to a Value Domain

In order to indicate that a UML Attribute should use a value domain defined within the model, the model owner should add a tagged value of type 'CADSR Local Value Domain' to the attribute in EA. The value is the name given to the local value domain.

Example: The model owner creates a class with stereotype <<CADSR Value Domain>> or << enumeration>> with name 'My Value Domain'. For a UML attribute to use this value domain, the model owner adds a tagged value: 'CADSR Local Value Domain' / 'My Value Domain'.

Troubleshooting

Beginning to Use the SIW "Midstream"

If you have already begun using the caCORE SDK semantic integration workflow described in this guide to complete any of these steps prior to release 3.2, but you would like to use the SIW to complete the process, contact the NCICB application support group for assistance.

The Status Bar

At the bottom of the main viewer window, the Status Bar displays confirmation and informative messages. Use it to confirm that a file was saved, that changes were applied or need to be applied.

The Tabs

Several tabs can be opened at the same time. Use the mouse middle button (or equivalent) on an element to open its description in a new tab.

EVS Search Dialog

Within one session, the EVS Search dialog box remembers your previous search and column widths preferences. You can resize the table to your liking.

CHAPTER

8

REGISTERING METADATA

This chapter describes the process of registering and mapping metadata using the UML Loader.

Topics in this chapter include:

UML Loader

The UML Loader is a Java application that transforms UML object model class diagrams into caDSR metadata, creating or reusing existing caDSR administered components as needed. This process is also referred to as registering the UML model CDEs in caDSR. Specifically, UML classes, including inheritance and association links, and their attributes are transformed into caDSR metadata. UML class operations and other types of UML elements are not transformed. The UML Loader does not transform UML data models.

Once the UML object model is annotated with EVS concepts, as described in *Chapter 7*, and the EVS annotated version of the model has been approved, the model owner sends the XMI representation to the NCICB caDSR team for processing as described

in steps 1 and 2 on page 141. The EVS concept annotations form the basis for determining whether the caDSR component represented by the UML element is new, or exists in caDSR.

The resulting caDSR metadata is organized into Classification Scheme and Classification Scheme Items whose version corresponds to the version of the UML model. The details for the names of the classifications are specified by the model owner. Upon submission of a model for processing, the NCICB staff provides model owners with a template for specifying these and other required UML Loader run-time parameters.

Note: The UML Loader will not load XMI from UML model class diagrams that do not contain the mandatory EVS concept tags for all UML classes and attributes.

The mapping is summarized in Table *8.1*. Details describing the formation of each type of caDSR Administered Component, the methodology for comparing UML elements to existing caDSR components and creating Classification Schemes is found in other sections in this chapter.

caDSR Administered Component	***UML element***	***Description***
Concept Class	EVS Concept tagged values *Examples*: `ObjectClassConceptCode`, `ObjectClassConceptPreferredName`	See *Chapter 7* for a complete list of EVS concept tag names.
Object Class	UML Class	A caDSR Object Class is created for each unique UML Class.
Property	UML Attribute	A caDSR Property is created for each unique UML Attribute.
Data Element Concept (DEC)	UML Class:UML Attribute	A caDSR DEC is comprised of two key components, the Object Class and Property. Based on the Object Class and Property corresponding to the *UML Class* and *UML Attribute,* the UML Loader creates or reuses a DEC. One DEC is created for each combination of a *UML Class* and one of its *Attributes*. E.g. If *UML Class* Chromosome has 4 *Attributes*, there will be 4 DECs.

Table 8.1 caDSR Administered Component to UML Mapping

caDSR Administered Component	UML element	Description
Data Element	UML Class:UML Attribute:UML Attribute Datatype	Within a context, a caDSR Data Element is based on the unique combination of a DEC and a Value Domain (VD). To derive the Data Element, the UML Loader uses the DEC corresponding to a *UML Class* and one of its *Attributes*, and an existing VD corresponding to the *datatype* of the *UML Attribute* or to a VD defined in the Model file. Based on an evaluation of the uniqueness of these two components, the DEC and the VD, the UML Loader creates or reuses a Data Element. One Data Element is created for each DEC associated with the Model. E.g. If *Class Chromosome* has 4 *Attributes*, there will be 4 DECs and 4 Data Elements.
Classification Scheme	Project Full Name	Specified by the model owner in the UML Loader run-time parameters.
Classification Scheme Item	Package name or user defined.	UML Loader detects the package names in the XMI file and can use this to create a Classification Scheme Item, or the user can set a Classification Scheme Item default at run time for the UML Loader. The Classification Scheme/Classification Scheme Item is used to classify all the UML model-derived caDSR components.

Table 8.1 caDSR Administered Component to UML Mapping (Continued)

Submitting a UML Model to caDSR

Perform the following steps to submit the UML model for loading into the caDSR:

1. Submit XMI File created by Semantic Integration steps to NCICB for processing.
2. Send an email to the NCICB helpdesk at ncicb@pop.nci.nih.gov with the details of your request. You will be contacted and given instructions for providing user specified run-time parameters.

The process to access/verify caDSR Metadata UML Loader results using the CDE Browser are as follows.

Perform the following steps to use the Admin Tool to review UML Class transformations:

1. To find all transformed UML Classes, now represented as "Object Class" administered component in caDSR, sign on to the Admin tool using your caDSR User account and select **Object Class**, as in *Figure 8.1*.

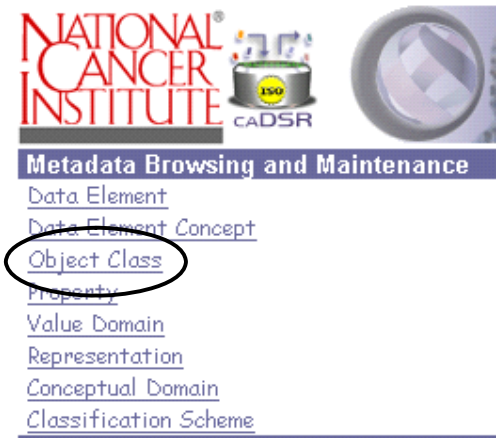


Figure 8.1 caDSR Metadata Browsing

   a. On the Search for Object Classes screen, select **List** and set **Context = caBIG** (all object classes are loaded into caBIG regardless of which Context the UML model itself belongs to). As illustrated in *Figure 8.2*, using the **Classification Scheme/Classification Scheme Item Filter,** set the search string to the name specified for each in the UML Loader run-time parameters provided to NCICB when the model was loaded. Click **Search** to launch the search.

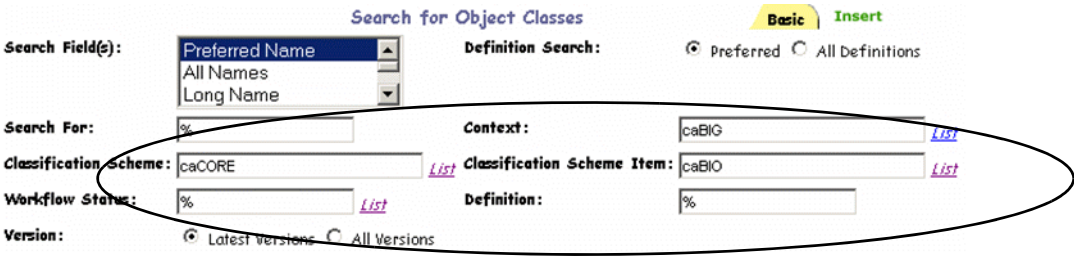
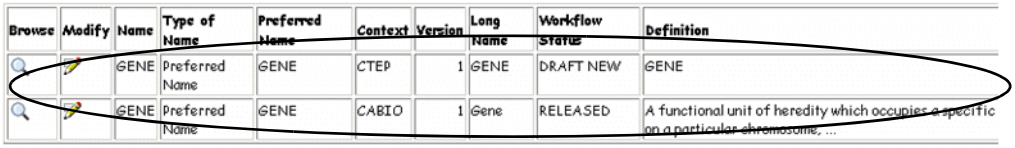


Figure 8.2 Search for Object Classes

   b. Review the list of Object Classes in the results list to be sure that all the UML classes from your object model were transformed into an equivalent Object Class as shown in *Figure 8.3*.



Browse	Modify	Name	Type of Name	Preferred Name	Context	Version	Long Name	Workflow Status	Definition
		GENE	Preferred Name	GENE	CTEP	1	GENE	DRAFT NEW	GENE
		GENE	Preferred Name	GENE	CABIO	1	Gene	RELEASED	A functional unit of heredity which occupies a specific on a particular chromosome, ...

Figure 8.3 Object Class search results

c. Click the **Browse** icon next to the class to view the object class details displayed in *Figure 8.4*. Here you can check for alternate names, alternate definitions and view EVS concept mapping.

Figure 8.4 Object Class Details

Perform the following steps to use the Admin Tool to review UML Class/attribute transformations:

1. Search for DECs associated with UML Class using the class name and a wildcard. For example, to find all the DECs associated with the class named “Gene” in the Admin Tool (*Figure 8.5*):
    a. Select **Long Name** as the Search Field(s).
    b. Enter ***gene%*** in **Search For**. (The % symbol is a wild card.)

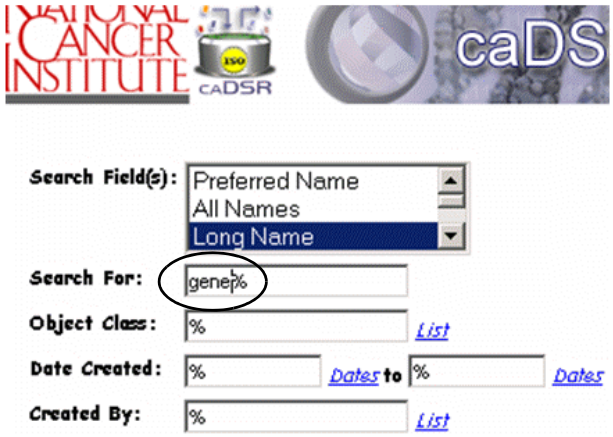


Figure 8.5 caDSR Admin Tool

Alternatively, perform the following steps to use the Curation Tool to review UML Class/attribute transformations:

1. Sign on to the Curation Tool and enter the following information as shown in *Figure 8.6*.
    a. Select **Data Element Concept** from **Search For**.

b. Select **Names & Definition** from **Search In**.

c. Enter the UML Class name and wildcard in **Enter Search Term.**

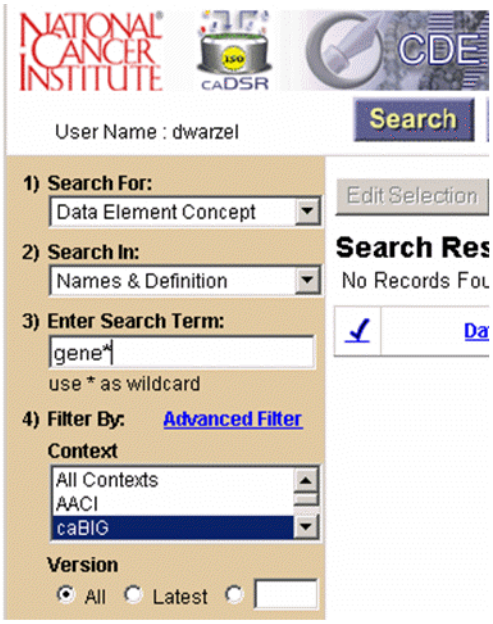


Figure 8.6 caDSR Curation Tool

Perform the following steps to use the CDE Browser:

1. In order to see a Classification caDSR Browser tree, the workflow status of the Classification must be set to **Released**. After doing so, using an Internet Explorer web browser, visit the caCORE CDE Browser site at http://ncicb.nci.nih.gov/CDEBrowser.
2. Using the various methods available in the browser search functions, search for or drill down to the **Classification Scheme/Classification Scheme** item for your model within your context. *Figure 8.7* illustrates opening the caBIG Context and the location of the **Classifications** leaf node.

Figure 8.7 CDE Browser, caBIG context

3. View the details of each CDE in the results panel by clicking on the CDE's name.
4. Download the data elements into Excel or XML by clicking on the appropriate action button (*Figure 8.8*). Contact your local IT staff if you have problems

downloading these files, as sometimes local firewalls prevent this feature from functioning properly.

[Download Data Elements to Excel] [Download Data Elements as XML]

Add to CDE Cart

1 - 40 of 820 Next 40

Long Name	Document Text	Owned By	Used By Context	Registration Status	Workflow Status	Public ID	Version
Acetylsalicylic Acid Use at Randomization Indicator	ASA Use at Randomization	caBIG			RELEASED	2185684	1.0
Address City Name	City	DCP	caBIG	Qualified	RELEASED	2179601	1.0
Address Country Name	Country	DCP	CCR,CIP,caBIG	Qualified	RELEASED	2179604	1.0
Address Line 1 Text	Address 1	DCP	caBIG	Qualified	RELEASED	2179598	1.0

Figure 8.8 Data Elements Download

Each of the components of the caCORE 3.2 Infrastructure has been transformed from their respective UML Object Models, EVS, caBIO, and caDSR. When available, the 3.2 version can be found in the caCORE Context under Classifications, Classification Scheme caCORE version 3.2.

Reviewing UML-Derived caDSR Metadata

The UML Model Browser is a web-based tool designed to let UML Model users/owners search for and view UML model components loaded in the caDSR database. It is an intuitive way to view UML classes, attributes, the association between classes and attributes, and their related ISO Components.

The following sections illustrate the data displayed by the Browser. For complete instructions on using the UML Model Browser, click on the "Help" icon from the Browser's main page.

Class Search

The UML Model Browser searches for model metadata based on either class or attribute properties. A class search for class names containing "ValueDomain" displays the following information in the result set (*Figure 8.9*).

Sort order : Class Name [Ascending]

1 - 5 of 5

Class Name	Project Name	Project Version	Project Workflow Stauts	Sub Project Name	Package Name	OC Public ID	OC Version
EnumeratedValueDomain	caCORE	3	RELEASED	Cancer Data Standards Repository (caDSR)	gov.nih.nci.cadsr.domain	2231687	1.0

Figure 8.9 UML Model Browser search for class or attribute properties

The "EnumeratedValueDomain" class was loaded in the caCORE project and belongs to the caDSR sub-project - the caDSR UML model is loaded to the caDSR just as other models are. This enabled the generation of the caDSR API via the caCORE code generation process.

Clicking on the Class Name link navigates to the list of attributes for that Class (*Figure 8.10*).

Attribute Name	Data Type	Definition	DE Name	DE Public ID	Project Name	Sub Project	Package Name
AdministeredComponent:beginDate	java.util.Date	The particular day, month and year this item became allowed. (ISO 11179)	Enumerated Value Domain Administered Item Begin Date java.util.Date	2232879	caCORE		gov.nih.nci.cadsr.domain

Figure 8.10 Navigating to attributes list for a class

Clicking on the "OC Public ID" opens the caDSR Object Class Browser to display the details for the Object Class with the Public ID listed for the Class.

Attribute Search

The alternative search option in the UML Model Browser is to search by attribute. A search for attribute names containing "beginDate" displays the following information for the result set (*Figure 8.11*).

1 - 21 of 21

Attribute Name	Data Type	Definition	DE Name	DE Public ID	Project Name	Sub Project	Package Name
AdministeredComponent:beginDate	java.util.Date	The particular day, month and year this item became allowed. (ISO 11179)	Administered Item Begin Date java.util.Date	2232297	caCORE		gov.nih.nci.cadsr.domain

Figure 8.11 UML Model Browser search for attributes

Clicking on the "DE Name" link opens the CDE Browser Data Element Details screen for that data element.

Accessing UML-Derived caDSR Metadata

The NCICB provides several methods for retrieving the UML object model-derived caDSR metadata for use in applications and deployment in data systems:

- caCORE caDSR Application Programming Interfaces (APIs), generated by following the methodologies and tools of the caCORE infrastructure
- caDSR curator tools provide a means to view and edit the UML derived Data Elements, Data Element Concepts and Value Domains associated with each. The CDEs derived from the model will not be visible using the CDE Browser Classification Scheme tree function until the Classification Scheme workflow status is set to **Released**, which can be changed when ready to make more widely accessible, using the Admin Tool.
- Once the Classification Scheme workflow status is set to **Released**, the CDE Browser may be used for public access to view, edit or download the UML Model CDEs into Excel or XML file format
- UML Model Query Service, available with caCORE 3.2, which greatly simplifies retrieval of the extensive set of metadata supporting the UML object model with little or no knowledge of the caDSR metadata structures or ISO/IEC 11179 (see *UML Domain Model Query Service* on page 147).

Note: The caCORE SDK process, in part, is applied to the caDSR resulting in the generation of caDSR APIs and transformation of the caDSR UML object model into caDSR meta-

data. The process began as described in the SDK by creating a caDSR UML domain object model in EA, performing semantic integration using the semantic connector tool, and transforming the EVS annotated XMI into caDSR metadata. The caDSR UML object model metadata is also available through each of these means.

For information about using the caCORE caDSR APIs, see the *caCORE 3.2 Technical Guide*. For links to the technical guide and access to the caDSR web based tools which provide access for viewing, updating and downloading caDSR metadata, visit the caDSR web site (http://ncicb.nci.nih.gov/NCICB/infrastructure/cacore_overview/cadsr). Though a UML object model can be viewed in any of the caDSR tools using the project's Classification Scheme/Classification Scheme Items specified when the model was loaded, the easiest and most accessible way is via the CDE Browser (http://ncicb.nci.nih.gov/CDEBrowser provides a direct link to the CDE Browser). The detailed views of CDEs in the browser provide access to all the underlying components of the CDE including concepts, Object Class, Property, Data Element Concept, Value Domain and classifications.

Note: The CDE Browser includes a Form Builder feature allowing you to organize CDEs into collections with properties analogous to paper forms, for sharing and communicating with end user communities. All caDSR published forms and templates from any Context can be centrally accessed, viewed, copied or downloaded from the caBIG context, Catalog of Published Forms Classification Scheme. caDSR tools are compatible with Internet Explorer. Use of other browser software may result in unexpected results

UML Domain Model Query Service

The UML Domain Model Query Service is a Java application for retrieving UML object model-derived caDSR metadata. The objective of the query service is to allow programmatic access to the transformed UML object model caDSR metadata with little or no knowledge about the caDSR database schema or the ISO/IEC 11179.

The 3.2 version of this API provides four basic methods to address some of the most likely types of queries. These methods are described in Table *8.2*.

Method Name	***Description***
findAllClasses `(java.lang.String projectName, float projectReleaseVersion)`	This method retrieves all classes in a particular UML domain model. The parameters required are the projectName and projectReleaseVersion.
findAssociatedClasses `(java.lang.String projectName, float projectReleaseVersion, java.lang.String packageName, java.lang.String className)`	This method retrieves all the associated domain object classes of a specific domain object class. The parameters are projectName, projectReleaseVersion, packageName and className. For example, the classes associated with "Gene" class.
findAttributeMetadata `(java.lang.String projectName, float projectReleaseVersion, java.lang.String packageName, java.lang.String className, java.lang.String attributeName)`	This method retrieves attribute level metadata. The parameters are projectName, projectReleaseVersion, packageName, className and attributeName. For example, the attribute "alignmentLength".

Table 8.2 UML Domain Query Service Method Summary

Method Name	Description
findAllAttributes (java.lang.String projectName, float projectReleaseVersion, java.lang.String packageName, java.lang.String className)	This method retrieves all the attributes of a specific domain object class. The parameters are projectName, projectReleaseVersion, packageName, and className. For example, all the attributes of the "Gene" class.
findClassMetadata (java.lang.String projectName, float projectReleaseVersion, java.lang.String packageName, java.lang.String className)	This method retrieves metadata for a UML class. The parameters are projectName, projectReleaseVersion, packageName and className. For example, all the metadata for "Gene" class.
findAssociationMetatada (java.lang.String projectName, float projectReleaseVersion, java.lang.String sourcePackageName, java.lang.String sourceClassName, java.lang.String targetPackageName, java.lang.String targetClassName)	This method retrieves metadata for an association between two UML classes. The parameters are projectName, projectReleaseVersion, sourcePackageName, sourceClassName, targetPackageName and targetClassName. For example, metadata for the association between "Gene" and "Protein" classes.

Table 8.2 UML Domain Query Service Method Summary (Continued)

The programmer invokes the API using parameter values from the UML elements of a specific UML object model. These parameters are described in Table *8.3*.

Parameter	Description	Example
projectName	The name of the project.	CaCORE
projectReleaseVersion	The Release version number of the project.	3.2
packageName	The name/alias of the package that the class belongs to.	CaBIO
className	The name of the UML class.	Gene
attributeName	The name of the UML attribute.	symbol

Table 8.3 UML Domain Model Query Service Parameter description

Creating a Concept for Object Class and Property

All UML Classes and their Attributes are mapped to one or more concepts in the NCI Thesaurus which is curated and served by Enterprise Vocabulary System (EVS). This mapping is captured in the UML domain model during the semantic integration steps described in *Chapter 7* by annotating classes and their attributes using tagged values (Name-Value pairs). The mapping process can be automated by the Semantic Connector utility which injects concept annotations into XMI by querying EVS using the caBIO EVS API. It can also be accomplished at design time by searching for concepts in EVS using its public web interface and then manually creating appropriate tagged values for various UML elements in the UML modeling tool.

Table *8.4* describes the class-level tagged values (using the example of Gene class) for primary concepts:

Tag Name	Description
ObjectClassConceptCode	The unique NCI Thesaurus concept code assigned to the primary concept associated with the UML Class. Example: C16612
ObjectClassConceptPreferredName	The NCI Thesaurus Preferred Name of the primary concept associated with the UML Class. Example: Gene
ObjectClassConceptDefinition	The NCI Thesaurus Definition of the primary concept associated with the UML Class.
ObjectClassConceptDefinitionSource	The source of the definition of the primary concept associated with the UML Class. Example: NCI-GLOSS

Table 8.4 Object Class concept UML Class-level tagged values

Table *8.5* describes the class-level tags for Qualifier concepts. The specific order of the Qualifier concepts conveys additional semantics. During semantic integration, an ordinal number, denoted by the “N” in the tag name, is assigned each Qualifier concept in relation to the primary concept. This number is used to create semantically meaningful names and definitions in caDSR. To determine the ordinal position, the primary concept is placed at the far right, each qualifier concept prepended, sequentially to form a human understandable name. The ordinal position is determined by the resulting string as follows: qualifierN qualifier2 qualifier1 primaryConcept.

Name	Description
ObjectClassQualifierConceptCodeN*	The unique NCI Thesaurus concept code assigned to the qualifier concept “N” associated with the UML Class.
ObjectClassQualifierConceptPreferred-NameN*	The NCI Thesaurus Preferred Name of the qualifier concept “N” associated with the UML Class.
ObjectClassQualifierConceptDefini-tionN*	The NCI Thesaurus Definition of the qualifier concept “N” associated with the UML Class.
ObjectClassQualifierConceptDefinition-SourceN*	The source of the definition being used for the Qualifier concept “N” associated with the UML Class.

Table 8.5 Object Class Qualifier concept UML Class-level tags

Table *8.6* describes the attribute-level tags for primary concepts:

Name	Description
PropertyConceptCode	The unique NCI Thesaurus concept code assigned to the primary concept associated with the UML Attribute.
PropertyConceptPreferredName	The NCI Thesaurus Preferred Name of the primary concept associated with the UML Attribute.
PropertyConceptDefinition	The NCI Thesaurus Definition of the primary concept associated with the UML Attribute.
PropertyConceptDefinitionSource	The source of the definition of the primary concept associated with the UML Attribute.

Table 8.6 Property concept UML Attribute-level tags

Table *8.7* describes the class-level tags for Qualifier concepts. To create names and definitions for the Property, the specific order of the Qualifier concepts is denoted by the "*n*" as described for Object Class Qualifiers above.

Tag Name	Description
PropertyQualifierConceptCodeN*	The unique NCI Thesaurus concept code assigned to the qualifier concept "N" associated with the UML Attribute.
PropertyQualifierConceptPreferredNameN*	The NCI Thesaurus Preferred Name of the qualifier concept "N" associated with the UML Attribute.
PropertyQualifierConceptDefinitionN*	The NCI Thesaurus Definition of the qualifier concept "N" associated with the UML Attribute.
PropertyQualifierConceptDefinitionSourceN*	The source of the definition being used for the Qualifier concept "N" associated with the UML Attribute.

Table 8.7 Property Qualifier concept UML Class-level tags

When the UML domain model is exported to XMI, it contains all concept tag annotations. UML Loader retrieves concept information by parsing them.

Creating New Concepts in caDSR

UML Loader first checks if a concept corresponding to the specified concept code already exists in caDSR. If it does not exist, then a new concept is created in caDSR. Data used for creating the new concept is shown in Table *8.8*, using the example of Gene UML class in caBIO 3.2.

Concept Attribute	Data	Example
Preferred Name	Derived from ConceptCode tagged value.	C16612

Table 8.8 Concept Attribute details

Concept Attribute	Data	Example
Long Name	Derived from ConceptPreferredName tagged value.	Gene
Preferred Definition	Derived from ConceptDefinition tagged value.	
Version	1.0 (Default)	1.0
Workflow Status	RELEASED (Default)	Released
Context	CaBIG (Default)	caBIG
Begin Date	Current Timestamp	01/23/2005

Table 8.8 Concept Attribute details (Continued)

Creating an Alternate Definition

Definitions from sources other than NCI are captured as alternate definitions for a Concept. Table *8.9* displays the details of the mapping.

Alternate Definition	Data
Definition	Derived from ConceptDefinition tagged value.
Context	Specified as a run-time parameter

Table 8.9 Alternate Definition details

Updating Existing Concepts in caDSR

If a concept corresponding to the specified concept code already exists in caDSR, UML Loader compares its existing definitions with the specified definitions and updates them if necessary.

Mapping a UML Class to an Object Class

Each class in the UML domain model is mapped to a caDSR Object Class. UML Loader resolves the semantic equivalence of two domain objects based on the NCI concepts to which they are mapped, as described in the Introduction. To map a UML class to a caDSR Object Class, the UML Loader retrieves the NCI concept codes of the UML class from the tagged values in the XMI and checks if a caDSR Object Class based on those values exists. If it exists, the domain object class is mapped to it; otherwise, a new corresponding object class is created. When an existing object class is reused, a new classification is assigned to it. Additional details for classifying existing object classes are discussed in sections *Mapping UML Model Metadata to Classification Scheme and Classification Scheme Items* on page 161 and *Assigning Classifications* on page 162.

To summarize, if two domain objects' UML classes are based on the same NCI concept(s), they are mapped to the same caDSR Object Class. Details of new object class creation are discussed in the following section.

Creating a New Object Class

Table *8.10* illustrates the details of a new object class that UML Loader creates.

Object Class Attribute	Description	Example
Preferred Name	Derived from the concept codes of the underlying concept(s). Usually the concept identifier(s).	C40992
Long Name	Derived from the long name of underlying concept(s).	Homologous Protein
Preferred Definition	Derived from the preferred definition of underlying concept(s).	A protein similar in structure and evolutionary origin to a protein in another species.
Version	1.0 (default)	1.0
Workflow Status	RELEASED (default)	Released
Context	caBIG (default)	caBIG
Begin Date	Current Timestamp	01/23/2005

Table 8.10 Object class attribute details

Creating an Alternate Name (Designation)

Two alternate names are created for each caDSR Object Class. Alternate names are based on the exact UML class name. The alternate name "Type" attribute of such alternate name is "UML Class". An alternate name based on the fully qualified UML class name (including the package name) is also created. The "Type" attribute for this alternate name is "UML Qualified Class". Appropriate classification is assigned to the alternate names. Details of assigning classifications are described in *Mapping UML Model Metadata to Classification Scheme and Classification Scheme Items* on page 161 and *Assigning Classifications* on page 162.

Creating an Alternate Definition

An alternate definition based on the UML class description of a domain object is created for the caDSR Object Class. The value for the description comes from the XMI documentation tag. The Alternate DefinitionType attribute of such alternate definition is “UML Class”. Appropriate classification is assigned to the alternate name. Details of assigning classifications are described in *Mapping UML Model Metadata to Classification Scheme and Classification Scheme Items* on page 161 and *Assigning Classifications* on page 162.

Using an Existing Object Class

An UML Class can be explicitly mapped to a caDSR Object Class by mapping any of the class's attributes. See *Using an Existing Property* on page 154. If an existing caDSR Object Class is re-used to map a UML class in a domain object, appropriate classification is assigned to it. An alternate name and an alternate definition are also

created for the existing caDSR Object Class based on the details specified *Creating an Alternate Name (Designation)* on page 152 and *Creating an Alternate Definition* on page 152.

Classifying an Object Class

UML-based caDSR Object Classes are classified using the principles described in *Mapping UML Model Metadata to Classification Scheme and Classification Scheme Items* on page 161 and *Assigning Classifications* on page 162.

Mapping a UML Attribute to a Property

Each attribute of a UML class is mapped to a caDSR Property. UML Loader resolves the semantic equivalence of two domain objects' properties based on the concepts to which they are mapped. To map a UML attribute to a caDSR Property, the UML Loader retrieves the concept codes of the UML attribute from the tagged values in the XMI and checks the existence of a caDSR Property based on those values. If it exists, the domain object's UML attribute is mapped to it; otherwise, a new caDSR Property is created. When an existing caDSR Property is re-used, a new classification is assigned to it.

To summarize, if two domain objects' UML class attributes are based on the same NCI concept (s), they are mapped to the same caDSR Property. Details of new Property attributes display in Table *8.11*.

Property Attribute	***Description***	***Example***
Preferred Name	Derived from the concept codes of the underlying concept(s). Usually the concept identifier(s).	C25552:C411167
Long Name	Derived from the long name of underlying concept(s).	Lead:OrganizationalUnit
Preferred Definition	Derived from the preferred definition of underlying concept(s).	Be in charge of.: Organizational unit like a laboratory, institute or consortium.
Version	1.0 (default)	1.0
Workflow Status	RELEASED (default)	Released
Context	caBIG (default)	caBIG
Begin Date	Current Timestamp	01/23/2005

Table 8.11 Property attribute details

Creating an Alternate Name (Designation)

An alternate name based on an exact UML attribute name is created for the caDSR Property. The Alternate Name Type of such alternate name is “UML Attribute”. Appropriate classification is assigned to the alternate name. Details of assigning classifications are described in *Mapping UML Model Metadata to Classification Scheme and Classification Scheme Items* on page 161 and *Assigning Classifications* on page 162.

Creating an Alternate Definition

An alternate definition based on a UML attribute description is created for the caDSR Property. The Alternate Definition Type of such alternate definition is "UML Attribute". Appropriate classification is assigned to the alternate name. Details of assigning classifications are described in *Mapping UML Model Metadata to Classification Scheme and Classification Scheme Items* on page 161 and *Assigning Classifications* on page 162.

Using an Existing Property

For each UML Attribute the UML Loader encounters, it checks for the following tagged values:

- CADSR_DE_ID
- CADSR_DE_VERSION

If both of these tagged values are present, no data element for this attribute is created. Instead, the CDE identified by the provided public ID and version is reused. The property, data element concept, value domain and object class for this CDE are also reused. If any concept annotation exists for this UML Attribute or its parent UML Class, the concept annotations are ignored.

Note: Mapping an attribute to an existing CDE automatically maps its parent UML Class to an existing Object Class.

If an existing caDSR Property is re-used to map a UML attribute, an appropriate classification is assigned to it. An alternate name and an alternate definition are also created for the existing caDSR Property based on the details specified in *Creating an Alternate Name (Designation)* on page 153 and *Creating an Alternate Definition* on page 154.

Classifying a Property

UML-based caDSR Properties are classified using the principles described in *Mapping UML Model Metadata to Classification Scheme and Classification Scheme Items* on page 161 and *Assigning Classifications* on page 162.

Note: For UML Model metadata to be consistently transformed and mapped across models, it is recommended that the UML Attribute names do not include the UML Class name. If for some reason they do, during the Semantic Integration steps, model owners should ensure that EVS concepts mapped to the Attribute in the Semantic Connector report represent only the Attribute portion of the Attribute name; concepts mapped to the Attribute's Class should not be repeated. For example, if you had not followed the naming conventions outlined for SDK, and have Class = 'Gene' and Attribute = 'geneSymbol' you would not use 'geneSymbol' for concept mapping. You would map the attribute to the concept 'symbol' and ignore the term 'gene'.

Creating Data Element Concepts

The relationship between a class and one of its attributes is represented by a caDSR Data Element Concept (DEC). A DEC is based on a caDSR Object Class that corresponds to the UML Class and a caDSR Property that corresponds to the UML class

attribute. UML Loader creates the DEC based on the details in Table *8.12* if it does not already exist. If it exists, it is re-used and an appropriate classification is assigned.

Data Element Concept Attribute	*Description*	*Example*
Preferred Name	Derived from the Object Class Public ID and version and Property Public ID and version. A colon is used as the separator character between these values.	1111111v1.0:2222222v1.0
Long Name	Derived from the Object Class Long Name and Property Long Name. A space is used as the separator character between these two values.	Homologous Protein Alignment Length
Preferred Definition	Object Class Preferred Definition and Property Preferred Definition. A colon is used as the separator character between these two values.	A protein similar in structure and evolutionary origin to a protein in another species: The linear extent in space from one end to the other. Often used synonymously with distance.
Version	Specified as a run-time parameter	3.0
Workflow Status	Specified as a run-time parameter	Draft New
Context	Specified as a run-time parameter	caCORE
Begin Date	Current Timestamp	01/23/2005
Object Class Long Name	Object Class corresponding to the UML Class	Homologous Protein
Property Long Name	Property corresponding to the UML Attribute	Alignment Length

Table 8.12 Data Element Concept details

Creating an Alternate Name (Designation)

An alternate name for the DEC based on the exact UML class name and attribute name is created for the caDSR DEC. The format used for the name is “UML class name:UML attribute name”. The Alternate Name Type of such alternate name is “UML Class:Attribute”. Appropriate classification is assigned to the alternate name. Details of assigning classifications are described in *Mapping UML Model Metadata to Classification Scheme and Classification Scheme Items* on page 161 and *Assigning Classifications* on page 162.

Creating an Alternate Definition

An alternate definition for the DEC based on the UML class description and attribute description is created for the DEC. The Alternate Definition Type of such alternate definition is “UML Class:Attribute”. Appropriate classification is assigned to the alternate definition. Details of assigning classifications are described in *Mapping UML Model*

Metadata to Classification Scheme and Classification Scheme Items on page 161 and *Assigning Classifications* on page 162.

Using an Existing Data Element Concept

If an existing DEC is re-used to represent the relationship of a UML class and one of its attributes, appropriate classification is assigned to it. An alternate name and an alternate definition are also created for the DEC based on the details specified in sections *Mapping UML Model Metadata to Classification Scheme and Classification Scheme Items* on page 161 and *Assigning Classifications* on page 162.

Classifying a Data Element Concept

UML-based DECs are classified using the principles described in *Mapping UML Model Metadata to Classification Scheme and Classification Scheme Items* on page 161 and *Assigning Classifications* on page 162.

Mapping a UML Class to a Value Domain

Each class stereotyped as "CADSR Value Domain" in the UML domain model is mapped to a caDSR Value Domain. The UML Loader resolves the semantic equivalence of two domain objects based on its long name and context.

For each Class with Stereotype <<CADSR Value Domain>>, the UML Loader looks for the following tagged Values:

- CADSR_ValueDomainDefinition
- CADSR_ValueDomainDatatype
- CADSR_ValueDomainType (E or N)
- CADSR_ConceptualDomainPublicID
- CADSR_ConceptualDomainVersion
- CADSR_RepresentationPublicID
- CADSR_RepresentationVersion

and optionally:

- ValueDomainConceptCode, ValueDomainQualifier
- ValueDomainConceptPreferedName, ValueDomainQualifier
- ValueDomainConcept Definition, ValueDomainQualifier
- ValueDomainConceptDefinitionSource, ValueDomainQualifier

During the load process, the UML Loader starts by creating a Value Domain for each class with Stereotype <<CADSR Value Domain>>. The Value Domain is created according to the following table:

caDSR Field	*Value Used*
Long Name	UML Class Name
Preferred Name	Similar to the generated public ID
Preferred Definition	From 'ValueDomainDefinition' tagged value

Table 8.13 caDSR fields for creating a Value Domain

caDSR Field	Value Used
Value Domain Type	From 'ValueDomainType' tagged value
Context	From run time default values provided by model owner
Version	1.0
Workflow Status	Specified as Default (e.g DRAFT NEW)
Conceptual Domain	From 'ConceptualDomainPublicID' and 'ConceptualDomainVersion' tagged values
Datatype	From 'ValueDomainDatatype'
Begin Date	The date of the load
Concept Derivation Rule	Optionally created from the list of Concepts in tagged values. Similar to Object Class Concept Derivation Rules

Table 8.13 caDSR fields for creating a Value Domain (Continued)

Value Meanings

For each attribute under each Class with Stereotype <<CADSR Value Domain>>, the UML Loader looks for the following tagged values:

- ValueMeaningConceptCode
- ValueMeaningQualifierConceptCode(n)
- ValueMeaningConceptPreferredName
- ValueMeaningQualifierConceptPreferredName(n)
- ValueMeaningConceptPreferredDefinition
- ValueMeaningQualifierConceptPreferredDefinition(n)
- ValueMeaningConceptDefinitionSource
- ValueMeaningQualifierConceptDefinitionSource(n)

During the load process and after each Value Domain has been loaded, the UML Loader loads Value Meanings for each attribute belonging to a Value Domain Class The Value Meaning is created according to the following table:

caDSR Field	Value Used
Value Meaning	Created from the concatenation of 'ValueMeaningConceptPreferredName' and qualifiers if present.
Value Meaning Description	Concatenation of 'ValueMeaningConceptPreferredDefinition(n)' tagged values and qualifiers if present.
Begin Date	The date of the load
Concept Derivation Rule	Created from the list of Concept in tagged values. Similar to Object Class and Properties Concept Derivation Rules

Table 8.14 caDSR fields used for Value Meaning

Permissible Values

For each Value Meaning, UML Loader creates a permissible value according to the following table:

caDSR Field	*Value Used*
Value	From the underlying attribute's name
Value Meaning	Similar to the Value Meaning
Meaning Description	
Begin Date	The date of the load

Table 8.15 caDSR fields used for Permissible Values

Using a Value Domain Defined within the Model

In order to indicate that a UML Attribute should use a value domain defined within the model, the model owner adds a tagged value in EA of type 'CADSR Local Value Domain' to the attribute. The value is the name given to the local value domain. Example: The model owner creates a class with stereotype 'CADSR Value Domain' with name 'My Value Domain'. For a UML attribute to use this value domain, the model owner will add a tagged value : 'CADSR Local Value Domain' / 'My Value Domain'

Creating Data Elements

A UML attribute and its datatype are represented by a Data Element. In caDSR, a Data Element is based on a Data Element Concept and a Value Domain. UML derived data element created similarly based on the DEC derived from the combination of a UML Class and one of its attributes, and a generic existing caDSR Value Domain that corresponds to the datatype of the attribute or a Value Domain defined in the model. UML Loader creates the data elements based on the details displayed in Table *8.16* if they do not already exist.

Data Element Attribute	*Description*	*Example*
Preferred Name	Derived from the DEC Public ID and version and the Value Domain Public ID and version. A colon is the separator character.	3333333v1.0v:4444444v1.0
Long Name	Derived from the DEC Long Name and VD Long Name. A space is the separator character.	Homologous Protein Alignment Length java.lang.Long

Table 8.16 Data Element details

Data Element Attribute	Description	Example
Preferred Definition	Derived from the DEC Preferred Definition and Value Domain Preferred Definition. The separator character is a colon.	A protein similar in structure and evolutionary origin to a protein in another species: The linear extent in space from one end to the other. Often used synonymously with distance. Value Domain for java language 'java.lang.Long' datatype.
Version	Specified as a run-time parameter	3.2
Workflow Status	Specified as a run-time parameter	Draft New
Context	Specified as a run-time parameter	caCORE
Begin Date	Current Timestamp	01/24/2005
Data Element Concept Long Name	The Data Element Concept corresponding to the UML Class and one of its attributes.	Homologous Protein Alignment Length
Value Domain	Corresponds to the Datatype of the Attribute. Supported datatypes include java classes and java primitives. For the following java classes, the default value is 'null' java.lang.String java.lang.Double java.lang.Boolean java.lang.Short java.lang.Long java.lang.Integer java.lang.Float java.util.Date For the following java primitives, the default is listed after the class: int 0 short 0 long 0 float 0 double 0 boolean FALSE char \u0000 byte 0	java.lang.Long

Table 8.16 Data Element details (Continued)

Creating an Alternate Name

Two alternate names are created for the Data Element derived in this manner:

1. The UML attribute's class name and the attribute name. The Alternate Name Type of such alternate name is "UML Class:Attribute";
2. The fully qualified attribute name. The Alternate Name Type of such alternate name is "UML Qualified Attr".

Appropriate classification is assigned to the alternate names. Details of assigning classifications are described in *Mapping UML Model Metadata to Classification Scheme and Classification Scheme Items* on page 161 and *Assigning Classifications* on page 162.

Creating an Alternate Definition

An alternate definition based on a UML attribute description is created when a new caDSR Data Element is created for a UML Class, Attribute and datatype. The Alternate Definition Type of such alternate definition is "UML Attribute". Appropriate classification is assigned to the alternate name. Details of assigning classifications are described in *Mapping UML Model Metadata to Classification Scheme and Classification Scheme Items* on page 161 and *Assigning Classifications* on page 162.

Note: The UML attribute description was chosen as opposed to the combination of the description of the class and attribute because, in practice, when defining an attribute for a class, the notion of the class is generally incorporated in the attributes description. E.g. Class = 'Protein Homolog' Attribute = 'alignmentLength' attribute definition = "The alignment length of the protein."

Using an Existing Data Element

There are two scenarios in which an existing data element is tied to a new UML Model by the UML Loader rather than creating a new data element:

1. When the Context into which the UML model is being loaded is different from the Context that owns the existing data element. In this case, a "USED_BY" designation is added for the existing data element to capture the use by another Context
2. When both Contexts are same but the UML Models are different.

In both cases, an appropriate classification is assigned to the existing data element and the value for the UML Attribute "Name" and the fully qualified name are added as alternate names for the existing Data Element as described above in Creating Alternate Names on this page.

Classifying a Data Element

UML based data elements are classified using the process described in sections *Mapping UML Model Metadata to Classification Scheme and Classification Scheme Items* on page 161 and *Assigning Classifications* on page 162.

Mapping UML Model Metadata to Classification Scheme and Classification Scheme Items

The UML Loader uses one Classification Scheme (CS) and at least one Classification Scheme Item (CSI) for each UML domain model. A UML class model very commonly is organized into different packages. A CS is created based on the project name and release information that are entered as parameters for the UML Loader.

There are two options for specifying a caDSR classification for UML Models:

1. The UML Loader can be configured to create a CSI corresponding to each package in the UML class model. With this option, the UML Loader associates the CSI with the CS. The UML Loader uses the package names from the XMI.
2. The UML Loader can be configured to ignore the packages in the UML class model, and the user can specify one CSI for the entire UML class model. UML Loader then creates only one CSI associated with the CS. This option should be used for loading UML class models that do not contain any packages.

Note: The Classification Scheme will not show up in the CDE Browser, which requires no caDSR user account to view, until the workflow status is set to “Released”. This provides model owners an opportunity to use the caDSR curator tools to review and edit the content until they are ready for more general access.

Table *8.17* displays Classification Scheme details. Table *8.18* displays Classification Scheme Item details.

CS Attribute Name	***Description***	***Example***
Preferred Name	Project abbreviated name - Specified as a run-time parameter	caCORE
Long Name	Project full name - Specified as a run-time parameter	Cancer Common Ontologic Representation Environment
Preferred Definition	Project description - Specified as a run-time parameter	This is the classification scheme for the caCORE Java Packages that have been transformed from UML into caDSR Metadata.
Version	Project Release version specified as a run-time parameter	3.2
Workflow status	Draft New (default)	Draft New
Context	Specified as a run-time parameter	caCORE
Begin Date	Current Timestamp	01/25/2005
Type	Project (Default)	Project

Table 8.17 Classification Scheme details

CSI Attribute Name	Description	Example
Name	Package name/alias from the XMI or a single CSI is specified as a run-time parameter.	caBIO, caArray
Type	UML Package(Default)	UML Package

Table 8.18 Classification Scheme Item details

Assigning Classifications

UML Loader assigns classifications using the appropriate CS and CSI which are created based on the details described in the preceding section.

Mapping UML Associations to Object Class Relationships

Each Association in the UML domain model is mapped to an Object Class Relationship in caDSR.

Creating a New Object Class Relationship

Table *8.19* illustrates the details of the new Object Class Relationship created by the UML Loader.

Object Class Relationship Attribute	Data
Preferred Name	Generated, equals source class corresponding Object Class public ID + version; target class corresponding Object Class and version.
Long Name	Derived from the role name of underlying association.
Preferred Definition	Derived from the type of association. Example of the derived value for preferred definition: Zero-to-Many Zero-to-One Many-to-One One-to-Many Many-to-Many Generalizes
Version	1.0
Workflow Status	Draft New – Specified as a parameter
Context	Specified as a parameter
Begin Date	Current Timestamp
Type	HAS_A
Source Low Cardinality	Derived from UML Association. The Source object is the class from which the link is drawn.

Table 8.19 New Object Class Relationship details

Object Class Relationship Attribute	Data
Source High Cardinality	Derived from UML Association. The Source object is the class from which the link is drawn.
Target Low Cardinality	Derived from UML Association. The Target object is the class to which the link is drawn.
Target High Cardinality	Derived from UML Association. The Target object is the class to which the link is drawn.
Direction	Navigability. Source-to-Target Target-to-Source Bidirectional

Table 8.19 New Object Class Relationship details

Classifying an Object Class Relationship

UML based object class relationships are classified using the process described in sections *Mapping UML Model Metadata to Classification Scheme and Classification Scheme Items* on page 161 and *Assigning Classifications* on page 162.

Mapping UML Inheritance

Each Inheritance type association in the UML model is mapped to an Object Class Relationship in caDSR with the same attributes described for Associations, except for Object Class Relationship Type, which in this case is “IS_A”.

Additionally, the child class inherits all attributes of the parent class. Data Element Concepts based on the child class and each of its parent’s attributes are derived according to the mapping rules outlined in *Classifying a Data Element Concept* on page 156. Data Elements are created corresponding to each Data Element Concept plus an existing caDSR Value Domain as described in *Creating Data Elements* on page 158; the parent attribute’s datatype is used to map the Value Domain. See Table *8.20* and Table *8.21*.

Data Element Concept Attribute	Data
Preferred Name	Child Object Class Public ID + Object Class Version: Parent Property Public ID + Property Version
Long Name	Child Object Class Long Name + Parent Property Long Name
Preferred Definition	Child Object Class Preferred Definition + Parent Property Preferred Definition
Version	1.0 (Specified as a parameter)
Workflow Status	Draft New (Specified as a parameter)
Context	Specified as a parameter
Begin Date	Current Timestamp

Table 8.20 Inheritance Data Element Concept mapping

Data Element Concept Attribute	Data
Object Class	Object Class corresponding to the Child UML Class
Property	Property corresponding to the Parent UML Attribute

Table 8.20 Inheritance Data Element Concept mapping (Continued)

Data Element Attribute	Data
Preferred Name	DEC Public ID + DEC Version: Value Domain Public ID + Value Domain Version
Long Name	DEC Long Name + VD Long Name
Preferred Definition	Derived from the underlying attribute description in the UML class diagram.
Version	1.0 – Specified as a parameter
Workflow Status	Draft New – Specified as a parameter
Context	Specified as a parameter
Begin Date	Current Timestamp
Data Element Concept	The Data Element Concept corresponding to the Child UML Class and the Parent Attribute.
Value Domain	Corresponds to the Datatype of the Parent Attribute: java.lang.String = Value Domain(VD) Name “java.lang.String” java.lang.Boolean = VD “java.lang.Boolean” java.lang.Long = VD “java.lang.Boolean” java.lang.Integer = VD “java.lang.Boolean” java.lang.Float = VD “java.lang.Float” java.util.Date = VD “java.lang.Date” int = VD “int” long = VD “long” boolean = VD “boolean” char = VD “char” double = VD “double” float = VD “float” byte = VD “byte” short = VD “short”

Table 8.21 Inheritance Data Element mapping

CHAPTER

9

GENERATING A CACORE-LIKE SYSTEM

This chapter describes the process for generating the code that produces a caCORE-like system, executing tests on the system, and creating manual ORMs.

Topics in this chapter include:

Generating Code

At this point, you have created an object model and a data model, exported those models to XMI, and generated a DDL script from the data model. You have also annotated your model with immutable concept codes from EVS and registered your metadata in caDSR, thereby enabling semantic interoperability. This section describes how to generate the Java source code for a data access API using the XMI file you generated.

Updating the Property File

Because you went through the test procedures described in the *caCORE SDK 3.2 Installation and Basic Test Guide (*http://ncicb.nci.nih.gov/NCICB/infrastructure/cacoresdk#Documentation), you have confirmed that you have a fully-functioning API, ORM, and database for the example model. Before you generate any code, you must update the `deploy.properties` file so that 1) you do not reinstall software that you previously installed and 2) you are using the correct name for all of the filenames and directories.

Open the property file `{home_directory}/conf/deploy.properties` and modify the values to conform with the file displayed in *Figure 9.1* and described in Table *9.1*. These user-defined values are used during the build step that follows.

Note: The property file `deploy.properties` as shown in *Figure 9.1* is a Windows specific properties file. See the *caCORE 3.2 SDK Installation and Basic Test Guide* describing modifications that must be made for UNIX/Linux systems.

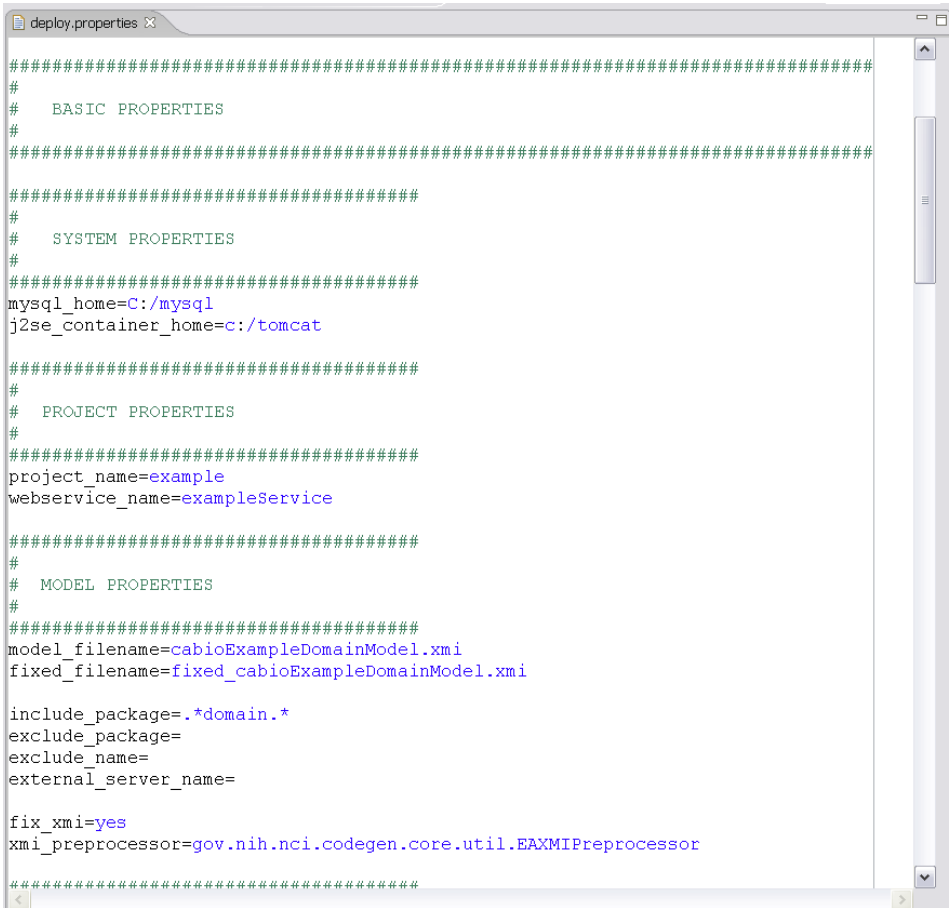


Figure 9.1 Example deploy.properties file

Property Name	Description
`project_name`	Provide a descriptive name for your project. This name must be one word and contain no spaces.
`install_tomcat`	Specify **yes** to install Tomcat (http://jakarta.apache.org/tomcat/) or specify **no** if Tomcat or another web container is already installed. CAUTION: If you specify **yes**, any previous versions of Tomcat may be overwritten which could adversely affect programs running on your computer.
`j2se_container_home`	Provide the root directory for your J2SE container (Tomcat or JBoss).
`create_schema`	Specify **yes** to create a schema for your database. For example, the cabioexampleschema.SQL file contains the Data Definition Language (DDL) scripts that will be used to create the schema for the provided example model. Specify **no** if your database schema has already been created or you are using another database.

Table 9.1 deploy.properties descriptions

Property Name	Description
import_data	Specify **yes** to import data to your database. For example, the cabioexampledata.SQL file contains the data for the provided sample model. Specify **no** if your database is already populated with data or you are using another database.
ddl_filename	Provide the name of your database DDL script (for example, cabioexampleschema.SQL)
datadump_name	Provide the name of your data file (for example, cabioexample-data.SQL).
db_server_name	Provide the name of your database host. This can be localhost, a fully-qualified DNS name or an IP address.
db_user	Provide the user name for database authentication.
db_password	Provide the password for the database authentication.
schema_name	Provide a name for your database schema. If the schema does not exist and you have set create_schema to yes (see above), a new schema will be created.
install_mysql	Specify **yes** to install MySQL or specify **no** if MySQL or another database is already installed. CAUTION: If you specify yes and the mysql_home you enter (see below) contains an existing MySQL installation, that version and any associated data will be deleted.
create_mysql_user	Specify **yes** to create a MySQL user account. If this is set to **yes**, a new user account will be created based on the values specified in “db_user” and db_password” (see above).
mysql_home	Provide the home directory for MySQL.
database_type	Enter the database platform that you are using. The following values are currently supported: **mysql**, **oracle**, **db2**
create_cache	Set this value to **yes** to enable second-level caching. Specify **no** if you do not want second-level caching.
cachepath	Specify the path to the directory where you want your cache files to be saved. This value is ignored if create_cache is set to **false.**
logical_model	Provide the name of the XMI file created from your object and data model. For example, the sample provided is cabioExam-pleDomainModel.xmi. This file must reside in {home_directory}/models/xmi.
fix_xmi	Specify **yes** to run the XMI pre-processor to convert an XMI file to a valid NetBeans MDR file.
xmi_preprocessor	Provide the fully-qualified Java class name of the XMI pre-pro-cessor to use. By default, the SDK provides a pre-processor for EA files, gov.nih.nci.codegen.core.util.EAXMIPreprocessor.

Table 9.1 deploy.properties descriptions (Continued)

Property Name	Description
include_package	Provide a list of packages to include in code generation separated by the pipe '|' symbol(s).[a] For example: include_package=.*cabio.domain.*|.*camod.domain.*.
exclude_package	Provide a list of packages to exclude from code generation separated by the pipe '|' symbol(s). [b]For example: exclude_package=.cabio.domain.*|.*camod.domain.*
default_security_level	Determines if the security provided through CSM is enabled or disabled. 1 indicates security on by default, 0 indicates security off by default
application_name	Application Context Name used for CSM security. This should be the same name used in the UPT.
default_session_timeout	Determines the default timeout for client sessions when security is enabled. It is set in milliseconds.
disable_writable_api_generation	Determines whether writable APIs should be disabled or enabled for this application. If this value is set to "yes", writable APIs are disabled. To turn write functionality on, set this property value to "no".

Table 9.1 deploy.properties descriptions (Continued)

a.These properties use regular expressions to determine elements within the model's package tree structure to include or exclude for code generation. Additional information about forming more complex regular expressions can be found on the Internet.

b.These properties use regular expressions to determine elements within the model's package tree structure to include or exclude for code generation. Additional information about forming more complex regular expressions can be found on the Internet.

Building the System

Perform the following steps to build your system.

1. In a Command Prompt window, enter ***cd {home_directory}*** to go to your home directory (for example, in Windows `c:\cacoretoolkit`).
2. Enter ***ant build-system***.

   Ant messages display as each task is processing. The `build-system` task builds the entire system and deploys the software to the webapp directory of the web application server installation specified in the `deploy.properties` file.
3. After your web application server has completely finished starting, run the following command to deploy the system web services: ***ant deployWS***.

Selectively Generating Artifacts

Running `ant build-system` generates all of the code and artifacts that are required to build a caCORE-like system. Sometimes, however, it is desirable to only generate certain portions of the system (particularly when validating the model to see whether code generation will be successful).

The following targets can be run from the command prompt by entering `ant <target>`.

Ant Target	*Description*
`build-beans`	Generates Java beans representing domain objects, copies custom beans (see below) and compiles all beans
`build-orm`	Generates ORM configuration files from model, using custom ORM files when specified
`build-schema`	Creates database user and loads schema and data (when specified in deploy.properties)
`build-artifacts`	Create configuration and other required system files from model
`generate-beans`	Generate beans based on model specified in deploy.properties (does not compile them)
`generate-cache`	Generate cache configuration files (used by ehCache)
`generate-dao-conf`	Generate data access object (DAO) configuration files from model
`generate-junit`	Generate JUnit test classes from model
`generate-orm-conf`	Generate ORM configuration files from model (used by Hibernate)
`generate-schemas`	Generate XML schemas from model (used by XML conversion utility)
`generate-wsdd`	Generate Web service deployment descriptor (WSDD) from model (used to deploy Web services)
`generate-xml-mapping`	Generate XML mapping files (used by XML conversion utility)

Table 9.2 Ant targets and descriptions

Documentation and Style Tools

This section contains tools that are part of the SDK framework and are useful for documentation and styling.

- **Javadoc** – Execute the Ant task `doc` to generate Javadocs for your beans. Your javadocs will be generated to the `{home_directory}/output/{project_name}/doc` directory. For more information on Javadoc see http://java.sun.com/j2se/javadoc/.
- **Jalopy** – Execute the Ant task `format` to make your code well formatted. The default indentation format is used in the SDK. This task is configurable to enforce coding standards that you wish to adhere to for your project. See http://jalopy.sourceforge.net/manual.html for information on how to customize this task.

Executing JUnit Tests

Note to Windows Users: Command screens that pop up during the build indicate that MySQL and Tomcat are running. You must leave those windows open as you execute the SDK tests. Closing them kills the associated applications.

JUnit test cases can be automatically generated and run by using the Ant `runtest` task located in the `{home_directory}\output\{project_name}\package\thick-client\build.xml` file. This task generates one test case for each domain object which exercises all methods contained within the domain objects. The test cases are generated to the `{home_directory}\output\{project_name}\{package_structure}\test` directory, and the results of running the JUnit tests are output to `{home_directory}\output\{project_name}\junit` directory. *Figure 9.2* shows where these files are located for the example model.

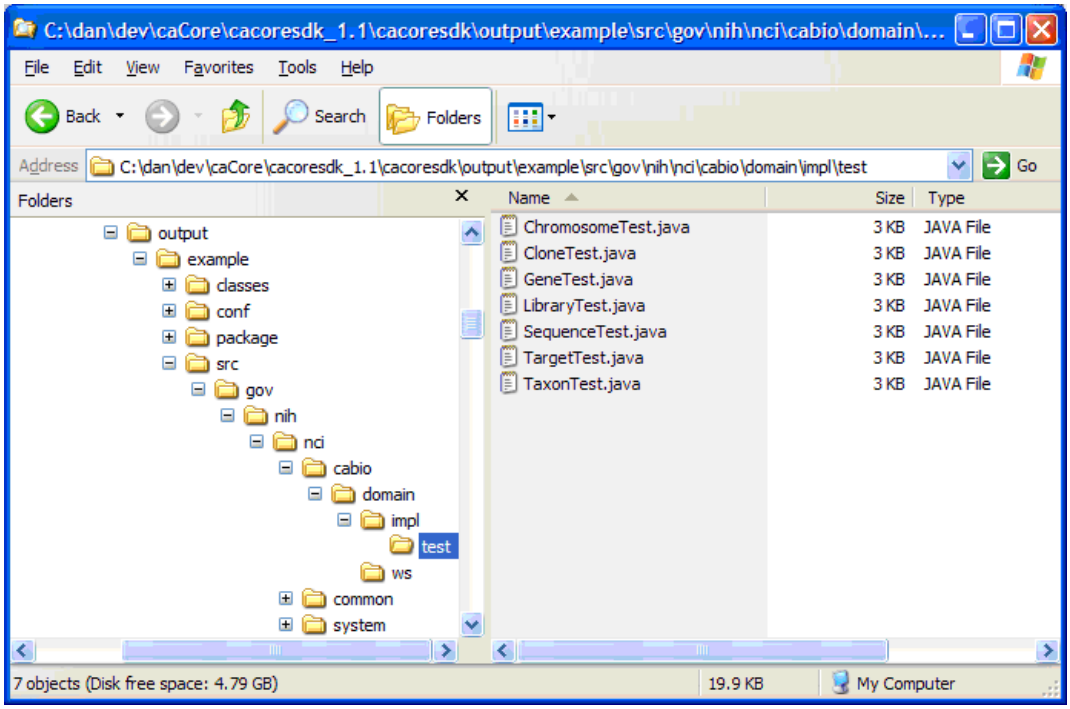


Figure 9.2 JUnit test files

Using a Generated System

Configuring Java Clients

During the process of generating a caCORE-like system, the Code Generator also produces and packages the components necessary to access the generated server from a Java-based client. The `{home_directory}/output/{project_name}/package/client` directory contains these files. Most notably, a JAR file called `{project_name}_client.jar` is created in the `lib` subdirectory that contains the domain objects and framework classes required to access the running system. This file, along with the rest of the contents of the `lib` and `conf` directories, must be on the classpath of clients that access the generated system. Several example classes are included in the `…/client` directory that demonstrate use of these files.

To access a caCORE-like system from your client application:

1. Copy the contents of the `lib` and `conf` subdirectories of `{home_directory}/output/{project_name}/package/client` to your classpath.

2. Modify the file called `remoteService.xml` (located in the `conf` subdirectory) to refer to the instance of the caCORE-like server to which you wish to connect. The service URL is of the form:

   `http://{web_server_name}:{web_server_port}/{project_name}/http/remoteService`

   Note that the port number is not necessary if the Web server is running on port 80.

3. In your client code, instantiate an object that implements the `ApplicationService` interface (typically using `ApplicationServiceProvider.getRemoteInstance()`) and use this object to query the caCORE-like system.

Details about the service interfaces and examples of their use can be found in Chapter 3 of the caCORE 3.2 Technical Guide.

Configuring Non-Java Clients

In addition to the Java interface, caCORE-like systems can also be accessed using the Web services and XML-HTTP interfaces. For more information on how to configure and use these interfaces, refer to Chapter 3 of the caCORE 3.2 Technical Guide.

Variations to Generating a caCORE-like System

This section contains variations to the normal process described in previous sections of this chapter.

Using Second-Level Caching

Hibernate has multiple levels of built in caching mechanisms. The first and second-level caches resolve circular/shared references and repeated requests for the same instance in a particular session. By default, the first level cache is turned on in caCORE-like systems, but due to the stateless nature of the generated API, when sessions are returned to the factories at the end of each request the first-level cache is cleared and does not provide any performance enhancement.

Hibernate features an extremely granular (class or collection role) second-level cache and offers various pluggable implementations for it. The SDK is set up to use the Ehcache (http://ehcache.sourceforge.net./) implementation (Hibernate–default). A second-level caching strategy improves performance for frequently run queries. Ehcache also provides a memory to disk persistence caching strategy, which is highly scalable.

The SDK code generator can enable second-level caching by setting the **create-cache** parameter to **yes** in the `deploy.properties` file and specifying where the cache files should be written to in your system in the **cachepath** property setting.

If activated, the default-generated caching configuration is set to **read-only**. This caching strategy is generally appropriate for systems using databases that are not subject to frequent updates. The time-to-live setting for the cache is set to 100000 seconds or a little over 27 hours. After this point, the cache will be flushed. The output of the `generate-cache` Ant task is the `ehcache.xml` file which contains system cache settings. To customize this file, you need to modify the `UML13EHCacheTransformer.java` file

located in the `gov.nih.nci.codegen.core.transformer` package of the SDK src directory.

To properly understand caching strategies and what will work best in specific scenarios for your systems, it is recommended that developers read the ehcache documentation located at http://ehcache.sourceforge.net/documentation/#mozTocId747622.

Using Custom OR Mappings

ORM using Hibernate allows you to serialize/de-serialize object queries to and from relational database result sets. If you did not create a data model as described in *Chapter 5 Creating the UML Models*, (for reasons such as you already have a database schema), then you must do a manual data mapping.

ORMs are defined in an XML document. For example, if you want to manually create an ORM for a Gene object in the provided example model, then you must create an XML file similar to that shown in *Figure 9.3*.

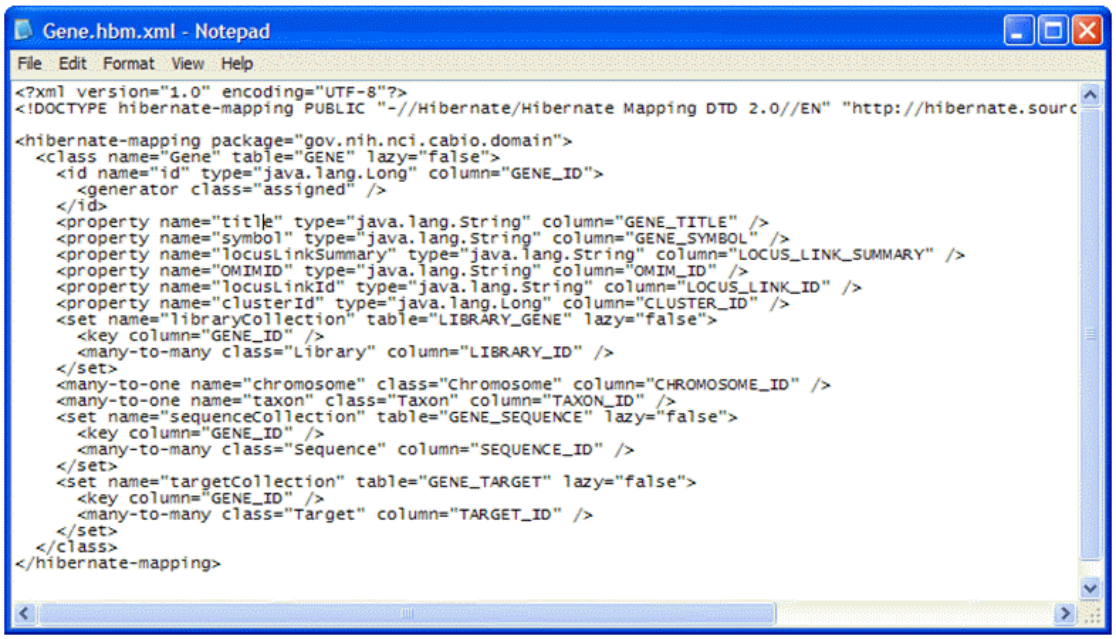


Figure 9.3 Hibernate ORM

Perform the following steps to use a manual ORM with the caCORE SDK.

1. Create a manual ORM file for each domain object as shown in *Figure 9.3*. For a detailed explanation of how ORM works using Hibernate (http://www.hibernate.org/hib_docs/reference/en/html/), see Chapter 5 Basic O/R Mapping (http://www.hibernate.org/hib_docs/reference/en/html/mapping.html).
2. Create a new directory called `{home_directory}/custom/{project_name}`.

3. Save the manually created ORM files to the directory you just created, using the same package structure as the domain objects themselves. See *Figure 9.4* for an example from the provided sample model.

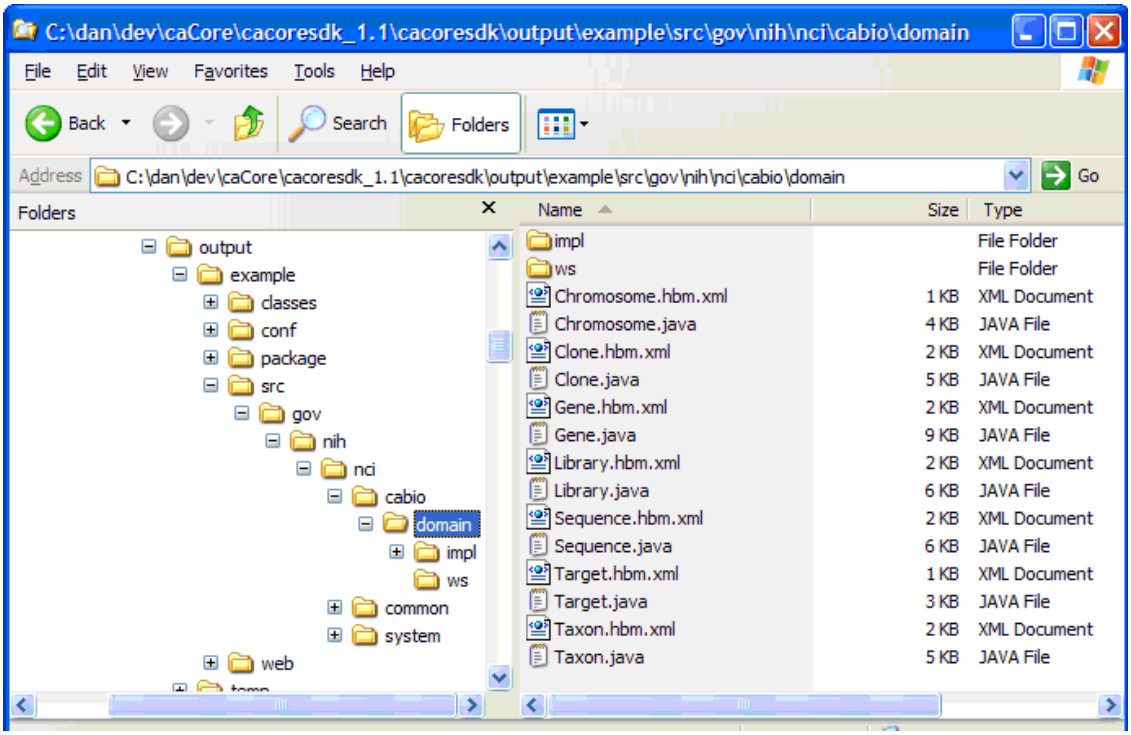


Figure 9.4 ORM directory structure

4. From the **Command Prompt**, go to your home directory (for example, in Windows `c:\cacoretoolkit`) and run the ant build-system command.

   The SDK will attempt to generate ORM files from the UML model but will override any auto-generated files with the ones that it finds in the custom directory. In this manner, you can provide custom OR mappings for as few or as many domain objects as necessary.

Using Custom Classes

Developers may wish to replace some or all of the automatically generated Java class files representing the domain objects in their model with files from other sources (hand-coded or generated by another tool).

Perform the following steps to use custom class files with the caCORE SDK.

1. Create a directory called `{home_directory}/custom/{project_name}`, if it does not already exist.
2. Save each custom class file to the directory you just created, using the appropriate package structure.
3. Run the ant `build-system` or the ant `fix-xmi` command from the project's home directory.

The SDK will overwrite its generated classes with the ones that it finds in the custom directory and use those to build the server and client packages.

Implementing a Custom XMI Preprocessor

The caCORE SDK requires XMI files used by the semantic integration tools and the Code Generator to be valid NetBeans Metadata Repository (MDR) files. Currently, this means that not all files saved or exported by UML modeling tools can be natively used by the SDK—even if they state conformance with UML 1.3 and the XMI 1.1 standard. (Additional information about NetBeans MDR can be found at http://mdr.netbeans.org/.)

The SDK framework includes an extensible mechanism to convert, or "fix" XMI files to the proper structure. Out of the box, support is provided for fixing files output by Enterprise Architect. In addition, an implementing class is provided that does no actual processing (i.e. the input XMI is simply saved as the output file). The class diagram for the XMI preprocessor component is shown in *Figure 9.5*.

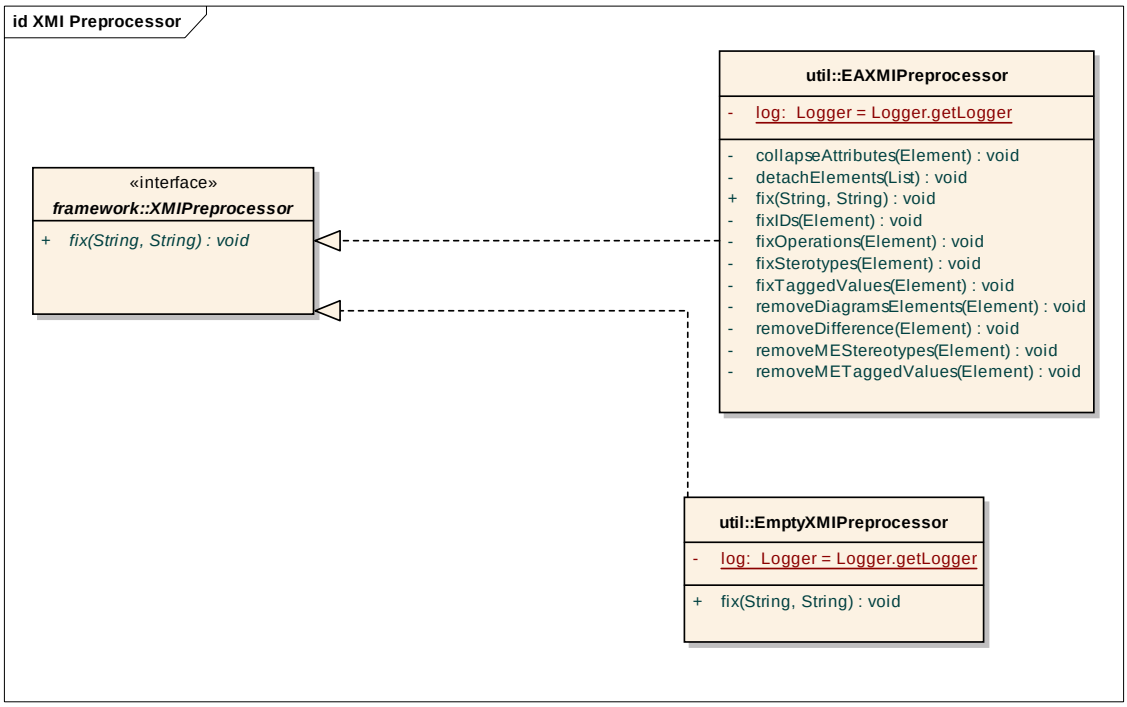


Figure 9.5 Class diagram for the XMI preprocessor component

Classes implementing the XMIPreprocessor interface must expose a fix() method which takes two strings as parameters: the full path and file name of the input XMI, and the full path and file name of a new XMI file to output. If directed to do so in the deploy.properties file, the SDK build script will call the implementing class and provide it with the input and output filenames to be used.

To use a custom implementation of the XMI preprocessor:

1. Compile the preprocessor implementation and archive it to a jar file
2. Copy the jar file to the `{home_directory}/lib` directory (or otherwise make sure that it is located on the classpath used by the SDK)
3. In deploy.properties, set `xmi_preprocessor` to the fully-qualified name of the implementing class (e.g., `org.example.xmi.MyToolXMIPreprocessor`) and make sure that fix_xmi=yes
4. Run ant `build-system` or ant `fix-ea`.

Customizing the Build Process

Controlling the Build Process

The Ant build script, `build.xml`, allows highly granular control over the entire build process. Table *9.3* describes the targets and their functions.

Ant Target	*Description*
`build-artifacts`	Generates artifacts from model
`build-beans`	Generates beans from model, copies custom beans and compiles all beans
`build-framework`	Compiles and packages SDK framework files
`build-orm`	Generates beans from model, copies custom beans and compiles all beans
`build-schema`	Creates database user and loads schema and data (when specified in deploy.properties)
`build-system`	Runs entire SDK code generation process
`clean-all`	Removes all generated code and all artifacts related to all projects
`compile-source`	Compiles the generated system
`compile-generator`	Compiles the SDK code generator classes
`copy-custom-beans`	Adds custom-written beans to the project src directory
`copy-custom-orm`	Adds custom-written OR mapping files to the project directory
`copy-project-framework`	Copies SDK framework files to project directory, adding project-specific properties
`copy-server-file`	Copies .war file to J2EE container deployment directory
`copy-template-files`	Copies code generation templates to project output directory
`create-control-files`	Creates control files required by code transformers for the project specified in deploy.properties
`create-output-dirs`	Creates output directories for the SDK framework
`create-project-dirs`	Creates output directories for the project specified in deploy.properties
`deploy-server`	Deploys system to J2EE container
`deployWS`	Deploys Web services to system described in deploy.properties
`disable-writable-api`	Disables functions that enable write functionality to the API
`doc`	Creates Javadoc HTML documentation for both the SDK framework and the system defined in deploy.properties
fix-xmi	Pre-processes XMI file to ensure that it can be parsed by NetBeans MDR component
`format`	Formats generated source code
`generate-OR-mapping`	Generates OR mapping from model

Table 9.3 Targets and their functions in the Ant build script

Ant Target	Description
generate-artifacts	Generates configuration and other required system files from model
generate-beans	Generates beans based on model specified in deploy.properties
generate-cache	Generates cache configuration files
generate-common-package-util	Generates common package utility class
generate-common-role-util	Generates common role utility class
generate-dao-conf	Generates DAO configuration files from model
generate-junit	Generates JUnit test classes from model
generate-orm-conf	Generates ORM configuration files from model
generate-schemas	Generates XML schemas from model
generate-wsdd	Generates web services deployment descriptor
generate-xml-mapping	Generates XML mapping files
help	Default target that lists commonly used targets within this build script
init-project	Initializes project build by creating output directories and necessary files
jetc	Compiles Java JET templates for the project specified in deploy.properties
package-client	Creates client.zip file
package-framework	Packages compiled SDK framework classes as a jar
package-server	Creates .war file for server
package-system	Creates .war file for server and client.zip file for client
runWSdemo	Runs Web Services-based demo client for provided SDK example system located in the build.xml file within the client subdirectory
runXMLdemo	Runs demo of XML conversion utility on provided SDK example system located in the build.xml file within the client subdirectory
rundemo	Runs Java-based demo client for provided SDK example system located in the build.xml file within the client subdirectory
runtest	Runs JUnit test located in the build.xml file within the thick-client subdirectory
semantic-connector	Runs Semantic Integration Workbench using Java Web Start
show-properties	Display a list of all properties currently set
undeployWS	Un-deploys Web services to system described in deploy.properties

Table 9.3 Targets and their functions in the Ant build script (Continued)

Modifying Build Locations and Filenames

All of the file and path names used by the build.xml file are contained in Ant properties located in a separate file called build.properties. This is a standard Java property file that can be consumed by the java.util.Properties class (see http://java.sun.com/j2se/1.4.2/docs/api/java/util/Properties.html for more information).

The build.properties file is broken down into five sections that define properties relating to the SDK directories, the developer's project, files used by the SDK, external URLs, and custom build directives. Developers can use these settings to make changes to output directories, control code generation, etc., but should employ particular care when modifying this file in any way.

Customizing Build Targets

Note: It is recommended that developers wishing to use this feature have a full understanding of the Ant paradigm and be well-versed in creating and modifying Ant scripts.

In addition to the control provided at the target level, the build script is fully customizable by developers wishing to modify the specific tasks that occur at each stage in the process. Changes to the build process can be of three kinds: adding tasks within a target, replacing the set of tasks within a target, or skipping the execution of the tasks in a target.

In order to make these changes possible, each target within the build process follows the model shown in *Figure 9.6*. For each target accessible to the command line, there are two private targets (main and custom) executed as dependencies to the public target. The main target contains the tasks required to build a caCORE-like system out-of-the-box. The custom targets are empty and can be modified by developers wishing to modify the process.

The execution of each of these targets is controlled by property values. By default, the main target is always called and the custom target is not.

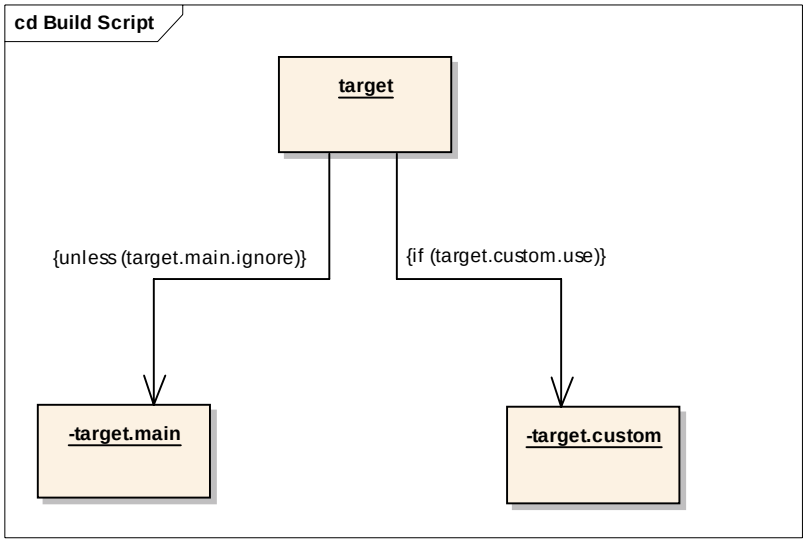


Figure 9.6 Targets within the build process

The following segments of code demonstrate the structure of each target. The first section can be found in build.xml:

```
<target name="deployWS"
   depends="-deployWS.main,-deployWS.custom"
   description="Deploy Web services"/>

<target name="-deployWS.main" unless="deployWS.main.ignore">
   <SDKecho message="Deploying web services"/>
   <java classname="org.apache.axis.client.AdminClient"
   fork="true">
      <classpath refid="classpath"/>
      <arg value="-l${url.project.axisservice}"/>
      <arg value="${dir.output.project.conf}/
      ${file.wsdd.deploy}"/>
   </java>
</target>
```

The following comes from the custom build script file:

```
<target name="-deployWS.custom" if="deployWS.custom.use"/>
```

To modify the flow of the script, two control properties can be defined for each target in a separate custom properties file. Reference to these properties is made to attributes of the main and custom targets, respectively. The effect of setting these properties are summarized in Table *9.4*.

target.main.ignore is commented out (i.e. not set) *target*.custom.use is commented out (i.e. not set)	-*target*.main will execute -*target*.custom will not execute
target.main.ignore = yes *target*.custom.use is commented out	-*target*.main will not execute -*target*.custom will not execute
target.main.ignore = yes *target*.custom.use = yes	-*target*.main will not execute -*target*.custom will execute
target.main.ignore is commented out *target*.custom.use = yes	-*target*.main will execute -*target*.custom will execute

Table 9.4 Control properties for targets

The filenames for the custom build and properties files are defined in the main build.properties file. By default, these files are called `build-custom.xml` and `build-custom.properties` and a skeleton version of each of these is packaged with the SDK.

Generating a Writable API for an Application

In previous releases, the caCORE SDK could only generate readable APIs that allowed various mechanisms to query the underlying data. The caCORE 3.2 SDK provides the capability to write, modify or delete data in the database.

In order to enable the writable APIs, the disable_writable_api_generation should be set to "no" while generation of the application using caCORE SDK. Enabling these APIs

will add the following new methods to the ApplicationService Interface of the caCORE SDK generated system.

```
public abstract Object createObject(Object object) throws ApplicationException;

public abstract Object updateObject(Object object) throws ApplicationException;

public abstract void removeObject(Object object) throws ApplicationException;

public abstract List getObjects(Object object) throws ApplicationException;
```

Figure 9.7 Methods added to ApplicationService for writable APIs

All of these methods take generic java Object as input. These objects can be any of the domain objects of the generated application. The return type of these methods is also a generic java Object that should be typecast into an appropriate domain object.

The *createObject* method allows the client application to create a new object. It accepts a domain object as input and persists it into the database table (determined by the OR mapping). Then this method returns the newly created object to the client application. The returned object now includes values that were auto generated by the database. If there is any error during the creation of the domain object then an ApplicationException is thrown explaining the error.

The updateObject method allows the client application to update an existing object. It accepts any domain object as input then, updates it in the database. The updated object is then returned back to the client application. If an error occurs during the updating of the domain object, then an *ApplicationException* is thrown explaining the error.

The *removeObject* method accepts a domain object which needs to be removed from the underlying database. It does not return a value. If an error occurs during the deletion of the domain object, then an *ApplicationException* is thrown explaining the error.

The *getObjects* method allows the client application to query for a list of domain objects of a particular type. It accepts a domain object as input whose attribute values is used as query criteria. The resulting set is returned to the client application in the form of a *ListProxy* implementation of java List Object. If an error occurs during the querying operation, then an *ApplicationException* is thrown explaining the error.

CHAPTER

10

INTEGRATING CSM WITH A CACORE SDK GENERATED APPLICATION

This chapter describes how to configure and use the integrated security provided by NCICB's Common Security Module (CSM) services in a caCORE SDK generated application.

Topics in this chapter include:

CSM Overview

This chapter addresses the following topics:

- Security - This topic includes CSM's mechanisms for authentication, authorization, and user provisioning. For more details on CSM itself, refer to the CSM Chapter in the caCORE Technical Guide.
- Session management - Session management eliminates the need to authenticate every request sent to the server. Session management also facilitates tracking the user on the server.

The NCICB Common Security Module (CSM), first developed for the caCORE 3.0 release, provides a flexible solution for application security and access control. CSM provides a common starting point for any development team that has security require-

ments, and thus helps to avoid duplication of effort and inconsistent security implementations.

CSM has three main functions:

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

CSM's integration with the caCORE SDK requires configuring CSM for your application. For instructions on configuring CSM, refer to the CSM Guide for Application Developers on the CSM section of the NCICB Downloads page (http://ncicb.nci.nih.gov/download/downloadcsm.jsp). After installing and configuring CSM, application administrator(s) will use the User Provisioning Tool (UPT) (see the UPT User Guide.pdf at http://ncicb.nci.nih.gov/download/downloadcsm.jsp) to create an authorization policy for the application. An authorization policy is the knowledge of what to protect. Within the UPT, users can be given different roles (and permissions) for domain objects. Any change in the authorization policy is reflected in the application at run time. This means the CSM service continuously provides the latest authorization policy to the application service.

The caCORE SDK generates two components - client and server. Only the server component is integrated with CSM for the purpose of authentication and authorization. The CSM service integration is not obtrusive; there is a flag to turn the CSM service on or off.

Session Management Overview

The session management service has been provided as a part of the CSM solution. Once a user successfully logs into an application, the Session Manager (on the server side) generates a unique key for the user session. When the user sends another request, the server does not ask the user to authenticate again as long as the session has not expired. Application administrators can configure the duration of the session timeout by setting the `default_session_timeout` property in the deploy.properties file.

Configuring CSM for the Generated Application

Instructions for CSM configuration can be found in the CSM Guide for Application Developers at http://ncicb.nci.nih.gov/download/downloadcsm.jsp.

- This service uses the Authentication and Authorization services provided by CSM. For this configuration, follow the Authentication and Authorization Deployment sections of the CSM Guide.
- The application context name should match the name used in the `deploy.properties` file (mentioned in a previous step).
- To enable security the `default_security_level` property in the `deploy.properties` file should be set to 1 at the time of the generation of the application.

Configuring the Application's Authorization Data Using UPT

- The domain objects in your application should be created as protection elements in the UPT. The fully qualified class name of the domain object (no need to specify the Impl classes as separate protection elements) is used as the object ID for each protection element.
- Application administrators will be aware of the authorization policy for these protection elements. They can grant correspondingly appropriate privileges on the domain objects.
- The writeable APIs use the authorization schemes listed below. They use the name of the domain objects passed to them as protection element object IDs. Based on the operation it is performing, the corresponding privileges are used while invoking the `checkPermission` method of a CSM API. This determines if the user has access privileges. Since all of the methods of ApplicationService are query operations, the privilege used for all of them is "READ". Below is a list of methods and the corresponding privilege.

```
createObject - uses the privilege "CREATE"
updateObject - uses the privilege "UPDATE"
removeObject - uses the privilege "DELETE"
getObjects - uses the privilege "READ"
```

All other Methods of ApplicationService use the privilege "READ"

The UPT User Guide at http://ncicb.nci.nih.gov/download/downloadcsm.jsp provides a detailed description of how to use and configure authorization data using the UPT.

Using the CSM-Enabled ApplicationService API

The code example below shows how to query a caDSR domain object called "DataElement."

1. To start the client session, enter the userId and password.
2. Obtain a reference to the application service. This reference is provided by the *ApplicationServiceProvider* class.
3. Once a reference to the service is obtained, all methods on the service are available. Multiple methods of the ApplicationService can be invoked within a session. There is no need to initialize the session before every method call.

Using the CSM-Enabled ApplicationService Web Services

The code example below shows how to query an example domain object called "Gene" using the webservices that are secured using CSM.

1. To invoke the secured web services, first create a web services call pointing to the server hosting the services and setting various other required arguments as shown in the example below.
2. Then create a SOAPHeaderElement and name it "CSMSecurityHeader".
3. Set the prefix of the header element to "csm".

4. Add child elements to supply the user credentials. The user name should be provided as element name "username" whereas the password should be provided as element "password".
5. Once the header element is populated, add it to the SOAP call and continue with querying the service to obtain the result.
6. If there are any Authentication or Authorization errors, the server will respond accordingly with an error message. Otherwise, a response is returned.

```
import java.net.URL;
import org.apache.axis.AxisFault;
import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import org.apache.axis.message.SOAPHeaderElement;
import org.apache.axis.utils.Options;

import javax.xml.namespace.QName;
import javax.xml.rpc.ParameterMode;
import javax.xml.soap.SOAPElement;

import gov.nih.nci.cabio.domain.ws.*;

import java.util.HashMap;

public class WSTestClient
{

  public static void main(String [] args) throws Exception {

    Service  service = new Service();
    Call     call    = (Call) service.createCall();
    /*****************************************************/

    QName qnGene = new
QName("urn:ws.domain.cabio.nci.nih.gov", "Gene");
    call.registerTypeMapping(Gene.class, qnGene,
        new
org.apache.axis.encoding.ser.BeanSerializerFactory(Gene.class
, qnGene),
        new
org.apache.axis.encoding.ser.BeanDeserializerFactory(Gene.cla
ss, qnGene));

    /
*************************************************************/

    String url = "http://localhost:8080/example/ws/
exampleService";

    call.setTargetEndpointAddress(new java.net.URL(url));
    call.setOperationName(new QName("exampleService",
"queryObject"));
```

```
    call.addParameter("arg1",
org.apache.axis.encoding.XMLType.XSD_STRING,
ParameterMode.IN);
    call.addParameter("arg2", qnGene, ParameterMode.IN);

call.setReturnType(org.apache.axis.encoding.XMLType.SOAP_ARRA
Y);
    SOAPHeaderElement headerElement = new
SOAPHeaderElement(call.getOperationName().getNamespaceURI(),"
CSMSecurityHeader");
    headerElement.setPrefix("csm");
    headerElement.setMustUnderstand(false);
    SOAPElement usernameElement =
headerElement.addChildElement("username");
    usernameElement.addTextNode("userId");
    SOAPElement passwordElement =
headerElement.addChildElement("password");
    passwordElement.addTextNode("password");
    call.addHeader(headerElement);
    gov.nih.nci.cabio.domain.ws.Gene gene = new
gov.nih.nci.cabio.domain.ws.Gene();
    gene.setSymbol("IL*");

    try {
      Object[] resultList = (Object[])call.invoke(new
Object[] { "gov.nih.nci.cabio.domain.ws.Gene", gene });

      System.out.println("Total objects found: " +
resultList.length);

      if (resultList.length > 0) {
        for(int resultIndex = 0; resultIndex <
resultList.length; resultIndex++) {
          Gene returnedGene = (Gene)resultList[resultIndex];
          System.out.println(
            "Symbol: " + returnedGene.getSymbol() + "\n" +
            "\tName " + returnedGene.getTitle() + "\n" +
            "");
        }
      }
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Using the CSM-Enabled HTTP Interface

If security is turned on, then both HTTP interfaces, getHTML (Happy.jsp) and getXML, are secured. In order to access them, user credentials must be provided.

getHTML Interface

The get HTML interface can be accessed by invoking Happy.jsp. Once the object to be queried is selected and criteria is entered, a username and password are required in the javascript windows. These credentials are appended to the query that is routed to the server and used to authenticate and authorize the user. If there is a security error, it is displayed in the browser. Otherwise, the queried result is displayed. The returned result set can be accessed by clicking the links. However, if a new browser is opened, reauthentication is required.

Alternatively credentials can be provided directly in the browser's address bar or as a URL of an HTTP Request to obtain the result as shown below. The following is an example URL to obtain all the chromosomes from the example model of the SDK.

```
http://localhost:8080/example//
GetHTML?query=gov.nih.nci.cabio.domain.Chromosome&gov.nih.
nci.cabio.domain.Chromosome&username=<<USERNAME>>&password
=<<PASSWORD>>
```

getXML Interface

This interface can be accessed by generating an HTTP Request either programmatically or via a browser to obtain the result in an XML format. User credentials must be appended to the actual query as shown below.

```
http://localhost:8080/example//
GetXML?query=gov.nih.nci.cabio.domain.Chromosome&gov.nih.n
ci.cabio.domain.Chromosome&username=<<USERNAME>>&password=
<<PASSWORD>>
```

7. When finished, call `terminateSession` to end the session on the server (*Figure 10.1*).

```
import java.util.Date;
import java.util.List;
import gov.nih.nci.cadsr.domain.impl.DataElementImpl;
import gov.nih.nci.system.applicationservice.ApplicationService;
import gov.nih.nci.system.applicationservice.ApplicationServiceProvider;
import gov.nih.nci.system.comm.client.ClientSession;

import org.hibernate.criterion.DetachedCriteria;
import org.hibernate.criterion.Expression;

public class TestClient
{

  public static void main(String[] args)
  {
    ApplicationServiceProvider applicationServiceProvider = new
ApplicationServiceProvider();
    ApplicationService appService = applicationServiceProvider.getApplicationService();
    ClientSession cs = ClientSession.getInstance();
    try
    {
      cs.startSession("userId", "password");
    }
    catch (Exception ex)
    {
      System.out.println(ex.getMessage());
    }
    try
    {
      DetachedCriteria deCrit = DetachedCriteria.forClass(DataElementImpl.class);
      deCrit.add(Expression.eq("publicID", new Long(2199715)));
      int count = appService.getQueryRowCount(deCrit, DataElementImpl.class.getName());
      String val = String.valueOf(count);
      System.out.println("The size of the records is " + val);
      List listR = appService.query(deCrit, DataElementImpl.class.getName());
      System.out.println("The size of the records is second time is " + listR.size());
      cs.terminateSession();
    }
    catch (Exception e)
    {
      e.printStackTrace();
    }
  }
}
```

Figure 10.1 Using the CSM-enabled caCORE generated application in a client application

APPENDIX

A

UNIFIED MODELING LANGUAGE

The caCORE team bases its software development primarily on Unified Modeling Language (UML). This appendix is designed to familiarize the reader who has not worked with 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 is composed of nine different types of modeling diagrams, which form, in essence, a software blueprint.

Only a subset of the diagrams, those used in caCORE development, is described in this chapter.

- 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 locations of source modules and the allocations 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 chapter are necessary for using the caCORE SDK. They are included in this chapter to provide a more complete overview of UML.

While many good development tools provide support for generating UML diagrams, the Enterprise Architect (EA) software was used to create the screen shots in the *caCORE Software Development Kit Programmer's Guide*. 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)

Usecase ID:100300

Actor

- caBIO Application developer

Starting Condition

The actor establishes reference to the caBIO software

Flow of Events

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.

End Condition

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

Figure A.1 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.* 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 *Figure A.2* can be interpreted as follows:

- A caBIO application triggers the actions to build a search query, connect to server, and search server.

- The caBIO application receives the output from the search.

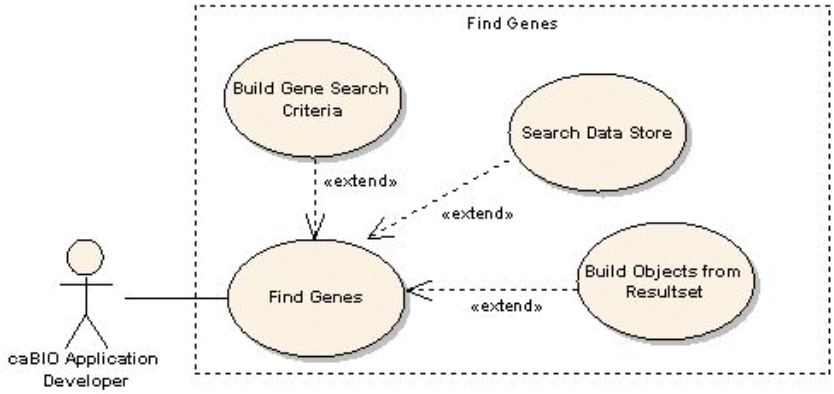


Figure A.2 Building a search query use-case

Class Diagrams

The system designer uses 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.

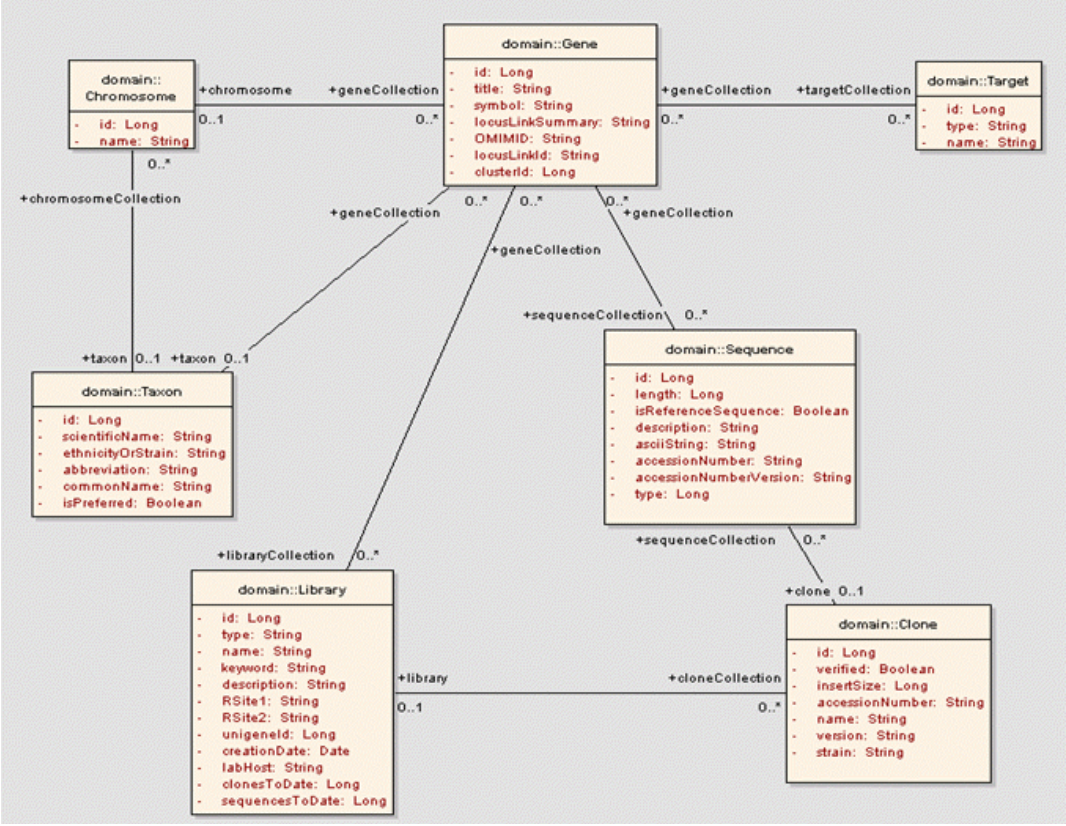


Figure A.3 caBIO class diagram

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().*

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

Naming Conventions

Naming conventions are very important when creating class diagrams. The 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 9. 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 11. 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.

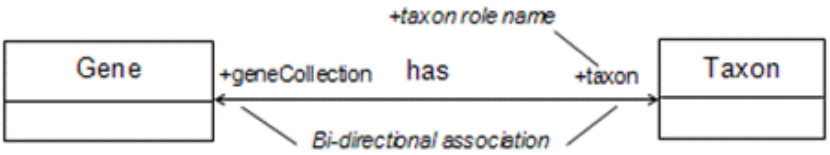


Figure A.5 A one-to-one association with bi-directional navigability

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 relation 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.

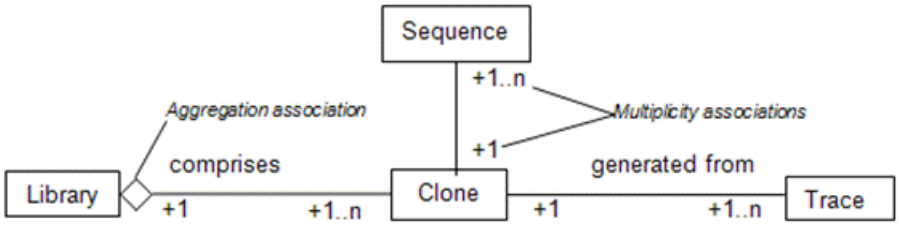


Figure A.7 Aggregation and multiplicity associations

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*.

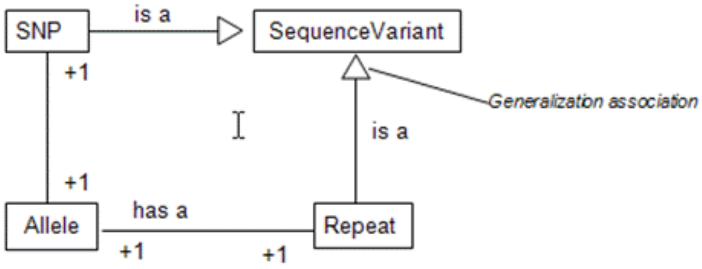


Figure A.8 Generalization relationship

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 UML 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 pack-

age depends on another if changes in one could force changes in the other. This figure is the hierarchical representation of *Figure A.9*.

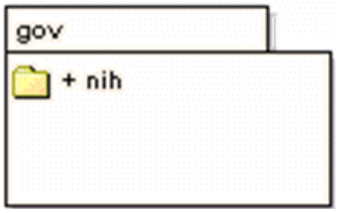


Figure A.9 Package diagram generated in EA

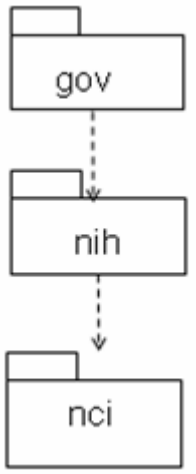


Figure A.10 Hierarchical representation of a package diagram

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 web published pages approximately reflect the actual Java package structure but do not have a one-to-one correspondence.

on the web published 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 an executable component.

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 the runtime. The User query from the client traverses the following sequence path before reaching the database.

4. The user uses search() method in ApplicationService and queries the server.
5. This call is picked up at HTTPClient as query() with Request as the input parameter
6. HTTPClient calls the HTTPServer (Interface Proxy for HTTP Tunneling) and sends the same Request to BaseDelegate
7. BaseDelegate calls ServiceLocator to find the name of Data Access Object.
8. Using this name BaseDelegate creates the corresponding DAO factory and passes the Request object.

9. In this scenario the ORMDAO is the right DAO to be called.
10. 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.

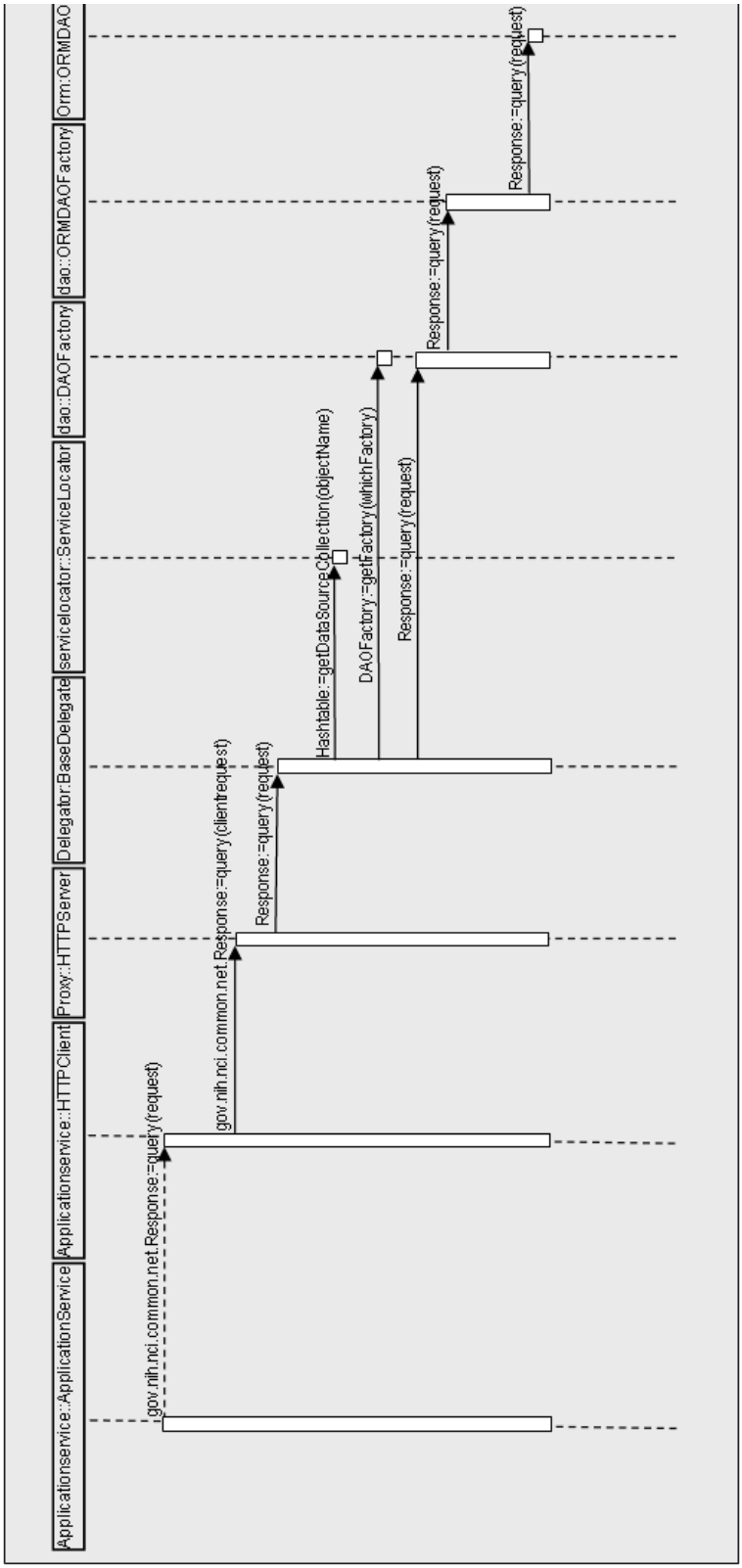


Figure A.13 Sequence Diagram

APPENDIX B

SOFTWARE CONFIGURATION MANAGEMENT

This appendix describes the defined set of software configuration management (SCM) processes centered on a number of open source tools developed by NCICB. In particular, NCICB uses CVS (https://www.cvshome.org/) for version control and Ant (http://ant.apache.org/) for build management.

The SCM procedures in place at NCICB are documented in a series of white papers, including the following:

- SCM – Project Charter
- SCM – NCICB Change Control Plan
- SCM – Version Control Guidelines
- SCM – Deployment Guidelines
- SCM – CVS Users Guide
- SCM – Ant Users Guide

These documents are publicly available and can be obtained by following the appropriate links on the Programming and API Support page at http://ncicbsupport.nci.nih.gov/sw/.

Very broadly, an SCM practice must address the following four primary functions:

1. Configuration identification: consists of identifying those elements (configuration items) of a system that are to be managed. As a good rule of thumb, all non-derived resources (object files, executables, or any other resource that can be derived from a controlled resource) should be managed.
2. Configuration control: consists of the evaluation, coordination, approval/disapproval, and implementation of changes to configuration items.
3. Status accounting: consists of the recording and reporting of information needed to manage a configuration efficiently – for example, the status of proposed changes and implementation of approved changes.

4. Audits and reviews: consist of activities carried out to ensure that the SCM system is functioning correctly, and to ensure that the configuration has been tested to demonstrate that it meets its functional requirements and that it contains all deliverable entities.

A full discussion of identification, accounting, and auditing is beyond the scope of this document, but the notion of configuration control includes the following core concepts:

Version control — refers to the mechanisms used to keep track of the history of changes to a product component (configuration item) throughout the software development life cycle. There are numerous tools available, both open source and commercial, to support version control, but the open source tool CVS is used at NCICB for this purpose.

Build management — refers to the discipline of efficiently building the whole or a subset of a version of a product from the selected configuration of product components. The open source tool Ant is used at NCICB for this purpose.

Change control — refers to the discipline of evaluating, coordinating, approving or disapproving, and implementing changes to artifacts that are used to construct and maintain a software system. NCICB has an organization-wide change control board that meets regularly to discuss and evaluate the impact of software change requests.

Version control, build management, and change control are the key processes of a successful software configuration management practice, and are the baseline processes that should always be in place for any software development project.

APPENDIX

C

PERFORMANCE ISSUES

This appendix explains a known performance issue with Hibernate v3.0.5.

Hibernate Issue with Enabling Security in the SDK

In order to become CFR 21 / part 11 compliant, the CSM APIs had to implement Audit Logging, which has to log each and every operation performed by the user on the data stored in the security schema. It also has keep track of all the changes made to the data. To aid in doing so, CSM uses features provided by the Common Logging Module (CLM). The CLM APIs are wrapped over log4j and uses it to log into a database through a JDBC Appender.

In order to track the changes performed to the object states, CLM uses Hibernate's Interceptor Model to intercept all the user actions. It intercepts all the create, update, and delete operations performed by the user on the objects. Based on the operation that the user is performing, it creates audit logs and stores the appropriate object state.

However, the Interceptor Model is available in Hibernate v3.0.5 and above. caCORE SDK has traditionally used Hibernate v3.0.2. During the merging of the CSM SDK Adapter features into the caCORE SDK project, there were issues discovered with the usage of the Hibernate v3.0.5. It was enabling the eager fetching of the child objects even if the lazy initialization was enabled. As the result of this, there were performance issues when the queried object had many associated child objects.

To overcome this issue, the caCORE SDK (as well as the caCORE APIs) is packaged and distributed with Hibernate v3.0.2 by default. Hence, for all regular users who do not want to enable security, this version provides optimal performance.

The projects that need security services provided by CSM must upgrade to Hibernate v3.0.5. This might result in the eager fetching issue mentioned above and cause a performance issue. However this performance issue becomes evident only in the case of objects that have a huge collection of associated objects.

APPENDIX D
REFERENCES

Technical Manuals/Articles

1. National Cancer Institute. *caCORE 3.2 Technical Guide* http://ncicb.nci.nih.gov/NCICB/infrastructure/cacore_project_doc
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/hib_docs/reference/en/html/
6. Basic O/R Mapping: http://www.hibernate.org/hib_docs/reference/en/html/mapping.html
7. Java Programming: http://java.sun.com/learning/new2java/index.html
8. Jalopy User Manual: http://jalopy.sourceforge.net/manual.html
9. Javadoc tool: http://java.sun.com/j2se/javadoc/
10. JDiff: http://javadiff.sourceforge.net/
11. JUnit http://junit.sourceforge.net/
12. Extensible Markup Language: http://www.w3.org/TR/REC-xml/
13. XML Metadata Interchange: http://www.omg.org/technology/documents/formal/xmi.htm
14. Ehcache: http://ehcache.sourceforge.net/documentation/

Scientific Publications

1. 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.
2. 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.
3. 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/)

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
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
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/

GLOSSARY

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

Term	*Definition*
`{home_directory}`	Indicates the directory where you installed the SDK
`{package_structure}`	Indicates the package structure from the UML models
`{project_name}`	Indicates the `project_name` specified in the `deploy.properties` file
AGT	Artifact Generation Tool
AOP	Aspect Oriented Programming
API	Application Programming Interface
Writeable API	Methods exposed by the CSM to create, update and delete a domain object. These methods are generated using the code generation component.
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
CASE	Computer Aided Software Engineering
CCR	Center of Cancer Research
CDE	Common Data Element
CGAP	Cancer Genome Anatomy Project
CMAP	Cancer Molecular Analysis Project
CODEGEN	Code generator tool

Term	Definition
Code Generator Tool	The SDK tool that leverages Model-Driven Architecture to convert a UML model to a fully-functioning caCORE-like system
CS	Classification Scheme
CSI	Classification Scheme Item
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
DDL	Data Definition Language
DEC	Data Element Concept
DOM	Document Object Model
DTD	Document Type Definition
DU	Deployment Unit
EA	Enterprise Architect
EBI	European Bioinformatics Institute
EMF	Eclipse Modeling Framework
EVS	Enterprise Vocabulary Services
FK	Foreign Key - A collection of columns (attributes) that enforce a relationship to a Primary Key in another table used in data model tables in Enterprise Architect
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
IDE	Integrated Development Environment
ISO	International Organization for Standardization
JAF	JavaBeans Activation Framework
Jalopy	Source code formatting tool for the Sun Java Programming Language (http://jalopy.sourceforge.net/manual.html)
JAR	Java Archive
Javadoc	Tool for generating API documentation in HTML format from doc comments in source code (http://java.sun.com/j2se/javadoc/)
JDBC	Java Database Connectivity
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

Term	Definition
JSP	JavaServer Pages
JUnit	A simple framework to write repeatable tests (http://junit.sourceforge.net/)
MDR	Metadata Repository
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
NCI	National Cancer Institute
NCICB	National Cancer Institute Center for Bioinformatics
NSC	Nomenclature Standards Committee
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.
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
PK	Primary Key – Key used to uniquely identify a data model table in Enterprise Architect.
QA	Quality Assurance
RDBMS	Relational Database Management System
RUP	Rational Unified Process
SCM	Software Configuration Management
SDK	Software Development Kit
Semantic Connector	The SDK tool that links model elements to NCICB EVS concepts.
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
UML	Unified Modeling Language
UML Loader	SDK tool that converts a domain model in UML to corresponding common data elements (CDEs) in caDSR.
UPT	User Provisioning Tool
URI	Uniform Resource Identifier
URL	Uniform Resource Locators
WSDL	Web Services Description Language

Term	Definition
XMI	XML Metadata Interchange (http://www.omg.org/technology/documents/formal/xmi.htm) - The main purpose of XMI is to enable easy interchange of metadata between modeling tools (based on the OMG-UML) and metadata repositories (OMG-MOF) in distributed heterogeneous environments
XML	Extensible Markup Language (http://www.w3.org/TR/REC-xml/) - XML is a subset of Standard Generalized Markup Language (SGML). Its goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML
XP	Extreme Programming

INDEX

D

T

U

V

X