

CANCER DATA STANDARDS REGISTRY (CADSR)

Release 4.0 Technical Guide



**NATIONAL[®]
CANCER
INSTITUTE**

Center for Biomedical Informatics
and Information Technology

TABLE OF CONTENTS

About This Guide	1
Purpose	1
Release Schedule	1
Audience	2
Additional Documentation	2
Topics Covered	2
Text Conventions Used	3
Credits and Resources	3
Chapter 1	
Overview of caCORE	5
caCore Architecture Overview	5
Components of caCORE	6
Enterprise Vocabulary Services (EVS)	6
Cancer Data Standards Repository (caDSR)	6
Software Development Kit (SDK)	6
caAdapter	7
Common Security Model (CSM)	7
Common Logging Module (CLM)	7
Chapter 2	
caDSR Architecture	9
caDSR System Architecture	9
Client Technologies	10
caDSR Packages	11
Chapter 3	
Understanding caDSR and the caDSR API	13
Modeling Metadata: The ISO/IEC 11179 Standard	13
caDSR Metamodel	17
caDSR API	23
caDSR Domain Object Catalog	25

Downloading the caDSR	33
caDSR API Examples	34
Using the caDSR Java API	34
Using the caDSR Web Services API	36
UML Project API Examples	37

Chapter 4

Interacting With caDSR	41
Java API	41
Installation and Configuration	42
A Simple Example	44
Application Service Interface	45
<i>Convenience Query Methods</i>	46
<i>HQL Query Methods</i>	46
<i>Nested Search Criteria Query Methods</i>	47
<i>Detached Criteria Query</i>	48
<i>CQL Query</i>	49
Web Services API	51
XML-HTTP API	51

Appendix A

Understanding Unified Modeling Language (UML)	53
UML Modeling	53
Use-case Documents and Diagrams	54
Class Diagrams	55
Naming Conventions	57
Relationships Between Classes	57
<i>Association</i>	57
<i>Directionality</i>	57
<i>Multiplicity</i>	58
<i>Aggregation</i>	58
<i>Generalization</i>	59
Package Diagrams	60
Sequence Diagrams	62

Appendix B

References	65
Technical Manuals/Articles	65
Scientific Publications	65
caBIG Material	67
caCORE Material	67
Modeling Concepts	67

Applications Currently Using caCORE	67
Software Products	68
Glossary	69

ABOUT THIS GUIDE

This section introduces you to the *caDSR 4.0 Technical Guide*. It includes the following topics:

- *Purpose* on this page
- *Release Schedule* on this page
- *Audience* on page 2
- *Additional Documentation* on page 2
- *Topics Covered* on page 2
- *Text Conventions Used* on page 3
- *Credits and Resources* on page 3

Purpose

The *caCORE caDSR 4.0 Technical Guide* describes the Cancer Data Standards Registry (caDSR) component of the Cancer Common Ontologic Representation Environment (caCORE).

caCORE is an open-source, standards-based, semantics-computing environment and toolset created by the National Cancer Institute (NCI) Center for Biomedical Informatics and Information Technology (CBIIT).

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

Release Schedule

This guide is updated for each caDSR release. It may be updated between releases if errors or omissions are found. This version reflects the caDSR 4.0, released October 2008 by the NCI CBIIT (formerly the National Cancer Institute Center for Bioinformatics (NCICB)).

Audience

The primary audience of this guide is the application developer who wants to learn about the architecture of caDSR and to access and/or use the caDSR APIs. caDSR is generated using the caCORE Software Development Kit (SDK). For more information, see the caCORE SDK 4.0 Developer's Guide hosted on [GForge](#).

The caDSR 4.0 Technical Guide assumes familiarity with the Java programming language and/or other programming languages, database concepts, and the Internet. For consuming caDSR resources in software applications, experience with building and using complex data systems is also assumed. Neither the caDSR API nor this documentation is intended for health professionals and members of the general public who do not have the requisite software development experience noted previously.

Additional Documentation

The caDSR 4.0 Release Notes contain a description of the end user tool enhancements and bug fixes included in this release.

The caCORE SDK 4.0 Developer's Guide contains detailed instruction on the use of the SDK and how it aids in creating a caCORE-like software system.

Topics Covered

This technical guide focuses on the caDSR component of caCORE 4.0 and provides information regarding:

- The purpose, architecture, and components of caCORE and caDSR;
- The APIs for accessing the caDSR system including Java, Web services, and XML-HTTP;
- An overview of Unified Modeling Language (UML).

The following is a list of the chapters you will find in this guide and a brief description of the information provided in each of those chapters:

- *Chapter 1, Overview of caCORE*, on page 5
- *Chapter 2, caDSR Architecture*, on page 9
- *Chapter 3, Understanding caDSR and the caDSR API*, on page 13
- *Chapter 4, Interacting With caDSR*, on page 41
- *Appendix A, Understanding Unified Modeling Language (UML)*, on page 53
- *Appendix B, References*, on page 65
- *Glossary* on page 69

For details on components other than caDSR, refer to the caCORE Overview web page at http://ncicb.nci.nih.gov/NCICB/infrastructure/cacore_overview.

Text Conventions Used

This section explains conventions used in this guide. The various typefaces represent interface components, keyboard shortcuts, toolbar buttons, dialog box options, and text that you type.

Convention	Description	Example
Bold	Highlights names of option buttons, check boxes, drop-down menus, menu commands, command buttons, or icons.	Click Search .
<u>URL</u>	Indicates a Web address.	http://domain.com
text in SMALL CAPS	Indicates a keyboard shortcut.	Press ENTER.
text in SMALL CAPS + text in SMALL CAPS	Indicates keys that are pressed simultaneously.	Press SHIFT + CTRL.
<i>Italics</i>	Highlights references to other documents, sections, figures, and tables.	See <i>Figure A.3.5</i> .
<i>Italic boldface monospaced type</i>	Represents text that you type.	In the New Subset text box, enter <i>Proprietary Proteins</i> .
Note:	Highlights information of particular importance	Note: This concept is used throughout the document.
{ }	Surrounds replaceable items.	Replace {last name, first name} with the Principal Investigator's name.

Credits and Resources

The following people contributed to the development of this document.

caDSR Development and Management Teams		
Development	Documentation	Project and Product Management
Denis Avdic ³	Bronwyn Gagne ²	Stephen Alred ⁴
Rebecca Angeles ³	Ann Wiley ⁵	Larry Hebel ³
Dave Hau ³	Jill Hadfield ¹	Denise Warzel ¹
¹ National Cancer Institute Center for Bioinformatics and Information Technology (CBIT)	² Lockheed Martin	³ ScenPro, Inc.
⁴ Oracle	⁵ Ann Wiley Consultants, Inc.	⁶ Ekagra Software Technologies

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

CHAPTER 1

OVERVIEW OF caCORE

The NCI Center for Biomedical Informatics and Information Technology (CBIIT) (formerly NCICB) provides biomedical informatics support and integration capabilities to the cancer research community. CBIIT has created a core infrastructure called Cancer Common Ontologic Representation Environment (caCORE).

This chapter provides an overview of the NCI CBIIT caCORE infrastructure.

caCore Architecture Overview

caCORE is a data management framework designed for researchers who need to be able to navigate through a large number of data sources. By providing this common framework, caCORE helps streamline the informatics development throughout academic, government and private research labs and clinics.

The components of caCORE support the semantic consistency, clarity, and comparability of biomedical research data and information. caCORE provides open-source enterprise architecture for NCI-supported research information systems built using formal techniques from the software engineering and computer science communities.

The four characteristics of caCORE include:

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

Systems with these properties are said to be “caCORE-like.”

When all four of these development principles are addressed, the resulting system has several desirable properties:

- 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.
- 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.
- Most importantly, the system is *semantically interoperable*; that is, there exists runtime-retrievable information that provides an explicit definition and complete data characteristics for each object and attribute supplied by the data system.

The use of MDA and *n*-tier architecture, both standard software engineering practices, allow for easy access to data, particularly by other applications.

The use of controlled vocabularies and registered metadata, less common in conventional software practices, requires specialized tools. As a result, the NCI CBIT (in cooperation with the NCI Office of Communications) developed the Enterprise Vocabulary Services (EVS) system to supply controlled vocabularies, and the Cancer Data Standards Registry (caDSR) to provide a dynamic metadata registry.

Components of caCORE

The NCI provides a range of tools and components that assist in the development of caCORE and caCORE-like systems. EVS and the caDSR provide the semantic infrastructure and the tools necessary for semantic integration. The caCORE components that support the development of systems include the caCORE Software Developers Toolkit (SDK), caAdapter, the Common Security Module (CSM) and the Common Logging Module (CLM). Each is described briefly below.

Enterprise Vocabulary Services (EVS)

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

Cancer Data Standards Repository (caDSR)

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

Software Development Kit (SDK)

The caCORE Software Development Kit or SDK is a set of development resources that allows you to create, compile, and run caCORE-like software.

The SDK can be used outside of the caCORE process to generate a system from a UML model that runs on standardized query languages, or within the caCORE process to quickly generate a caBIG silver-level compatible system from a semantically integrated UML model.

By providing a common data management framework, caCORE SDK helps streamline the informatics development throughout academic, government and private research labs and clinics.

caAdapter

The caAdapter Mapping Tool is an open source application that supports several types of data mapping and transformation.

Perhaps the most useful aspect is the ability of the tool to support object-to-data model mapping. This component allows users to parse and load data and object models from an XMI file, map the object model to the data model using drag-and-drop, add the SDK-required tags and tagged values into the XMI file, and then generate a Hibernate mapping file.

caAdapter also allows analysts and database engineers who are knowledgeable in HL7 to create a mapping from Comma Separated Value (CSV) formatted clinical data to an equivalent target HL7 v3 XML format. In addition, it allows HL7 v2 analysts to convert v2 messages into CSV format, allowing them to then be mapped to HL7 v3 format. These capabilities are provided through a front-end GUI and a back-end engine that supports specification of file formats, drag-and-drop mapping between source and target, validation of specifications and data, and transformation of actual CSV data into HL7 v3 XML message instances.

Common Security Model (CSM)

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

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

Common Logging Module (CLM)

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

CHAPTER 2

CADSR ARCHITECTURE

This chapter describes the architecture of the caDSR system.

caDSR System Architecture

The caDSR Domain Model and a view of the caDSR from a UML Project perspective are offered as services, based on the caCORE SDK architecture. The caCORE SDK exhibits an n-tiered architecture with client interfaces, server components, back-end objects, data sources, and additional back-end systems (*Figure 2.1*).

This n-tiered system divides tasks or requests among different servers and data stores. This isolates the client from the details of where and how data is retrieved. The system also performs common tasks such as logging and provides a level of security.

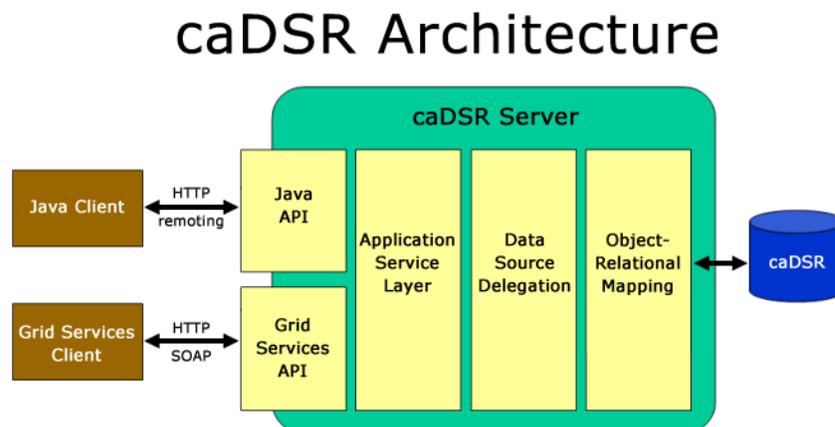


Figure 2.1 caDSR Architecture

Clients, such as browsers or applications, receive information from back-end objects. Java applications also communicate with back-end objects via domain objects

packaged within the client.jar. Non-Java applications can communicate via SOAP (Simple Object Access Protocol). Back-end objects communicate directly with caDSR data sources using Hibernate.

The caDSR services offered by the NCI CORE Infrastructure Program Area and many of the caBIG™ data services use the Java programming language and leverage reusable, third-party components.

The infrastructure is composed of the following layers:

- **The Application Service layer:** Consolidates incoming requests from the various interfaces and translates them to native query requests that are then passed to the data layers. This layer is also responsible for handling client authentication and access control using the Java API.

Note: The client authentication feature is currently disabled for the caCORE system running at CBIIT; all interfaces provide full, anonymous, read-only access to all data.
- **The Data Source Delegation layer:** Responsible for conveying each query it receives to the respective data source capable of performing the query. The presence of this layer allows multiple data sources to be exposed by a single running instance of a caCORE server.
- **Object-Relational Mapping (ORM):** Implemented using Hibernate. Hibernate is a high performance object/relational persistence and query service for Java. Hibernate provides the ability to develop persistent classes following common object-oriented design methodologies such as association, inheritance, polymorphism, and composition.

The *Hibernate Query Language (HQL)* was designed as a “minimal” object-oriented extension to SQL and provides a bridge between object and relational databases. Hibernate allows for real-world modeling of biological entities without having to create complete SQL-based queries to represent them.

Client Technologies

Applications using the Java programming language can access caDSR directly through the domain objects provided by the client.zip bundle (see [Chapter 4, Interacting With caDSR](#), on page 41). The network details of the communication to the caDSR server are abstracted away from the developer. Hence developers need not deal with issues such as network and database communication, but can instead concentrate on the biological problem domain.

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

caDSR Packages

Table 2.1 below shows the caDSR packages from which you can access the Java interfaces and classes. All of the objects in the domain package implement the `java.io.Serializable` interface.

To view the JavaDocs page for each package, go to <http://cadsrapi.nci.nih.gov/cadsrapi40/docs/>.

Packages	Description
gov.nih.nci.cadsr.domain	Contains the domain classes in the caDSR ISO 11179 related objects such as DataElement, ValueDomain and AdministeredComponent. For a list of caDSR domain objects, see Chapter 3, Understanding caDSR and the caDSR API , on page 13.
gov.nih.nci.cadsr.umlproject	Contains the convenience classes for the UML Project related data in the caDSR.

Table 2.1 caDSR Packages and Descriptions

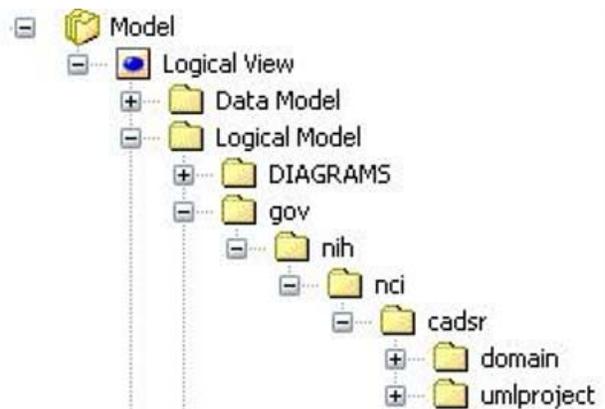


Figure 2.2 caDSR Packages

In addition to domain packages, the caDSR API client.zip includes the following framework packages:

System —The system package has subpackages including the application service package, data access package, delegate/service locator package, proxy package, and web service package.

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

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

CHAPTER 3

UNDERSTANDING CADSR AND THE CADSR API

The Cancer Data Standards Registry (caDSR) at NCI is part of a larger effort associated with the 11179 standard defined by the ISO (International Organization for Standardization) and IEC (International Electrotechnical Commission). The purpose of the [ISO/IEC 11179](#) is to standardize the metadata used in representing and annotating shared electronic data. The caDSR is a conforming implementation of ISO/IEC 11179 Edition 2 with extensions, though some of the attributes in the ISO 11179 pertaining to Stewardship/Submission and Registrar are not exposed in the user interfaces.

This page describes the caDSR, the details of the metadata it contains, and the caDSR application programming interface (API). The first sections provide a brief review of the ISO/IEC 11179 standard and its realization as an Oracle database at NCI. Later sections provide information regarding the Java API to the repository and introduce the Java classes that participate in this programmatic interface.

Modeling Metadata: The ISO/IEC 11179 Standard

Regardless of the application domain, any particular data item must have associated with it a variable name or tag, a conceptualization of what the item signifies, a value, and an intended interpretation of that value.

For example, an entry on a case report form is intended to capture the patient's place of birth, with the corresponding value tagged electronically as *Patient_placeOfBirth*. But what is the intended concept? Is the data element designed to capture the country, the city, or the specific hospital where the person was born? Assuming that the intended concept is country, how is the resulting value to be represented electronically? Possible representations could include the full name of the country, a standard two- or three-letter abbreviation, a standard country code, or perhaps a specific encoding unique to the application.

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

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

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

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

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

Note: The term *object* is used here in the sense defined by the ISO/IEC 11179 and does not have any literal correspondence to a caCORE Java object.

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

- A *DataElementConcept* consists of an *object* and a selected *property* of that object;
- The *ConceptualDomain* is the set of all intended meanings for the possible values of an associated *DataElementConcept*;
- The *ValueDomain* is a set of accepted representations for these intended meanings; and
- A *DataElement* is a combination of a selected *DataElementConcept* and a *ValueDomain*.

The example below diagrams these definitions.

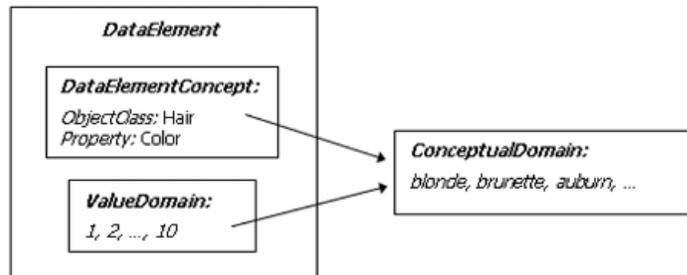


Figure 3.1 Representing data in the ISO/IEC 11179 model

Figure 3.1 above shows how a *DataElement* that might be used to represent hair color. The associated *DataElementConcept* uses the *ObjectClass* “Hair” and the *Property* “Color” to define the intended concept. The intended meanings for this data element are the familiar hair colors blonde, brunette, etc., but the *ValueDomain* uses a numeric representation that is mapped to these intended meanings. Both the *DataElementConcept* and the *ValueDomain* are components of the *DataElement*, and each references the same *ConceptualDomain*, which is defined outside the *DataElement*.

Important principles of this design are:

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

Figure 3.2 uses a UML Class diagram to show the cardinality constraints that hold for these relations.

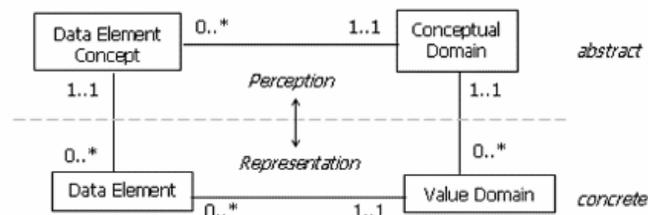


Figure 3.2 Abstract and concrete components of the data representation

Each *DataElement* must specify exactly one *DataElementConcept* and one *ValueDomain*, in order to fully specify the data element. Similarly, each *DataElementConcept* and *ValueDomain* must specify exactly one *ConceptualDomain*. Conversely, a *ConceptualDomain* may be associated with any number of *ValueDomains* and any number of *DataElementConcepts*.

Figure 3.3 shows an example of this, using the color property of different geometric objects as *DataElementConcepts*, and alternate color representations for the *ValueDomains*.

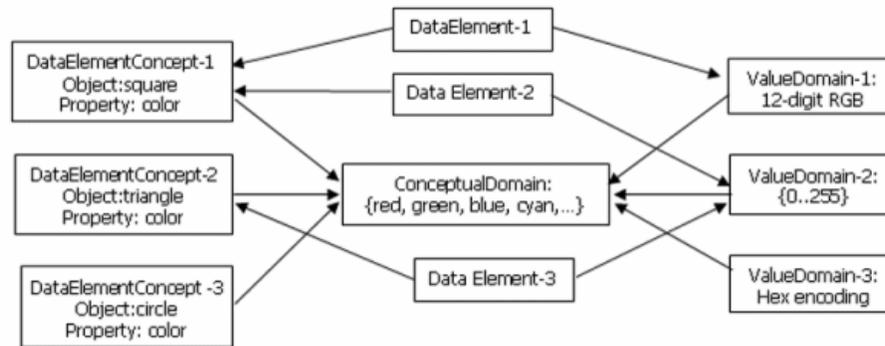


Figure 3.3 Many-to-one mappings of information elements in the metadata model

A constraint not shown in any of these figures is that it is not possible to reuse the same *DataElementConcept-ValueDomain* pair to define a new *DataElement*, as this defines a logical redundancy. Thus, the “0..*” cardinality constraints implied by Figure 3.2 are not quite as open-ended as they imply.

Specifically defined, the cardinality constraints are as follows:

- A *DataElement* specifies exactly one *DataElementConcept* and one *ValueDomain*;
- A *DataElementConcept* specifies exactly one *ConceptualDomain*;
- A *ValueDomain* specifies exactly one *ConceptualDomain*;
- A *ConceptualDomain* may be associated with any number of *ValueDomains*;
- A *ConceptualDomain* may be associated with any number of *DataElementConcepts*;
- A *DataElementConcept* may be associated with as many *DataElements* as there are *ValueDomains* (i.e. alternate representations) associated with the *ConceptualDomain*; and
- A *ValueDomain* may be associated with as many *DataElements* as there are *DataElementConcepts* associated with the *ConceptualDomain*.

Many additional information components collaborate with these four core elements to provide the ISO/IEC 11179 infrastructure for content representation. These are described in the caDSR model in the next section, along with the organizational and administrative components that are used to document, classify, and in general, manage the information components.

Component Name	Definition	Administered Component
<i>DataElementConcept</i>	A subclass of an Administered Component that depicts a characteristic of something in the real world that can be represented in the form of a data element, independent of any particular representation.	Yes
<i>ObjectClassRelationship</i>	An affiliation between two instances of ObjectClass.	Yes
<i>EnumeratedValueDomain</i>	A subtype of value domain expressed as a list of all permissible values.	Yes
<i>NonenumeratedValueDomain</i>	A subtype of value domain expressed by a generative rule or formula; for example: "all even integers less than 100."	Yes
<i>ObjectClass</i>	A subclass of an Administered Component that depicts a set of ideas, abstractions, or things in the real world that can be identified with explicit boundaries and meaning and whose properties and behavior follow the same rules.	Yes
<i>PermissibleValue</i>	The exact names, codes, and text that can be stored in a data field in an information management system.	No
<i>Property</i>	A subclass of an Administered Component that depicts a characteristic common to all members of an Object Class. It may be any feature naturally used to distinguish one individual object from another. It is conceptual and thus has no particular associated means of representation.	Yes
<i>Qualifier</i>	A term that helps define and render a concept unique. For example, given the ObjectClass household and the Property annual income, a Qualifier for the Property could be used to indicate previous year. One or more Qualifiers can be present for ObjectClass or Property.	No
<i>Representation</i>	A subclass of Administered Component that depicts a mechanism by which the functional and/or presentational category of an item may be conveyed to a user. Examples: 2-digit country code, currency, YYYY-MM-DD, etc.	Yes
<i>ValueDomain</i>	A subclass of an Administered Component that describes the attributes of a set of permissible values for a data element.	Yes
<i>ValueDomainPermissibleValue</i>	The many-to-many relationship between value domains and permissible values; allows one to associate a value domain to a permissible value.	Yes
<i>ValueMeaning</i>	The significance or intended meaning of a permissible value.	Yes

Table 3.1 Information Classes in the caDSR Metamodel

An *AdministeredComponent* is a component for which administrative information must be recorded. It may be a *DataElement* itself or one of its associated components (*Representation*, *ValueDomain*, *DataElementConcept*, *ConceptualDomain*, *ObjectClass*, or *Property*). Regardless, every administered component requires explicit specifications for reuse in or among enterprises. The term *AdministeredComponent* is a generalization for all of the descendant components highlighted in [Figure 3.4](#).

[Table 3.2](#) below lists the class attributes of the Administration Record common to all *AdministeredComponents*.

Attribute Name	Type	Required/ Options	caDSR UI Name
id (Public ID)	String	required	Public ID
shortName (NCI Extension)	String	required	Short Name
preferredDefinition	String	required	Definition
longName (see NOTE below)	String	required	Long Name
version	Float	required	Version
workflowStatusName	String	required	Workflow Status
workflowStatusDescription	String	required	n/a
latestVersionIndicator	Boolean	required	Latest Version
beginDate	Date	required	Effective Begin Date
endDate	Date	required	Effective End Date
deletedIndicator	Boolean	optional	n/a
changeNote	String	optional	
unresolvedIssue	String	optional	
origin	String	optional	Origin or Database
dateCreated	Date	required	
dateModified	Date	required	
registration (Registration Status)	String	optional	Registration Status

Table 3.2 Class Attributes of an AdministeredComponent

Note: The *longName* identified in caDSR for a component equates to the ISO/IEC 11179 *PreferredName*.

The class attributes listed in [Table 3.2](#) tell only part of the story however. Additional critical information about each *AdministeredComponent* is derived from its associations with the organizational and administrative items depicted in [Figure 3.5](#) below. Of these components, the only item that is also itself an Administered Component is the *ClassificationScheme*.

[Figure 3.5](#) outlines three “regions”: (1) the Naming and Identification region (upper right), (2) the Classification region (lower left), and (3) the Contact Information section

(bottom center). The *ReferenceDocument* component (right side) is not included in any region. This is because each *AdministeredComponent* may be associated with one or more *ReferenceDocuments* that identify where and when the component was created and provide contact information for the component's designated registration authority.

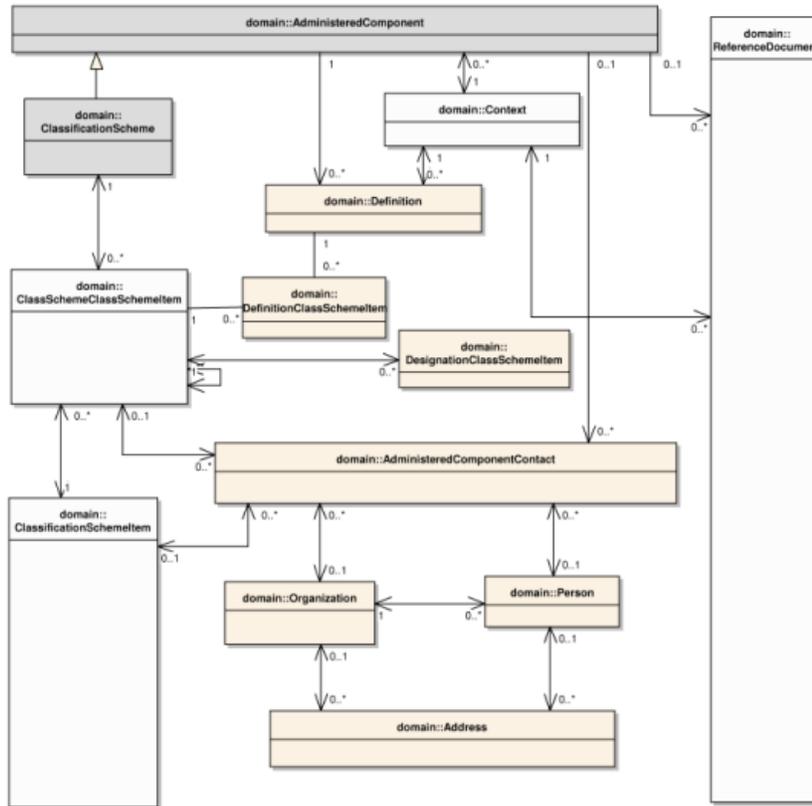


Figure 3.5 Administrative and organizational components of the caDSR metamodel

The purpose of the Naming and Identification region is to manage the various names by which components are referenced in different contexts. Many components may be referenced by different names depending on the discipline, locality, and technology in which they are used. In addition to the name and attributes contained in the component itself (*shortName*, *longName*), an administered component may have any number of alternative *Designations*. Each *Designation* is associated with exactly one *Context* reflecting its usage, and may be classified by one or more Classification Scheme Items.

The Classification region is used to manage classification schemes and the administered components that are in those classification schemes. Classification is a fundamental and powerful way of organizing information to make the contents more accessible. Abstractly, a classification *scheme* is any set of organizing principles or dimensions along which data can be organized. In the ISO/IEC 11179 model, a *ClassificationScheme* may be something as simple as a collection of keywords or as complex as an ontology. The classification scheme element in [Figure 3.5](#) is highlighted in light gray to reflect that it is an administered component.

Classification schemes that define associations among components can greatly assist navigation through a large network of elements; the associations may describe simple subsumption hierarchies or more complex relations such as causal or temporal

relations. In particular, classification schemes with inheritance can enhance self-contained definitions by contributing the definition of one or more ancestors.

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

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

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

The Contact region of [Figure 3.5](#) consists of the *Organization*, *Person*, and *Address* classes. A *Person* or an *Organization* can be the contact for an *AdministeredComponent* and can be reached at an *Address*. In particular:

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

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

components exercise the representational power of the metamodel, and are used to specify how clinical trials data should be captured and exchanged.

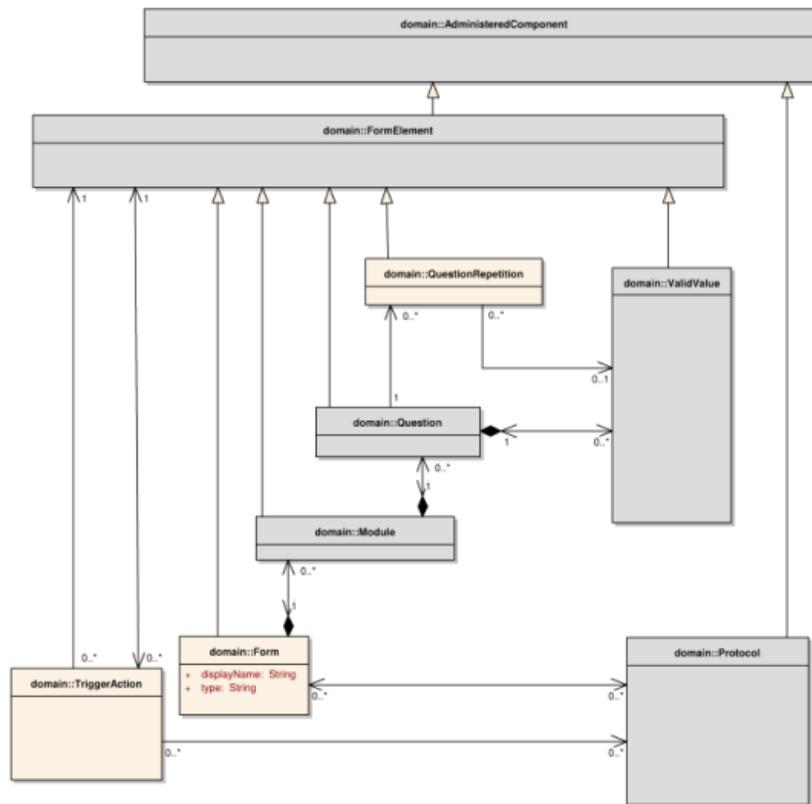


Figure 3.6 Components in the caDSR metamodel for clinical trials data

All of the components in [Figure 3.6](#) are highlighted, as they are *AdministeredComponents* designed for use in NCI-sponsored clinical trials. Note that because these elements are not part of the ISO-11179 specification, they are not, technically speaking, ISO administered components. This caDSR design decision was made to ensure that these shared data elements could be stewarded and controlled adequately.

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

- [The Cancer Therapy Evaluation Project](#) (CTEP)
- [Specialized Programs of Research Excellence](#) (SPORES)
- [The Early Detection Research Network](#) (EDRN)
- [The Division of Cancer Prevention](#) (DCP)
- [The Cancer Imaging Program](#) (CIP)
- [The Division of Cancer Epidemiology and Genetics](#) (DECG)
- [The Cancer Bioinformatics Infrastructure Objects Project](#) (caBIO)

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

Table 3.3 caDSR Form component names and descriptions

caDSR API

The previous section described three broad categories of components in the caDSR metamodel and presented each of these independently, thus implying that there are no dependencies among these groupings. [Figure 3.7](#) below brings these components together and exposes the associations that actually occur between the components in different categories.

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

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

The three categories of components are also outlined in [Figure 3.7](#) below: administrative and organizational components in the upper left, forms components in the upper right, and information components centered underneath these two.

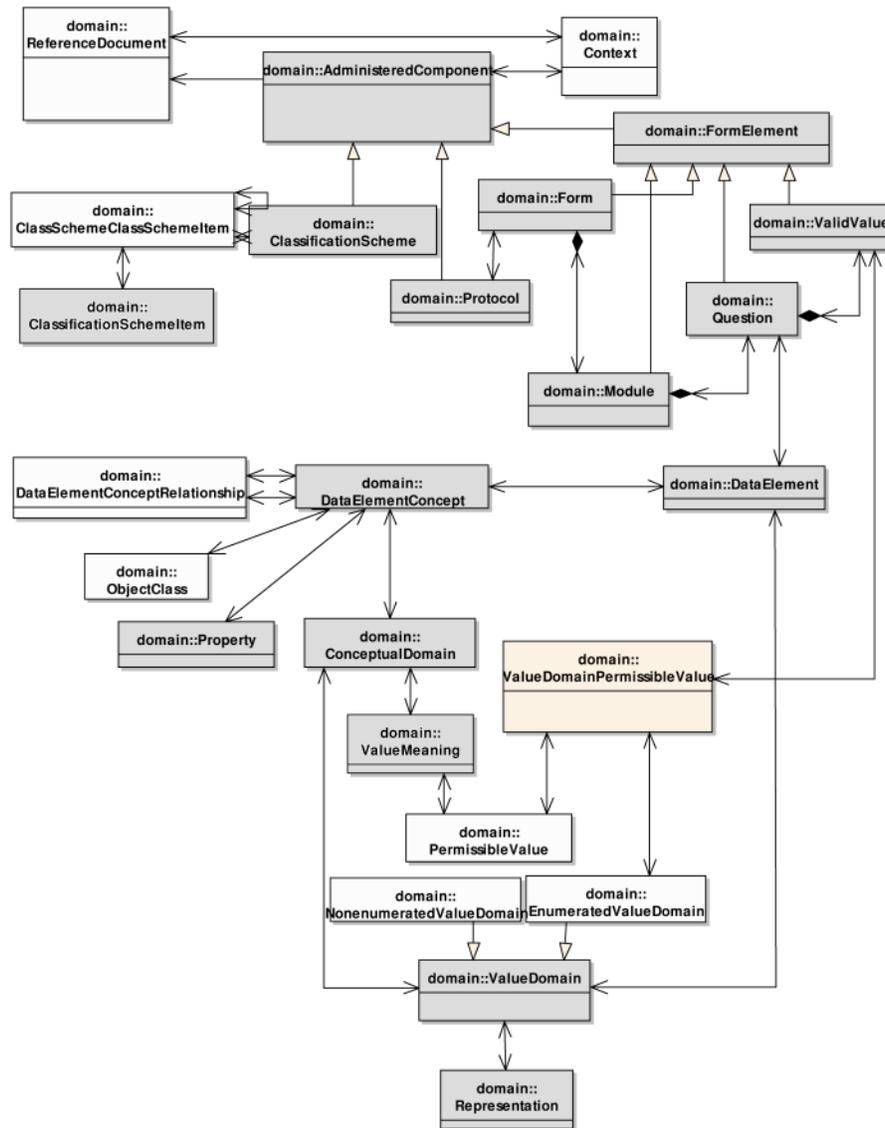


Figure 3.7 caDSR domain objects in the caCORE Java API

[Figure 3.8](#) below summarizes the caDSR API class hierarchy. At the most abstract level, a single distinguished object called the *DomainObject* class is the ancestor to all other classes. At the next level, the *AdministeredComponent* class is defined, along with all of the other classes that do not represent elements requiring administration.

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

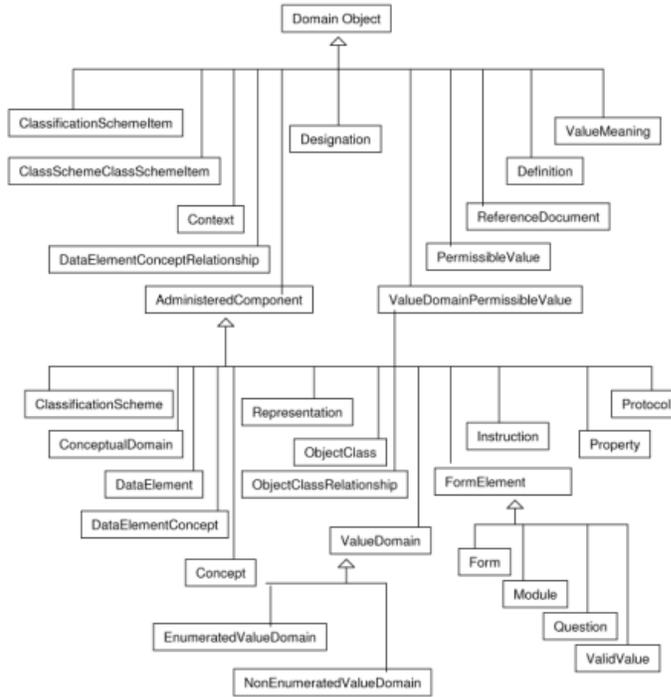


Figure 3.8 caDSR API class hierarchy

caDSR Domain Object Catalog

The previous discussion was intended to provide a descriptive overview of the domain objects included in the caDSR API.

Table 3.4 below lists each class along with a description. Detailed descriptions about each class and its methods are present in the caCORE 4.0 JavaDocs located at: <http://ncicb.nci.nih.gov/core/caDSR/API/4.0>.

caDSR Domain Object	Description
Address	A physical location at which Persons or Organizations can be contacted.
AdministeredComponent	A class for which attributes (or characteristics) are collected; Data Elements are one type of administered component. Other administered components have relationships to data elements as well as each other.
AdministeredComponentClassSchemeItem	A class that serves to allow many to many relationships between Administered Component (AC) and ClassSchemeClassSchemeItem (CS/CSI), providing uniqueness to the CS/CSI pairing to an AC.
AdministeredComponentContact	A relationship between an Administered Component and contact information (e.g. Address).

Table 3.4 caDSR Domain Objects and Descriptions

caDSR Domain Object	Description
CaseReportForm	A questionnaire that documents all the patient data stipulated in the protocol and used by clinicians to record information about patient's visits while on the clinical trial.
ClassificationScheme	A class that serves to describe an arrangement or division of objects into groups based on characteristics that the objects have in common, e.g., origin, composition, structure, application, function, etc. Adds information not easily included in definitions, helps organize the registry and facilitates access to the registry. ISO DEF: the descriptive information for an arrangement or division of objects into groups based on characteristics, which the objects have in common.
ClassificationSchemeItem	A component of content in a Classification Scheme. This may be a node in a taxonomy or ontology or a term in a thesaurus, etc.
ClassificationSchemeItemRelationship	The affiliation between two occurrences of Classification Scheme Items.
ClassificationSchemeRelationship	The affiliation between two occurrences of Classification Schemes.
ClassSchemeClassSchemeItem	Information pertaining to the association between Classification Schemes and Classification Scheme Items. This information is used to get all Classification Scheme Items that belong to a particular Classification Scheme as well as the information about it
ComponentConcept	The concept component(s) used for a concept derivation
ComponentLevel	Level of the component of the derivation rule
Concept	A class that serves to describe an administered component
ConceptDerivationRule	The derivation rule between one or more concepts.
ConceptualDomain	The set of all possible Valid Value meanings of a Data Element Concept expressed without representation.
Context	A designation or description of the application environment or discipline in which a name is applied or from which it originates.
DataElement	A class that serves to describe a unit of data for which the definition, identification, representation and permissible values are derived from its association with one DataElementConcept and one ValueDomain.
DataElementConcept	A class that serves to describe a concept that can be represented in the form of a data element, described independently of any particular representation.
DataElementConceptRelationship	A description of the affiliation between two occurrences of Data Element Concepts.
DataElementDerivation	The data element component(s) used for a derived data element.

Table 3.4 caDSR Domain Objects and Descriptions

caDSR Domain Object	Description
DataElementRelationship	The affiliation between two occurrences of Data Elements.
Definition	A definition for an Administered Component in a specific Context.
DefinitionClassSchemeItem	A class that serves to allow many to many relationships between Definitions and ClassSchemeClassSchemeItem, providing uniqueness to the CS/CSI pairing to a definition. <i>Please be advised this class will be removed during a future release</i>
DerivationType	The type of Derived Data Element that is being created. For example a Data Element that is derived/created by subtracting two dates represented by other data elements would be a Calculated Representation Type. Types include: Calculated, Complex Recode, Compound, Concatenation, Object Class, and Simple Recode.
DerivedDataElement	The Data Element that is derived from one or more data elements. ISO DEF: the relationship among a Data Element which is derived, the rule controlling its derivation, and the Data Element(s) from which it is derived.
Designation	A name by which an Administered Component is known in a specific Context. Also a placeholder to track the usage of Administered Components by different Contexts.
DesignationClassSchemeItem	A class that serves to allow many to many relationships between Designation and ClassSchemeClassSchemeItem, providing uniqueness to the CS/CSI pairing to an Designation.
EnumeratedValueDomain	A ValueDomain with an associated set of discrete PermissibleValues; the alternative to a NonenumeratedValueDomain.
Form	A questionnaire that documents all the patient data stipulated in the protocol and used by clinicians to record information about patient's visits while on the clinical trial.
FormElement	An element on a Case Report Form. Examples: The form itself, groups of questions (modules), questions, valid values.
Function	Function to be applied to the relationship.
Instruction	Instruction for a Form, Module, Question or Valid Value on a Form.
Module	A collection of data elements, or Common Data Elements, logically grouped on a case report form.
NonenumeratedValueDomain	A value domain not expressed as a list of all permissible values.
ObjectClass	A class that serves to describe a set of ideas, abstractions, or things in the real world that can be identified with explicit boundaries and meaning and whose properties and behavior follow the same rules.

Table 3.4 caDSR Domain Objects and Descriptions

caDSR Domain Object	Description
ObjectClassRelationship	A class that provides a description of the affiliation between two occurrences of Object Classes
Organization	Information about an Organizational unit like a laboratory, institute or consortium
PermissibleValue	An enumerated list of the exact names, codes and text that can be stored in a data field in an information management system. ISO DEF: An expression of a value meaning in a specific value domain.
Person	Information about a contact person.
Property	A class that serves to describe a characteristic common to all members of an Object Class. It may be any feature that humans naturally use to distinguish one individual object from another. It is conceptual and thus has no particular associated means of representation by which property.
Protocol	A class that serves to provide identification of a Clinical Trial Protocol document and its collection of Case Report Forms (CRFs). Note: Protocols will be uniquely identified within each of the 3 areas of caCORE - caBIO, SPORES and caDSR- using the following three attributes: Protocol ID, Protocol Version and Context Name. This class will serve as a 'hook' across the three caCORE domains allowing a user to navigate across databases.
ProtocolFormsSet	Identification of a Clinical Trial Protocol document and its collection of Case Report Forms (CRFs). Note: Protocols will be uniquely identified within each of the 3 areas of caCORE - caBIO, SPORES and caDSR- using the following three attributes: Protocol ID, Protocol Version and Context Name. This class will serve as a 'hook' across the three caCORE domains allowing a user to navigate across databases.
ProtocolFormsTemplate	The collection of components (modules, questions and valid values) comprising a template Case Report Form. A template form is not associated with any particular clinical trial.
Question	The actual text of the data element as specified on a Case Report Form of a Protocol.
QuestionRepetition	Information about the default valid values every time the question repeats on a form.
ReferenceDocument	A place to document additional information about Administered Components that is not readily stored elsewhere.

Table 3.4 caDSR Domain Objects and Descriptions

caDSR Domain Object	Description
Representation	<p>A class that serves to describe the mechanism by which the functional and/or presentational category of an item maybe conveyed to a user. Component of a Data Element Name that describes how data are represented (i.e. the combination of a Value Domain, data type, and if necessary a unit of measure or a character set). The Representation occupies the last position in the Data Element name (e.g., right most).</p> <p>Examples: Code - A system of valid symbols that substitute for specified values (e.g. alpha, numeric, symbols and/or combinations). Count: Non-monetary numeric value arrived at by counting. Currency: Monetary representation. Date: Calendar representation (e.g. YYYY-MM-DD). Graphic: Diagrams, graphs, mathematical curves, or the like. Image: Usually a vector image. Icon: A sign or representation that stands for its object by virtue of a resemblance or analogy to it. Picture: A visual representation of a person, object, or scene; usually a raster image. Quantity: A continuous number such as the linear dimensions, capacity/amount (non-monetary) of an object. Text: A text field that is usually unformatted. Time: Time of day or duration (e.g. HH:MM:SS.SSSS).</p>
TriggerAction	A conditional branching between to FormElements triggered by a certain response to a Question.
ValidValue	The allowable values for a Question on a Form.
ValueDomain	A class that serves to describe the attributes for a set of permissible values for a data element.
ValueDomainPermissibleValue	This captures the many-to-many relationship between value domain and permissible values and allows to associate a value domain to a permissible value.
ValueDomainRelationship	The affiliation between two occurrences of Value Domains.
ValueMeaning	The significance associated with an allowable/permissible value.

Table 3.4 caDSR Domain Objects and Descriptions

The *Concept* object encapsulates a concept in a vocabulary served by EVS. It is modeled as a new administered component type. [Table 3.5](#) contains a brief description of some concept object attributes.

Concept Object Attribute	Description
Concept Attribute	Data
Short Name	Contains the immutable concept code of an EVS concept.
Long Name	Contains the preferred name of an EVS concept
Preferred Definition	Contains the definition of an EVS concept.
Definition Source	Contains a value provided via the EVS API indicating the source of the EVS preferred definition.

Table 3.5 Concept Object Attributes

Concept Object Attribute	Description
Database	Contains the name of the EVS resource for the immutable identifier.
Source Type	The type of concept code in the Short Name, e.g., NCI Concept Code, NCI Metathesaurus, etc.

Table 3.5 *Concept Object Attributes*

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

Each *ConceptDerivationRule* object is also associated with one *ConceptDerivationType* object which encapsulates a type of concept derivation rule. One *ComponentConcept* is designated as primary, the others are ordered, with the highest number taking the left most position in the concept expression, modifying the concept immediately to its right. For example, in *Begin Date*, “Begin” modifies “Date.” “Date” is the primary *ComponentConcept* with “Begin” as concept order number 1.

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

As illustrated in [Figure 3.9](#) below, the following administered components types are associated with the *ConceptDerivationRule* object:

- ObjectClass
- Property
- Representation
- ValueDomain
- ConceptualDomain

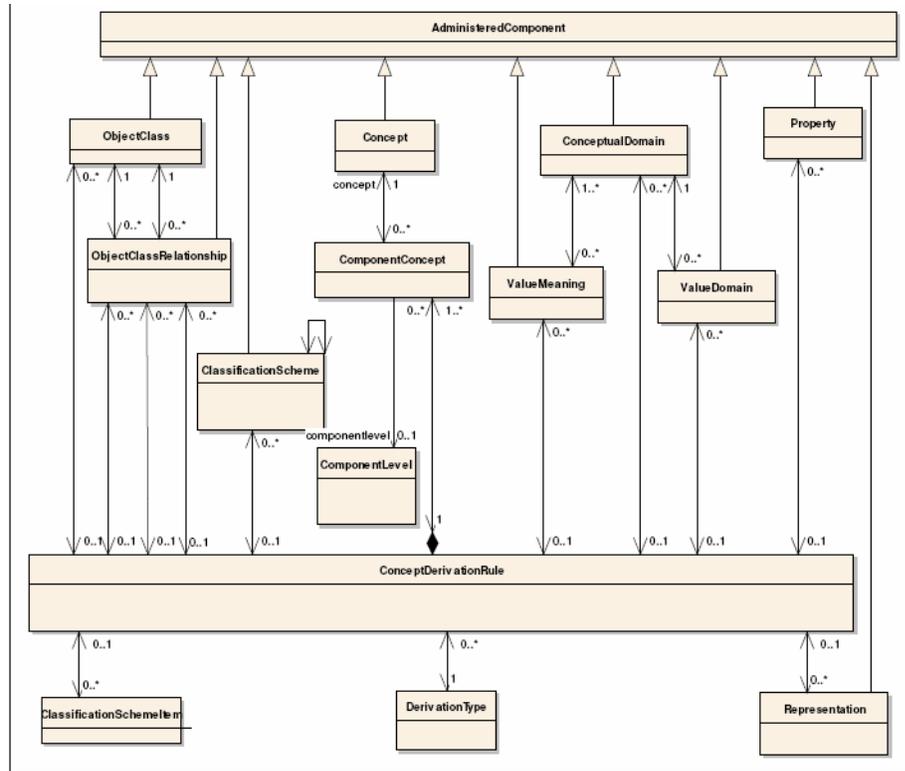


Figure 3.9 Extensions to the caDSR model

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

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

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

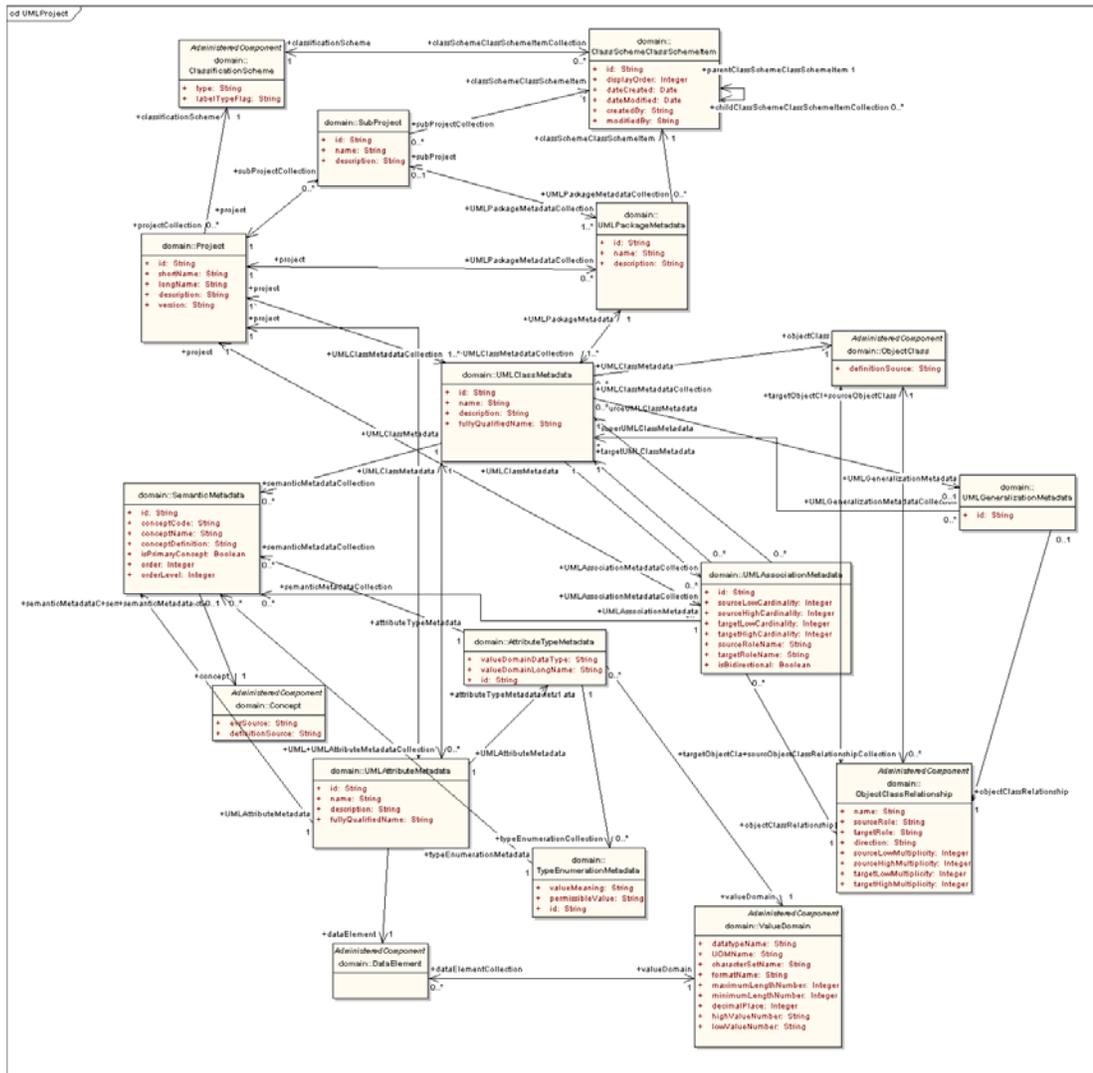


Figure 3.10 caDSR UML Project Diagram

For a full size view of the above UML Project Diagram, click here: [caDSR UML Project Diagram](#) (internet connection required).

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

Table 3.6 caDSR UML Project Domain Objects

Downloading the caDSR

The following caDSR distributions can be downloaded from the NCICB Download site located at: <http://ncicb.nci.nih.gov/download/>.

caDSR Tool	Description
CDE Curation Tool Distribution	Contains the cdecurate.war file and the CDE Curation Tool Installation Guide.
CDE Browser Distribution	Contains cdebrowser.ear file and installation instructions for the application. CDE Browser makes Java Database Connectivity (JDBC) connections to caDSR repository database so it is a prerequisite to have access to caDSR repository for installing CDE Browser.

Table 3.7 caDSR Tools Available for Download

caDSR Tool	Description
caDSR Sentinel Tool	Contains source, third party libraries, configuration files, documentation and installation instructions for the application. Access to a caDSR repository is required.
caDSR Repository/ Administration Tool Distribution	Contains the caDSR Installation Guide and scripts for both the caDSR repository and Administration Tool.
caDSR Repository/ Administration Tool Source Code Distribution	Contains the complete PL/SQL scripts for both the repository and the Administration Tool.
UML Model Browser Distribution	Contains umlmodelbrowser.ear file and installation instructions for the application.

Table 3.7 caDSR Tools Available for Download

Follow the instructions provided with the distributions to install the software. caDSR content is not downloaded via this process. caDSR data element content may be downloaded by using the CDE Browser available online at: <https://cdebrowser.nci.nih.gov>.

caDSR API Examples

Using the caDSR Java API

Example 3.1 Querying the latest version of a DataElement

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

```
import gov.nih.nci.cadsr.domain.DataElement;
import gov.nih.nci.cadsr.domain.DataElementConcept;
import gov.nih.nci.cadsr.domain.EnumeratedValueDomain;
import gov.nih.nci.cadsr.domain.PermissibleValue;
import gov.nih.nci.cadsr.domain.ValueDomain;
import gov.nih.nci.cadsr.domain.ValueDomainPermissibleValue;
import gov.nih.nci.system.applicationservice.ApplicationService;
import gov.nih.nci.system.client.ApplicationServiceProvider;
import java.util.Collection;
import java.util.List;

/**
 * A sample use of the caDSR api.
 * @author caDSR team.
 */
public class TestCaDsrApi
{
    /**
     * Search for Data Elements using the Long Name
     *
     * @param args
     */
    public static void main(String[] args)
    {
```

```

try
{
    ApplicationService appService =
ApplicationServiceProvider.getApplicationService("CaDsrServiceInfo"
);
    System.out.println("Searching for DataElements");

    // Search for the Data Element with the Long Name "Patient
Race Category*". The asterisk (*) is a wild card.
    DataElement dataElement = new DataElement();
    dataElement.setLongName("Patient Race Category*");
    dataElement.setLatestVersionIndicator("Yes");
    List<Object> results =
appService.search(DataElement.class, dataElement);

    for (Object obj: results)
    {
        // Show the DE and its DEC and VD
        DataElement de = (DataElement) obj;
        System.out.println("Data Element " +
de.getLongName());

        DataElementConcept dec = de.getDataElementConcept();
        System.out.println("Data Element Concept " +
dec.getLongName());

        ValueDomain vd = de.getValueDomain();
        System.out.println("Value Domain " +
vd.getLongName());

        if (vd instanceof EnumeratedValueDomain)
        {
            // Get the PermissibleValues for the ValueDomain
            EnumeratedValueDomain evd =
(EnumeratedValueDomain) vd;
            Collection<ValueDomainPermissibleValue> vdpvs =
evd.getValueDomainPermissibleValueCollection();
            for (ValueDomainPermissibleValue vdpv: vdpvs)
            {
                PermissibleValue pv =
vdpv.getPermissibleValue();
                System.out.println(" Permissible Value : " +
pv.getValue());
            }
        }
    }
}
catch (Exception exception)
{
    exception.printStackTrace();
    System.out.println("Error in the TestCaDsrApi");
}
}

```

Using the caDSR Web Services API

Example 3.2 Using datatypes generated from Apache Axis

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

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

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

    public static void main(String[] args)
    {
        try
        {
            // Get a caCORE Service instance.
            WSQuery wsQuery = new
                WSQueryServiceLocator().getcaCOREService();
            System.out.println("Query a DataElement");

            // Set the search criteria. * is used as a wild card.
            DataElement dataElement = new DataElement();
            dataElement.setLongName("Patient Race Category*");
            dataElement.setLatestVersionIndicator("Yes");
            System.out.println("Searching for data elements");
            Object[] results =
                wsQuery.queryObject("gov.nih.nci.cadsr.domain.ws.DataElement",
                    dataElement);
            for (int i = 0; i < results.length; i++)
            {
                DataElement dataElementQ = (DataElement) results[i];
                System.out.println("Queried DataElement " +
                    dataElementQ.getLongName());
                // Query DataElementConcept
                DataElement de = new DataElement();
                de.setId(dataElementQ.getId());
                DataElementConcept dec = (DataElementConcept)
                    wsQuery.queryObject("gov.nih.nci.cadsr.domain.ws.Data
                        ElementConcept", de)[0];
                System.out.println("Queried DataElementConcept " +
                    dec.getLongName());
                // Query ValueDomain
            }
        }
    }
}
```

```

        ValueDomain vd = (ValueDomain)
wsQuery.queryObject("gov.nih.nci.cadsr.domain.ws.ValueDomain ",
de)[0];
        System.out.println("Queried ValueDomain " +
vd.getLongName());
        if (vd instanceof EnumeratedValueDomain)
        {
            // Query Permissible Values
            EnumeratedValueDomain evd = new
EnumeratedValueDomain();
            evd.setId(vd.getId());
            Object[] valueDomainPermissibleValues =
wsQuery.queryObject("gov.nih.nci.cadsr.domain.ws.ValueDomainPermiss
ibleValue", evd);
            for (int j = 0; j <
valueDomainPermissibleValues.length; j++)
            {
                ValueDomainPermissibleValue vdpv =
(ValueDomainPermissibleValue) valueDomainPermissibleValues[j];
                ValueDomainPermissibleValue vdpv2 = new
ValueDomainPermissibleValue();
                vdpv2.setId(vdpv.getId());
                PermissibleValue pv = (PermissibleValue)
wsQuery.queryObject("gov.nih.nci.cadsr.domain.ws.Permis
sibleValue", vdpv2)[0];
                System.out.println("Queried permissible value
" + pv.getValue());
            }
        }
    }
}
catch (Exception exception)
{
    exception.printStackTrace();
    System.out.println("Error testing web service api.");
}
}
}

```

UML Project API Examples

Example 3.3 Using the caCORE client Java API

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

```

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

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

import java.util.Iterator;

```

```
import java.util.List;

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

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

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

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

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

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

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

```

        .getContext().getName());
    }

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

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

umlClass.getUMLAttributeMetadataCollection().iterator();
            resultsIterator.hasNext(); ) {
            UMLAttributeMetadata umlAttribute =
            (UMLAttributeMetadata)resultsIterator.next();
            printAttributeInfo(umlAttribute);
        }
    }

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

```

```
        } catch (Exception e) {
            e.printStackTrace();
        }
    } catch (RuntimeException e2) {
        e2.printStackTrace();
    }
}

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

CHAPTER 4

INTERACTING WITH caDSR

Interacting with the caDSR database is done via the caDSR API, a caCORE SDK generated interface. The caCORE architecture includes a service layer that provides a single, common access paradigm to clients using any of the provided interfaces. As an object-oriented middleware layer designed for flexible data access, caCORE-based systems rely heavily on strongly-typed objects and an object-in/object-out mechanism.

The most common use of a caCORE-based system, including the caDSR, is referred to as query-by-example (QBE), meaning that the inputs to the query methods are themselves domain objects that provide example data for the query criteria. This technique does not require knowledge of a specific query language, for more information refer to the general technique on http://en.wikipedia.org/wiki/Query_by_Example. Other query methods and techniques are also provided in the API when QBE is insufficient.

The basic order of operations required to access and use the caDSR follows:

1. Ensure that the client application has knowledge of the objects in the domain space.
2. Formulate the query criteria using the domain objects.
3. Establish a connection to the server.
4. Submit the query objects and specify the desired class of objects to be returned.

This page describes the service interface layer provided by the caDSR architecture, identifies installation and configuration requirements, and provides examples for using the APIs.

Java API

This API provides the fullest set of features and capabilities because the caDSR API is coded in Java.

Installation and Configuration

Accessing the caDSR system also requires an Internet connection. To use the Java API, download the client package provided on the NCICB website.

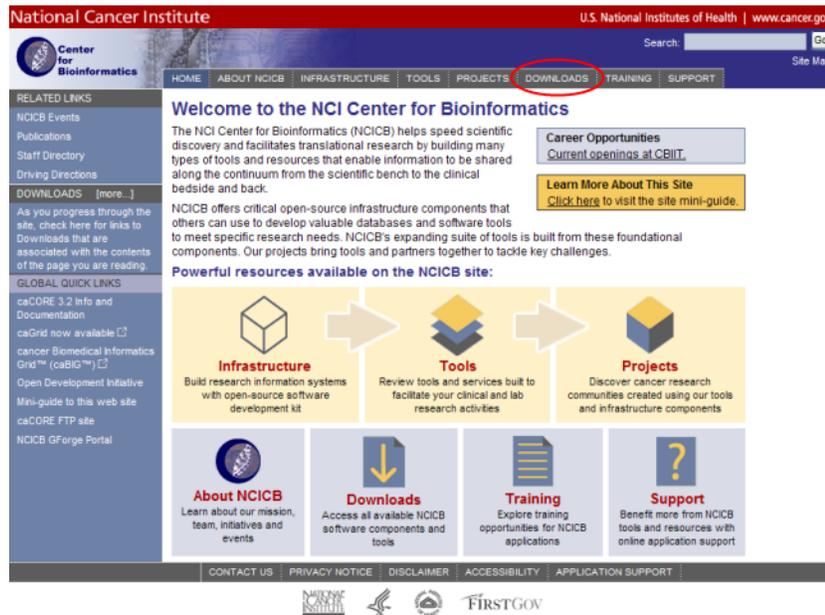


Figure 4.1 NCICB Website - Downloads Link Identified

To download the caDSR API:

1. Open your browser and go to the NCICB caDSR Download website: <http://ncicb.nci.nih.gov/download/>.
2. In the provided form, enter your name, email address and institution name and click to Enter the Download Area.
3. Accept the license agreement (<http://ncicb.nci.nih.gov/download/cacore/cadsrapilicenseagreement.jsp>).
4. Download the caDSR API Client Zip file (called the client bundle) from the Primary Distribution section.
5. Extract the contents of the downloadable archive to a directory on your hard drive (for example, **c:\cadsrapi40** on Windows or **/usr/local/cadsrapi40** on Linux).

Table 4.1 lists the extracted directories and files.

Directories and Files	Description	Component
application-config-client.xml	An alternate XML file which must be used to replace the remote-client/conf version when an application wished to use more than one caCORE SDK generated bundle, e.g. EVS and caDSR	
remote-client/build.xml	Build script for Local Client	http://ant.apache.org/

Table 4.1 Extracted Directories and Files in caDSR Client Package

Directories and Files	Description	Component
remote-client/conf/application-config-client.xml	Default application configuration which defines the Service Info and Beans	
remote-client/conf/gov.nih.nci.cadsr.domain.xsd	The caDSR Domain objects schema	
remote-client/conf/gov.nih.nci.cadsr.umlproject.domain.xsd	The caDSR UML Project objects schema	
remote-client/conf/log4j.properties	The logging (log4j) properties	http://logging.apache.org/log4j/
remote-client/conf/mapping.dtd	The Object to Database mappings	
remote-client/conf/unmarshaller-xml-mapping.xml	Unmarshalling the XML	
remote-client/conf/xml-mapping.xml	XML Mappings	
remote-client/lib/acegi-security-1.0.4.jar	Acegi Security provides a comprehensive security solution for J2EE-based enterprise software applications (required by Spring)	http://www.acegisecurity.org/
remote-client/lib/antlr-2.7.6.jar	Another Tool for Language Recognition, is a language tool that provides a framework for constructing recognizers, interpreters, compilers, and translators from grammatical descriptions containing actions in a variety of target languages.	http://www.antlr.org/
remote-client/lib/asm.jar	An all purpose Java bytecode manipulation and analysis framework.	http://asm.objectweb.org/
remote-client/lib/cadsrapi40-beans.jar	The caDSR object beans.	http://cadsrapi.nci.nih.gov/cadsrapi40/docs/
remote-client/lib/castor-1.0.2.jar	Castor is an Open Source data binding framework for Java.	http://www.castor.org/
remote-client/lib/cglib-2.1.3.jar	A powerful, high performance and quality Code Generation Library, It is used to extend JAVA classes and implements interfaces at runtime.	http://cglib.sourceforge.net/
remote-client/lib/commons-codec-1.3.jar	Provides implementations of common encoders and decoders such as Base64, Hex, Phonetic and URLs.	http://commons.apache.org/codecs/

Table 4.1 Extracted Directories and Files in caDSR Client Package

Directories and Files	Description	Component
remote-client/lib/commons-collections-3.2.jar	Builds upon the JDK classes by providing new interfaces, implementations and utilities.	http://commons.apache.org/collections/
remote-client/lib/commons-logging-1.1.jar	An ultra-thin bridge between different logging implementations.	http://commons.apache.org/logging/
remote-client/lib/hibernate3.jar	A powerful, high performance object/relational persistence and query service.	http://www.hibernate.org/
remote-client/lib/log4j-1.2.14.jar	Open-source software related to the logging of application behavior.	http://logging.apache.org/log4j/
remote-client/lib/sdk-client-framework.jar	The caCORE SDK Client Framework.	http://ncicb.nci.nih.gov/core/SDK/4.0/
remote-client/lib/spring.jar	Enterprise Java framework	http://www.springframework.org/
remote-client/lib/xercesImpl.jar	A collaborative software development project dedicated to providing robust, full-featured, commercial-quality, and freely available XML parsers and closely related technologies on a wide variety of platforms supporting several languages.	http://xerces.apache.org/xerces-j/
remote-client/src/TestClient.java	Example test client source.	
remote-client/src/TestGetXMLClient.java	Example test client source.	
remote-client/src/TestXMLClient.java	Example test client source.	

Table 4.1 Extracted Directories and Files in caDSR Client Package

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

The caDSR Java API requires JDK SE 1.5.0 or higher and Ant 1.6.5 or higher. If used with other caCORE SDK bundles, SDK 4.0 must be used for caDSR 4.0 compatibility and the content of the /application-config-client.xml noted at the top of [Table 4.1](#) must be used and combined with the application-config-client.xml from all other SDK bundles.

A Simple Example

To run the simple example program after installing the caDSR client, open a command prompt or terminal window from the directory where you extracted the downloaded archive and enter `ant rundemo`. This will compile and run the TestClient class;

successfully running this example indicates that you have properly installed and configured the caDSR client.

The following is a short segment of code from the TestClient class, comments have been added for clarity.

```
// Connect to the caDSR API Remote Server, this uses the default
// Service Info defined in the application-config-client.xml
ApplicationService appService =
    ApplicationServiceProvider.getApplicationService();

// Get the Classes available in the package.
Collection<Class> classList = getClasses();
for(Class klass:classList)
{
    Object o = klass.newInstance();
    System.out.println("Searching for "+klass.getName());
    try
    {
        Collection results = appService.search(klass, o);
        for(Object obj : results)
        {
            printObject(obj, klass);

            // This stops after printing the first object in the
            // result set.
            break;
        }
    }
    catch(Exception e)
    {
        System.out.println(">>>" + e.getMessage());
    }
}
```

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

Application Service Interface

Connections to the caDSR API server are established by the `gov.nih.nci.system.applicationservice` package. The `ApplicationServiceProvider` class uses the factory design pattern to return an implementation of the `ApplicationService` interface. The `ApplicationServiceProvider` method can be classified into two method groupings. Some methods use the configured connection information in the `application-config-client.xml` and others allow the application to specify the service URL.

The separation of the service methods from the domain classes is an important architectural decision that insulates the domain object space from the underlying service framework.

A variety of methods are provided to allow users to query data based on the specific needs and types of queries to be performed. In general, there are five types of searches:

Simple searches—Those that take one or more objects from the domain models as inputs and return a collection of objects from the data repositories that meet the criteria specified by the input objects.

Nested searches—Also take domain objects as inputs but determine the type of objects in the result set by traversing a known path of associations from the domain model.

Detached criteria searches—Use Hibernate detached criteria objects to provide a greater level of control over the results of a search (such as boolean operations, ranges of values, etc.).

HQL searches—Provide the ability to use the Hibernate Query Language for the greatest flexibility in forming search criteria.

caBIG Query Object criteria searches—These have been modeled to the Object representation of caBIG Query Language (CQL). The CQL criteria searches use a syntax similar to the Query-by-Example (QBE) query language to specify the way results are to be retrieved. The system formulates the query based on the navigation path specified in the query search criteria. This query mechanism allows the user to search for objects using platform-independent query syntax.

Convenience Query Methods

getMaxRecordsCount()—Returns the maximum number of records the ApplicationService interface has been configured to return at one time.

getQueryRowCount(Object criteria, String targetClassName)—Returns the number of records that meet the search criteria. This method is used by the client framework to determine the number of list chunks in the result set. caDSR users can also invoke this method in conjunction with the getMaxRecordsCount() method; however, this is not required.

getAssociation(Object source, String associationName)—Retrieves an associated object for the example object specified by the source parameter. |

Note: The retrieved results from these methods are truncated to the maximum number of supported records indicated by the getMaxRecordsCount() method. Result sets with a total record count less than this maximum are complete.

HQL Query Methods

The Hibernate Query Language (HQL) is similar to SQL in syntax, however, HQL is still fully object-oriented and understands concepts like inheritance, polymorphism and association. The caDSR API contains a wrapper class called HQLCriteria, which is used for submitting HQL queries. For more information on the Hibernate Query Language, see http://www.hibernate.org/hib_docs/v3/reference/en/html/queryhql.html.

Note: The retrieved results from these methods are truncated to the maximum number of supported records indicated by the getMaxRecordsCount() method. Result sets with a total record count less than this maximum are complete.

query(HQLCriteria hqlCriteria)—This method retrieves the results obtained by querying the data source using HQL. As such, the data source must use Hibernate at

the persistence tier. Internally, Hibernate executes the HQL query against the relational database and retrieves the results.

query(HQLCriteria hqlCriteria, String targetClassName)—Deprecated. Internally calls the query (HQLCriteria hqlCriteria) method without the targetClassName parameter. |

The example below shows how an HQLCriteria object representing an HQL query might be instantiated and submitted, and how the results would be returned.

```
// Get Data Elements using HQL
ApplicationService appService =
    ApplicationServiceProvider.getApplicationService();

HQLCriteria hqlCrit = new HQLCriteria("from
    gov.nih.nci.cadsr.domain.DataElement de where de.longName='Adverse
    Event'");
Collection results = appService.query(hqlCrit);

for (Object obj : results)
{
    printObject(obj, DataElement.class);
    break;
}
```

Nested Search Criteria Query Methods

caDSR Nested Search Criteria queries have two parts: 1) a comma separated path to the target search object and 2) an example of the source object. The comma separated path starts with the target object (the fully qualified name of the class) to retrieve from the database. The next item in the comma-separated path is a link in the chain to an element (fully qualified name of the class) that connects the element on its left to the element on its right. The element on the right could be the example object or another element in the chain. The linked element provides a mechanism to traverse from the example object to the object that is desired using a comma separated path.

Note: The retrieved results from these methods are truncated to the maximum number of supported records indicated by the getMaxRecordsCount() method. Result sets with a total record count less than this maximum are complete.

search(String path, List<?> objList)—Retrieves the result from the data source using a Nested Search Criteria. The path specifies the list of objects (separated by commas), which should be used to reach the target object from the example objects passed in the objList, or the associated object for the example object. Internally, the Nested Search Criteria is converted into the data-source-specific query language. For the ORM-based persistence tier, the query structure is first converted into HQL. Hibernate then converts the HQL into SQL and executes it against the relational database.

search(Class targetClass, List<?> objList)—Retrieves the result from the data source using QBE. The targetClass specifies the object to fetch after executing the query. The targetClass should be the same as the object specified in the objList or associated object for the example object. All the objects in the objList have to be the same type. The example query is converted into the data-source-specific query language. For the ORM-based persistence tier, the example query structure is first

converted to a Nested Search Criteria, and then to HQL. Hibernate then converts the HQL into SQL and executes it against the relational database.

search(Class targetClass, Object obj)—Retrieves the result from the data source using QBE. The targetClass specifies the object that the user intends to fetch after executing the query. The targetClass should be same as the example object or associated object for the example object. The example query is first converted into the data source specific query language. For the ORM-based persistence tier, the example query structure is first converted to a Nested Search Criteria, and then to HQL. Hibernate finally converts the HQL into SQL and executes it against the relational database.

search(String path, Object obj)—Retrieves the result from the data source using the Nested Search Criteria. The path specifies the list of objects (separated by commas) which should be used to reach the target object from the example object passed as obj, or the associated object for the example object. Internally, the Nested Search Criteria is converted into the data source specific query language. For the Object Relational Mapping based persistence tier, the query structure is first converted into HQL. Hibernate then converts the HQL into SQL and executes it against the relational database.

The following example query demonstrates how to use the nested search criteria.

```
public void testSearch() throws Exception
{
    ApplicationService appService =
        ApplicationServiceProvider.getApplicationService();

    ValueDomain vd1 = new ValueDomain();
    ValueDomain.setPublicID(2015097);

    ValueDomain vd2 = new ValueDomain();
    ValueDomain.setPublicID(2015119);

    List<ValueDomain> vdCollection = new ArrayList<ValueDomain>();
    vdCollection.add(vd1);
    vdCollection.add(vd2);

    String path = "gov.nih.nci.cadsr.domain.DataElement,"
        + "gov.nih.nci.cadsr.domain.ValueDomain";

    Collection results = appService.search(path, vdCollection);
    System.out.println("Number of qualifying records: " +
        results.size());
    for (Object obj : results)
    {
        printObject(obj, DataElement.class);
    }
}
```

Detached Criteria Query

To build queries dynamically using an object-oriented API rather than building query strings, Hibernate provides the Detached Criteria. This extends the Criteria concept allowing queries to be created outside the scope of a session for execution later using some arbitrary Hibernate Session. For more information on Hibernate Criteria queries, see http://www.hibernate.org/hib_docs/v3/reference/en/html_single/#querycriteria.

Note: The retrieved results from these methods are truncated to the maximum number of supported records indicated by the `getMaxRecordsCount()` method. Result sets with a total record count less than this maximum are complete.

query(DetachedCriteria detachedCriteria)—Retrieves the result from the data source using the `DetachedCriteria` query object. The `DetachedCriteria` query structure can be used only by the ORM-based persistence tier. Hibernate executes it against the relational database and fetches the results.

query(DetachedCriteria detachedCriteria, String targetClassName)—Deprecated. Internally calls the `query(DetachedCriteria detachedCriteria)` method without the `targetClassName` parameter.

The following sample shows how a Hibernate `DetachedCriteria` object might be instantiated and the query submitted, and how the results would be returned.

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

DetachedCriteria dCrit =
    DetachedCriteria.forClass(DataElement.class).add(
        Property.forName("longName").eq("Adverse Event") );
Collection results = appService.query(dCrit);

for (Object obj : results)
{
    printObject(obj, DataElement.class);
    break;
}
```

CQL Query

CQL queries are modeled similarly to the object representation of the caBIG Query Language (CQL), which uses syntax similar to the Query-by-Example (QBE). Always begin with the `CQLQuery` object which specifies the object (target object) to be retrieved. The QBE object has space for 1) an attribute (`CQLAttribute`) 2) an association (`CQLAssociation`) and 3) a group (`CQLGroup`) of association collection and attributes collection.

For example, to search for an object with “zipcode” equal to “20852”, a `CQLObject` must be created with a `CQLAttribute` for Name “zipcode” and Value “20852”. A `CQLPredicate` of “EQUAL_TO” is required for comparison between `CQLAttribute` and the database value. The `CQLGroup` is not needed in this example. It would be used to include additional `CQLAttribute` and `CQLPredicate` objects for more complex queries.

The `CQLQuery` related `ApplicationService` query methods are highlighted below.

Note: The retrieved results from these methods are truncated to the maximum number of supported records indicated by the `getMaxRecordsCount()` method. Result sets with a total record count less than this maximum are complete.

query(CQLQuery cqlQuery)—Retrieves the query result from the data source using the CQL query syntax. Internally, CQL query structure is converted into HQL. Hibernate in turn converts the HQL into SQL and executes it against the relational database.

query(CQLQuery cq|Query, String targetClassName)—Deprecated. Internally calls the query(CQLQuery cq|Query) method without the targetClassName parameter.

The example below shows a CQL query for an attribute value.

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

CQLAttribute attr = CQLAttribute();
attr.setName("longName");
attr.setValue("Adverse Event");
attr.setPredicate(CQLPredicate.EQUAL_TO);

CQLObject target = new CQLObject();
target.setName("gov.nih.nci.cadsr.domain.DataElement");
target.setAttribute(attr);

CQLQuery cq = new CQLQuery();
cq.setTarget(target);

Collection results = appService.query(cq);

for (Object obj : results)
{
    printObject(obj, DataElement.class);
    break;
}
```

The next example shows a CQL query with CQLGroup and CQLAssociation objects.

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

CQLAssociation ascDEC = new CQLAssociation();
ascDEC.setName("gov.nih.nci.cadsr.domain.DataElementConcept");

CQLAttribute attrDEC = new CQLAttribute();
attrDEC.setName("publicID");
attrDEC.setValue(2013222);
attrDEC.setPredicate(CQLPredicate.EQUAL_TO);

ascDEC.setTargetRoleName("dataElementConcept");
ascDEC.setSourceRoleName("dataElementCollection");
ascDEC.setAttribute(attrDEC);

CQLAssociation ascVD = new CQLAssociation();
ascVD.setName("gov.nih.nci.cadsr.domain.ValueDomain");

CQLAttribute attrVD = new CQLAttribute();
attrVD.setName("publicID");
attrVD.setValue(2017439);
attrVD.setPredicate(CQLPredicate.EQUAL_TO);

ascVD.setTargetRoleName("valueDomain");
ascVD.setSourceRoleName("dataElementCollection");
ascVD.setAttribute(attrVD);

CQLGroup group = new CQLGroup();
group.addAssociation(ascDEC);
group.addAssociation(ascVD);
group.setLogicOperator(CQLLogicalOperator.AND);
```

```

CQLObject target = new CQLObject();
target.setName("gov.nih.nci.cadsr.domain.DataElement");
target.setGroup(group);

CQLQuery cq = new CQLQuery();
cq.setTarget(target);

Collection results = appService.query(cq);
System.out.println("Number of qualifying records: " + results.size());

for (Object obj : results)
{
    printObject(obj, DataElement.class);
}

```

Web Services API

The caDSR 4.0 Web Service Interface is based on the Axis 1.4 framework, which adheres to the J2EE 1.4 server programming model described by JAX-RPC and JSR 109 (that is, the caDSR 4.0 Web Services uses the Remote Procedure Call (RPC) Web Service style). For more information on web services, see the Introduction to Web Services Metadata: http://dev2dev.bea.com/pub/a/2004/10/Anil_WServices.html.

There are four “styles” of service in Axis. RPC services use the SOAP RPC conventions, and also the SOAP “section 5” encoding. Document services do not use any encoding (so in particular, you will not see multiref object serialization or SOAP-style arrays on the wire) but DO still do XML<->Java databinding. Wrapped services are just like document services, except that rather than binding the entire SOAP body into one big structure, they “unwrap” it into individual parameters. Message services receive and return arbitrary XML in the SOAP Envelope without any type mapping/data binding. For more information, see <http://ws.apache.org/axis/java/user-guide.html#ServiceStylesRPCDocumentWrappedAndMessage>.

Note: While the caDSR Web Service continues to be based on the Axis 1.4 framework, the extraneous .ws layer found in previous SDK versions has been eliminated. In addition, the caDSR Web Service Deployment Descriptor (WSDD) is now packaged along with the rest of the system, thus allowing for automatic deployment of the Web Service (meaning manual deployment of the Web Service is no longer required).

A sample test program, TestClient.java, illustrates how the caDSR generated Web Service can be consumed, and is provided in the `output\example\package\ws-client\src` folder. More information about this test program and Web Services in general is provided in the [caCORE SDK 4.0 Developers Guide](#).

XML-HTTP API

All caCORE SDK based systems, including the caDSR API, support an XML-HTTP API, based on the Representational State Transfer (REST) architectural style. This can be invoked from most internet browsers and developers can use this interface to build applications that do not require any programming overhead other than an HTTP client. This is particularly useful for developing web applications using AJAX (asynchronous JavaScript and XML). The REST interface provided transmits domain-specific data

over HTTP without an additional messaging layer, such as SOAP, or session tracking via HTTP cookies. For more information on REST, see <http://en.wikipedia.org/wiki/REST>.

To invoke this interface for the caDSR, use a URL in the form.

```
http://cadsrapi.nci.nih.gov/cadsrapi40/
  GetXML?query=<target>&<criteria>[&rolename=<rolename>]
```

Parameter	Description
Target	target/result class name, e.g. gov.nih.nci.cadsr.domain.DataElement
Criteria	A string identifying the qualified or non-qualified criteria class name to be used as a filter/constraint on the result set. If desired, the value of the id attribute of the criteria class instance can also be supplied in order to further constrain the result set. <criteria_class_name>[@id=<id_value>] e.g. gov.nih.nci.cadsr.domain.DataElement[@longName=Adverse%20Event]
Rolename	The name of the attribute within the criteria class that identifies the association to be traversed when retrieving the target/result class(es). The rolename property must be specified whenever the Criteria class has two or more attributes representing associations to the same target/result class type.

Table 4.2 XML-HTTP API Parameters

A complete example.

```
http://cadsrapi.nci.nih.gov/cadsrapi40/
  GetXML?query=gov.nih.nci.cadsr.domain.DataElement&gov.nih.nci
  .cadsr.domain.DataElement[@longName=Adverse%20Event]
```

While such a URL can be invoked directly from a browser, it is most frequently done so programmatically via a remote client program. An example of such a program, TestGetXMLClient.java, is noted above.

UNDERSTANDING UNIFIED MODELING LANGUAGE (UML)

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

Note: 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 (<http://www.omg.org/>), the UML emerged as the result of several complementary systems of software notation and has now become the de facto standard for visual modeling. For a brief tutorial on UML, refer to <http://bdn.borland.com/article/0,1410,31863,00.html>.

The underlying tenet of any object-oriented programming begins with the construction of a model. In its entirety, the UML version 1.3 is composed of nine different types of modeling diagrams that form, in essence, a software blueprint. Only the following subset of diagrams, those used in caCORE development, are described in this document.

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

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

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

Note: Not all UML artifacts discussed here are necessary for using caDSR. They are included strictly to provide a more complete overview of UML.

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

Use-case Documents and Diagrams

A good starting point for capturing system requirements is to develop a structured textual description of how users will interact with the system, often called a use-case document. While there is no specific 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.

The following is an example use case from a 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.

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

The example use-case diagram in *Figure A.1* below can be interpreted as follows:

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

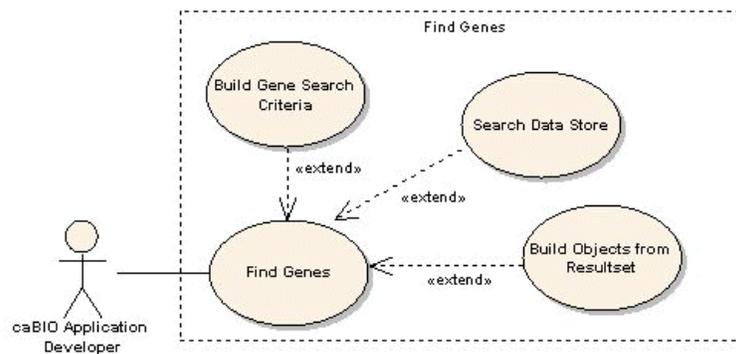


Figure A.1 Example Use Case Diagram

Class Diagrams

The system designer utilizes use-case diagrams to identify the classes that must be implemented in the system, their attributes and behaviors, and the relationships and cooperative activities that must be realized. A class diagram is used later in the design process to give an overview of the system, showing the hierarchy of classes and their

static relationships at varying levels of detail. *Figure A.2* shows an abbreviated version of a UML Class diagram depicting many of the caBIO domain objects.

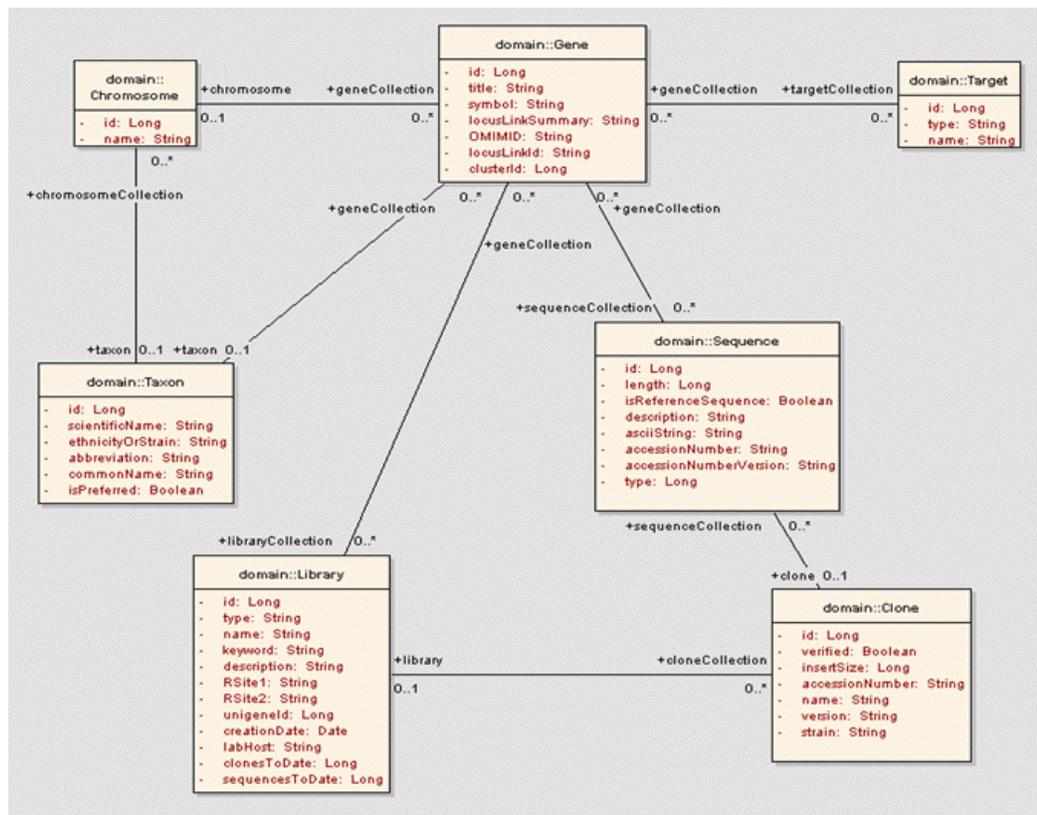


Figure A.2 Example UML Class Diagram Depicting the caBIO Domain Objects

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

Table A.1 shows a schematic for a UML class representation, the fundamental element of a class diagram, along with an example of how a simple class might be represented in this scheme.

The example shows the name of the class (often used as the identifier for the class), the attributes (structures) for the class, and the object’s operations (methods). The example specifies the *Gene* class as having a single attribute called *sequence* and a single operation called *getSequence()*.

Class	Gene
-attribute	-sequence
Operation	+getSequence()

Table A.1 Schematic for a UML class showing a simple class called Gene

Naming Conventions

Naming conventions are important when creating class diagrams. caDSR 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 the UML diagram in [Figure A.2](#) 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 that the diagram is intended to convey. Most UML design tools provide a 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.2](#), the emphasis is on the relationships and attributes that are defined among the objects rather than on the operations.

The following notations (as shown in [Figure A.2](#) above and [Figure A.6](#) later) are used to indicate that a feature is public or private:

- “-” prefix signifies a private feature
- “+” signifies a public feature

In [Table A.1](#) 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 here appear in the demonstration class diagram, [Figure A.2](#). They are, however, examples of models that may be found in caCORE.

A quick glance at [Figure A.2](#) above 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. Each of these are described in the sections that follow.

Association

The most primitive of relationships between objects is association. An association 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 shown in [Figure A.3](#) below. 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 (i.e., there is a corresponding `getGeneCollection()` method). Role names are also shown in [Figure A.2](#) above and [Figure A.3](#) below, clarifying the nature of the association between the two classes. For example, a taxon (rolename identified in [Figure A.3](#)) is a line item of each Gene object. The + indicates public accessibility.

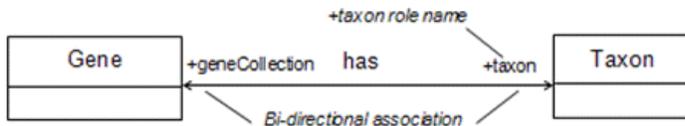


Figure A.3 One-to-one Association with Bi-directional Navigability

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. This notation indicates multiplicity. These cardinality constraints identify that the relationship between the objects is one-to-one, one-to-many, many-to-one, or many-to-many, according to the ranges specified and their placement. [Table A.2](#) below lists the most commonly used multiplicities.

Multiplicities	Definition
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 (8) is used to represent a multiplicity of many.
1	Exactly one instance
1..*	At least one instance to many

Table A.2 Multiplicities Table

[Figure A.4](#) below depicts a bi-directional many-to-one relationship between Sequence objects and Clone objects. Each Sequence may have at most one Clone associated with it, while a Clone may be associated with many Sequences. To get information about a Clone from the Sequence object requires calling the `getSequenceClone()` method. Each Clone in turn can return its array of associated Sequence objects using the `getSequences()` method. This bidirectional relationship is shown using a single undirected line between the two objects.



Figure A.4 Bi-directional Many-to-one Relationship

Aggregation

Another relationship exhibited by caDSR 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.5](#). As illustrated, a Library can contain Clones but not vice-versa.

In UML, the empty diamond of aggregation indicates 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.

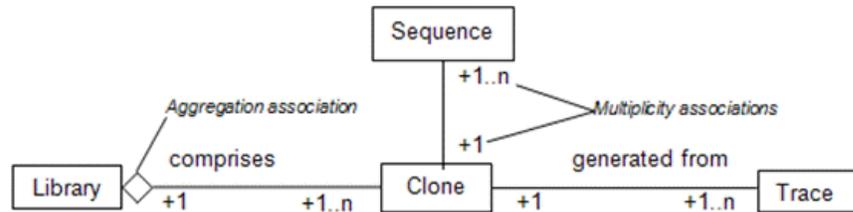


Figure A.5 Aggregation and Multiplicity Associations

[Figure A.5](#) also shows a more complex network of relations. Specifically this diagram indicates that:

- One or more Sequences is associated with a Clone.
- The Clone is contained in a Library, which comprises one or more Clones.
- 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.6](#) below 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.6](#).

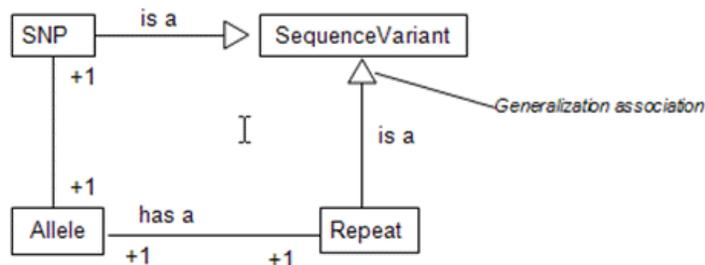


Figure A.6 Generalization Relationship

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

Package Diagrams

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

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

A package is depicted as a labeled rectangle with a small tab attached to its upper left corner, somewhat resembling a file folder (see [Figure A.7](#) below). This image represents a package diagram generated using the UML modeling program Enterprise Architect (EA). In our example, “gov” is the top level package; “nih” is a sub-package to “gov” with the + indicating that sub-packages to “nih” exist.

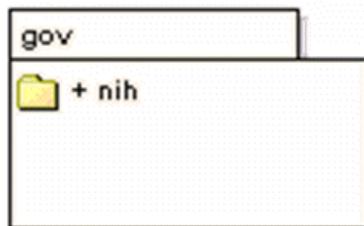


Figure A.7 Package Diagram Generated in EA

The dotted arrows connecting packages as displayed in [Figure A.8](#) represent dependencies: one package depends on another if changes in one could force changes in the other. [Figure A.8](#) below is the hierarchical representation of [Figure A.7](#).

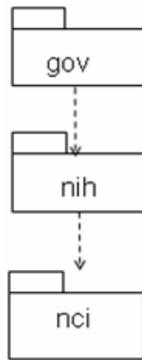


Figure A.8 Hierarchical Package Diagram

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 caDSR software that are available on the JavaDocs pages approximately reflect the actual Java package structure but do not have a one-to-one correspondence.



Figure A.9 Generic Component Diagram

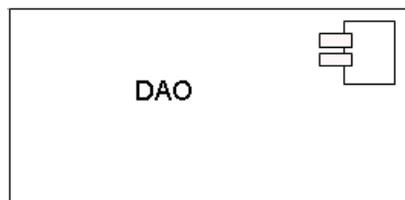


Figure A.10 Component Diagram as Represented in EA

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

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

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

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

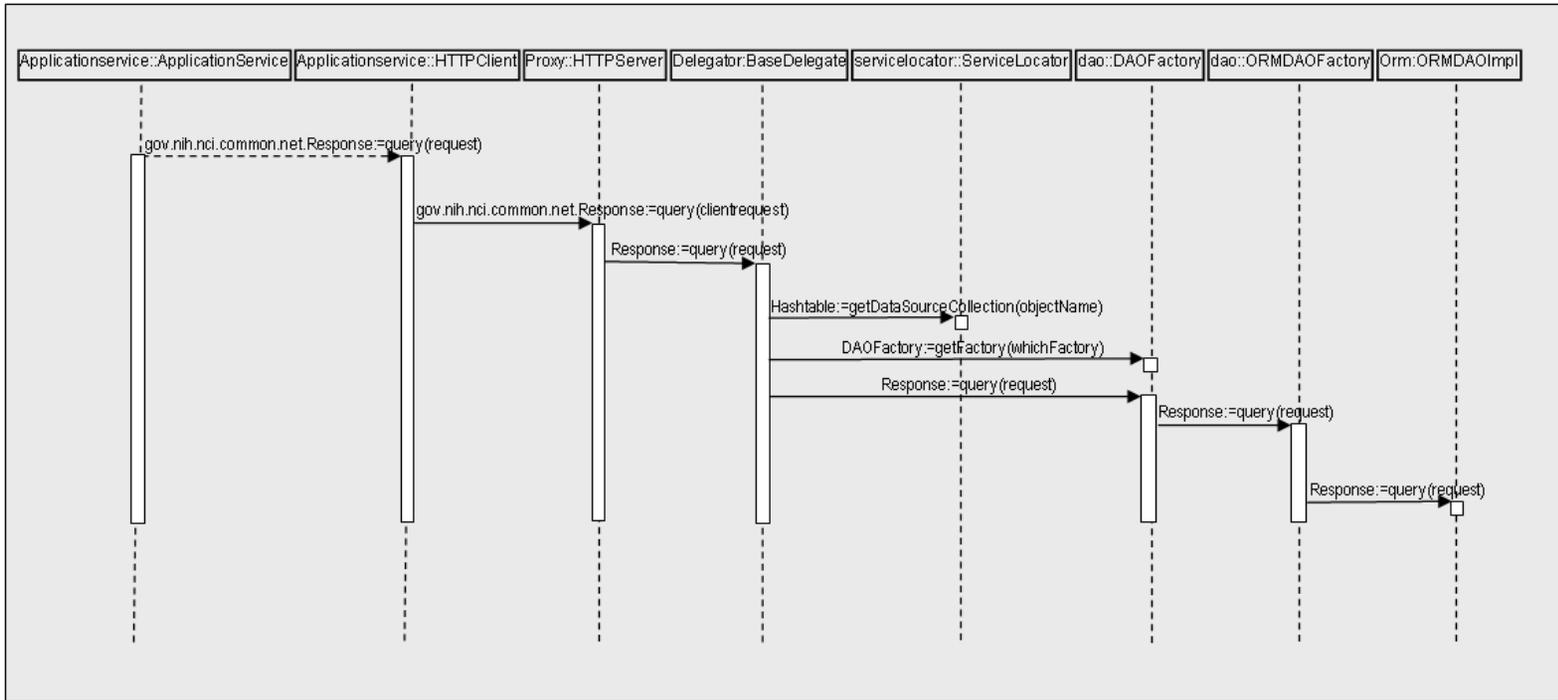


Figure A.11 Example of a Sequence Diagram

APPENDIX B REFERENCES

This appendix contains a listing of references relevant to caDSR 4.0 and the caDSR 4.0 Technical Guide.

Technical Manuals/Articles

- [National Cancer Institute caCORE SDK 4.0 Programmer's Guide](#)
- [Java Bean Specification](#)
- [Foundations of Object-Relational Mapping](#)
- [Object-Relational Mapping articles and products](#)
- [Hibernate Reference Documentation](#)
- [Basic O/R Mapping](#)
- [Java Programming](#)
- [Jalopy User Manual](#)
- [Javadoc tool](#)
- [JUnit](#)
- [Extensible Markup Language](#)
- [XML Metadata Interchange](#)
- [Ehcache](#)

Scientific Publications

- Komatsoulis, G.A., Warzel, D.B., Hartel, F.W., Shanbhag, K, Chilukuri, R, Fragoso, G., de Coronado, S, Reeves, D.M., Hadfield, J.B., Ludet, C., and P.A. Covitz (2007) caCORE version 3: Implementation of an model driven, service-

- oriented architecture for semantic interoperability. Accepted, BMC Medical Informatics and Decision Making.
- Covitz P, Warzel D, Fragoso G, Chilukuri R, Phillips J, The caCORE Software Development Kit: Streamlining construction of interoperable biomedical information services, BMC Medical Informatics and Decision Making 6:2
 - Crowley R, Wright L, Warzel D, Sioutos N, Mohanty S, Komatsoulis G, Chilukuri R, Tobias J, The CAP cancer protocols - a case study of caCORE based data standards implementation to integrate with the Cancer Biomedical Informatics Grid, BMC Medical Informatics and Decision Making (June 20 2006) 6:25
 - 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/>)
 - Ansher SS and Scharf R (2001). The Cancer Therapy Evaluation Program (CTEP) at the National Cancer Institute: industry collaborations in new agent development. Ann N Y Acad Sci 949:333-40.
 - Boon K, Osorio EC, Greenhut SF, Schaefer CF, Shoemaker J, Polyak K, Morin PJ, Buetow KH, Strausberg RL, De Souza SJ, and Riggins GJ (2002). An anatomy of normal and malignant gene expression. Proc Natl Acad Sci U S A 2002 Jul 15.
 - Buetow KH, Klausner RD, Fine H, Kaplan R, Singer DS, and Strausberg RL (2002). Cancer Molecular Analysis Project: Weaving a rich cancer research tapestry. Cancer Cell 1(4):315-8.
 - Clifford R, Edmonson M, Hu Y, Nguyen C, Scherpbier T, and Buetow KH (2000). Expression-based genetic/physical maps of single-nucleotide polymorphisms identified by the Cancer Genome Anatomy Project. Genome Res 10(8):1259-65.
 - 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.
 - Dowell RD, Jokerst RM, Day A, Eddy SR, Stein L. The Distributed Annotation System. BMC Bioinformatics 2(1):7.
 - The Gene Ontology Consortium. (2000). Gene ontology: tool for the unification of biology. Nature Genetics 25:25-9.
 - The Gene Ontology Consortium. (2001). Creating the gene ontology resource: design and implementation. Genome Res 11:1425-33.
 - Golbeck J., Fragoso G., Hartel F., Hendler J., Oberthaler J., Parsia B. (2003). The National Cancer Institute's th_saurus and ontology. Journal on Web Semantics. 1:75-80.
 - Hartel FW and de Coronado S (2002). Information standards within NCI. In: *Cancer Informatics: Essential Technologies for Clinical Trials*. Silva JS, Ball MJ, Chute CG, Douglas JV, Langlotz C, Niland J and Scherlis W, eds. Springer-Verlag.

caBIG Material

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

Table B.1 caBIG Material

caCORE Material

caCORE	http://ncicb.nci.nih.gov/NCICB/infrastructure/cacore_overview
caBIO	http://ncicb.nci.nih.gov/NCICB/infrastructure/cacore_overview/caBIO
caDSR	http://ncicb.nci.nih.gov/NCICB/infrastructure/cacore_overview/cadsr
EVS	http://ncicb.nci.nih.gov/NCICB/infrastructure/cacore_overview/vocabulary
CSM	http://ncicb.nci.nih.gov/NCICB/infrastructure/cacore_overview/csm

Table B.2 caCORE Material

Modeling Concepts

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

Table B.3 Modeling Concept References

Applications Currently Using caCORE

BIO Browser	http://www.jonnywray.com/java/index.html
caPathway	http://cgap.nci.nih.gov/Pathways

Table B.4 Products Using caCORE

Software Products

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

Table B.5 Software Products

GLOSSARY

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

Term	Definition
Apache Axis	Open source package that provides SOAP-based web services to users
API	Application Programming Interface
Application Service	This refers to the CSM interface which exposes all the writeable as well as business methods for a particular application
BO	Business Object
C3D	Cancer Centralized Clinical Database
caBIG	cancer Biomedical Informatics Grid
caBIO	Cancer Bioinformatics Infrastructure Objects
caCORE	Cancer Common Ontologic Representation Environment
caDSR	Cancer Data Standards Repository
caMOD	Cancer Models Database
cardinality	Cardinality describes the minimum and maximum number of associated objects within a set
CCR	Center of Cancer Research
CDE	Common Data Element
CGAP	Cancer Genome Anatomy Project
CLM	Common Logging Module
CMAF	Cancer Molecular Analysis Project
CS	Classification Scheme
CSI	Classification Scheme Item
CLM	Common Logging Module
CSM	Common Security Module
CTEP	Cancer Therapy Evaluation Program
CVS	Concurrent Versions System

Term	Definition
DAO	Data Access Objects
DAS	Distributed Annotation System
DCP	Division of Cancer Prevention
DDL	Data Definition Language
DEC	Data Element Concept
DL	Description Logic
DOM	Document Object Model
DTD	Document Type Definition
DTS	Distributed Terminology Server
DU	Deployment Unit
EA	Enterprise Architect
EBI	European Bioinformatics Institute
EMF	Eclipse Modeling Framework
EVS	Enterprise Vocabulary Services
FreeMarker	A "template engine"; a generic tool to generate text output (anything from HTML or RTF to auto generated source code) based on templates
GAI	CGAP Genetic Annotation Initiative
GEDP	Gene Expression Data Portal
Hibernate	A high performance object/relational persistence and query service for JavaProvides the ability to develop persistent classes following common object-oriented (OO) design methodologies such as association, inheritance, polymorphism, and composition (http://www.hibernate.org)
HQL	Hibernate Query Language is designed as a "minimal" object-oriented extension to SQL, provides a bridge between the object and relational databases
IDE	Integrated Development Environment
ISO	International Organization for Standardization
JAR	Java Archive
Java Bean	Reusable software components that work with Java
Java Servlet	Server-side Java programs, that web servers can run to generate content in response to client requests
Javadoc	Tool for generating API documentation in HTML format from doc comments in source code (http://java.sun.com/j2se/javadoc/)
JBoss	J2SE application server used as a a presentation layer in caCORE architecture. See also Tomcat.
JDBC	Java Database Connectivity

Term	Definition
JDiff	Javadoc doc-let which generates an HTML report of all the packages, classes, constructors, methods, and fields which have been removed, added or changed in any way, including their documentation, when two APIs are compared (http://javadiff.sourceforge.net/)
JET	Java Emitter Templates
JMI	Java Metadata Interface
JSP	Java Server Pages. Web pages with Java embedded in the HTML to incorporate dynamic content in the page
JUnit	A simple framework to write repeatable tests (http://junit.sourceforge.net/)
MDR	Metadata Repository
metadata	Definitional data that provides information about or documentation of other data.
MMHCC	Mouse Models of Human Cancers Consortium
multiplicity	Multiplicity of an association end indicates the number of objects of the class on that end that may be associated with a single object of the class on the other end
MVC	Model-View-Controller, a design pattern
navigability	Navigability defines the visibility of an object to its associated source/target object at the other end of an association. Navigability is the same as directionality.
NCI	National Cancer Institute
NCICB	National Cancer Institute Center for Bioinformatics
NSC	Nomenclature Standards Committee
OMG	Object Management Group
OR	Object Relation
ORM	Object Relational Mapping
PCDATA	Parsed Character DATA
persistence layer	Data storage layer, usually in a relational database system
RDBMS	Relational Database Management System
RUP	Rational Unified Process
SOAP	Simple Object Access Protocol. A lightweight XML-based protocol for the exchange of information in a decentralized, distributed environment
SPORE	Specialized Programs of Research
SQL	Structured Query Language
Tagged value	A UML construct that represents a name-value pair; can be attached to anything in a UML model. Often used by UML modeling tools to store tool-specific information

Term	Definition
Tomcat	J2SE application server used as a a presentation layer in caCORE architecture. See also JBoss.
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locators
war	Web archive file
Writeable API	Methods exposed by the CSM to create, update and delete a domain object. These methods are generated using the code generation component.
WSDL	Web Services Description Language
XMI	XML Metadata Interchange (http://www.omg.org/technology/documents/formal/xmi.htm) - The main purpose of XMI is to enable easy interchange of metadata between modeling tools (based on the OMG-UML) and metadata repositories (OMG-MOF) in distributed heterogeneous environments
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

A

- administered components 17, 19
- aggregation 30, 57, 58
- AJAX 51
- Apache Axis 10, 11, 36, 51, 69
- API 11, 23, 41
 - caCORE client Java API example 37
 - class hierarchy 25
 - examples 34, 36, 37
 - installation 42
 - Java example 34, 45
 - requirements 41
 - web services example 36
- application service interface 45
- application service layer 10
- architecture layers 10
- associations 20, 23, 31, 56, 57

B

- bi-directional 58

C

- caAdapter 7
- caBIG query language 49
- caBIG query language (CQL) 46
- caBIG silver-level compatibility 6
- caCORE 5
 - caCORE-like system 5
 - characteristics 5
 - components 6
- caCORE client Java API 37
- caDSR
 - API 11, 23, 41
 - API examples 34, 36, 37
 - architecture 9
 - client 10
 - data access package 11
 - domain model 9
 - domain objects 25
 - downloading 33, 42

- model extensions 30
- overview 6
- packages 11
- service layers 10, 41
- services 10
- system package 11
- toolset 33
- UML project diagram 32
- UML project domain objects 33
- web service package 11
- cardinality 15
- cardinality constraints 16
- CBIIT 5
- class attributes 19
- class descriptions 25
- class diagram 55, 56, 60, 61
- class hierarchy 24, 25, 55
- classification scheme 19, 20
- classification scheme items 21
- class naming conventions 57
- class relationships 57
- client.jar 10
- client access 41
- client authentication 10
- client test demo 45
- CLM 7
- clones 59
- component diagram 61
- concept 29
- concept derivation 30
- concept derivaton rule 30
- conceptual domain 14, 15
- controlled vocabularies 5, 6
- convenience query methods 46
- CQL criteria search 46
- CQL queries 49
- CQL query methods 49
- CSM 7

D

- data access package 11
- data element 14, 15, 19
 - latest version 34, 36
- data element concept 14, 15
- data management 14
- data model diagram 15
- data source delegation layer 10
- designation 20
- detached criteria query methods 49
- detached criteria search 46, 48
- directionality 57
- domain model 9
- domain object 9, 25, 33
- domain-specific elements 21
- downloaded files 42
- downloading caDSR 33
- downloading the API 42

E

- Enterprise Architect (EA) 54, 60
- EVS 6

F

- form registration 14
- forms 21
- forms components 22

G

- generalization 57, 59
- getAssociation 46
- getMaxRecordsCount 46
- getQueryRowCount 46

H

- Hibernate 10
- Hibernate query 46
- Hibernate Query Language 10
- HQL 10, 46
- HQLCriteria 46
- HQL Query Methods 46

I

- information classes 17
- information mapping diagram 16
- inheritance 59
- installing the API 42
- ISO/IEC 11179 13
- ISO/IEC 11179 components 14
- ISO/IEC 11179 model diagram 15

J

- jar files 44
- Java API 34, 44
- Java applications 9
- JavaDocs 11

L

- library 59
- longName 19

M

- metadata
 - associations 20, 23
 - cardinality 15
 - cardinality constraints 16
 - class diagram 15
 - conceptual domain 15
 - data element 14, 15
 - data element concept 15
 - definition 14
 - forms 21
 - framework 14
 - information mapping 16
 - model 14, 17
 - model diagram 16
 - object 14
 - overview 13
 - property 14
 - representation 14
 - standard 13
 - value 14
 - value domain 15, 25
- model extensions 30
- modeling 53
- multiplicity 57, 58

N

- navigability 58
- NCICB 5
- NCICB downloads 33
- nested search 46
- nested search criteria 47
- nested search query methods 47, 48
- non-Java applications 10
- n-tier architecture 5, 6, 9

O

- object 14
- object classes 31
- object management group 53
- object relational mapping (ORM) 10

OMG 53
ORM 10, 11
overview
 caAdapter 7
 caDSR 6
 CLM 7
 CSM 7
 EVS 6
 SDK 6

P

package diagram 61
package diagrams 60
packages 11
PreferredName 19
property 14
public vs. private 57

Q

query-by-example (QBE) 41, 46
query criteria 41

R

reference document 20
registered metadata 5
representation 14
representational state transfer 51
REST 51

S

sample demo program 44
sample query 34, 36, 37
SDK 6
search methods 45
semantic interoperability 6
sequence diagram 62
sequences 59
service layers 10
simple search 46
SOAP 10, 52
SOAP RPC 51
SQL 10
system package 11

T

TestClient.java 51
traces 59
truncated results 46, 49

U

UML 53
UML Class diagram 15
UML class representation 56
UML generalization 23
UML modeling 53
UML package 61
UML project diagram 32
UML project domain objects 33
UML project query 37
Unified Modeling Language 53
use-case
 analysis 54
 diagram 55
 document 54

V

value 14
value domain 14, 15, 25

W

web service package 11
web services API 36, 51
 sample test program 51

X

XML 10
XML-HTTP API 51
XML-HTTP example 52

